BỘ GIÁO DỤC VÀ ĐÀO TẠO

CHUYÊN ĐỀ HỌC TẬP
TIN HỌC 12
ĐỊNH HƯỚNG KHOA HỌC MÁY TÍNH



NHÀ XUẤT BẢN GIÁO DỤC VIỆT NAM

HỘI ĐỒNG QUỐC GIA THẨM ĐỊNH SÁCH GIÁO KHOA

Môn: Tin học – Lớp 12

(Theo Quyết định số 1882/QĐ-BGDĐT ngày 29 tháng 6 năm 2023 của Bộ trưởng Bộ Giáo dục và Đào tạo)

Chủ tịch: LÊ HOÀI BẮC
Phó Chủ tịch: TRẦN ĐĂNG HƯNG
Uỷ viên, Thư kí: HỒ VĨNH THẮNG

Các uỷ viên:
NGUYỄN TRUNG TRỰC – TRẦN CAO ĐỆ
QUÁCH XUÂN TRƯỞNG – ĐỖ TRUNG KIÊN
NGUYỄN THỊ VÂN KHÁNH – PHAN THỊ MAY
HOÀNG VĂN QUYẾN – HOÀNG XUÂN THẮNG

PHẠM THẾ LONG (Tổng Chủ biên) – BÙI VIỆT HÀ (Chủ biên)
NGUYỄN HOÀNG HÀ – LÊ HỮU TÔN

CHUYÊN ĐỀ HỌC TẬP
TIN HỌC
12
ĐỊNH HƯỚNG KHOA HỌC MÁY TÍNH

(Tái bản lần thứ nhất)

NHÀ XUẤT BẢN GIÁO DỤC VIỆT NAM

Hướng dẫn sử dụng sách

MỤC TIÊU Giúp em biết sẽ đạt được gì sau bài học.



KHỞI ĐỘNG Giúp em nhận biết ý nghĩa của bài học bằng cách kết nối những tình huống xuất hiện trong cuộc sống với nội dung bài học.

NỘI DUNG BÀI HỌC

Các hoạt động: Giúp lớp học tích cực, bài học dễ tiếp thu, học sinh chủ động hơn trong quá trình nhận thức.



Kiến thức mới: Cung cấp cho học sinh nội dung chính của bài học, giúp em bổ sung kiến thức nhằm đạt được mục tiêu của bài học.

Hộp kiến thức: Ghi ngắn gọn hoặc tóm tắt kiến thức mới. Em có thể dùng hộp kiến thức, cùng với bảng giải thích thuật ngữ (ở cuối sách), để ôn tập hoặc tra cứu thuật ngữ mới.



Câu hỏi: Giúp em kiểm tra xem mình đã hiểu bài chưa.

KẾT NỐI TRI THỨC VỚI CUỘC SỐNG



THỰC HÀNH Gồm những bài tập dưới dạng nhiệm vụ có hướng dẫn chi tiết.



LUYỆN TẬP Gồm những câu hỏi, bài tập để củng cố kiến thức, kĩ năng trong bài học.



VẬN DỤNG Gồm những câu hỏi, bài tập yêu cầu em kết hợp nội dung bài học với thực tiễn cuộc sống.

Hãy bảo quản, giữ gìn sách giáo khoa để dành tặng các em học sinh lớp sau!

Lời nói đầu

Các em thân mến!

Cuốn sách *Chuyên đề học tập Tin học 12 – Định hướng Khoa học máy tính* thuộc bộ sách *Kết nối tri thức với cuộc sống* được dành cho các em đăng kí học chuyên đề học tập Tin học 12 theo định hướng Khoa học máy tính.

Sách chuyên đề này nhằm cung cấp các kiến thức, kĩ năng Tin học 12 được xác định trong Chương trình Giáo dục phổ thông năm 2018, bao gồm một số cấu trúc dữ liệu cơ bản, quan trọng mà mỗi người làm khoa học máy tính cần phải biết trong tương lai.

Sách bao gồm ba chuyên đề:

– Chuyên đề 1: Giới thiệu hai cấu trúc dữ liệu: *ngăn xếp* (stack) và *hàng đợi* (queue). Các cấu trúc dữ liệu này, cùng với cấu trúc danh sách liên kết và mảng (đã được học trong chương trình lớp 11) là những cấu trúc dữ liệu tuyến tính đơn giản nhất, được sử dụng nhiều trong thực tế.

– Chuyên đề 2: Các em sẽ được tìm hiểu một cấu trúc dữ liệu có nhiều ứng dụng là *cấu trúc dữ liệu cây* (tree). Nội dung chính của chuyên đề tập trung vào *cây nhị phân* (binary tree) và *cây tìm kiếm nhị phân* (binary search tree). Các em sẽ thấy được cấu trúc cây tìm kiếm nhị phân có vai trò hết sức quan trọng trong giải quyết các bài toán tìm kiếm và sắp xếp.

– Chuyên đề 3: Các em sẽ được làm quen bước đầu với *đồ thị* (graph), một cấu trúc dữ liệu đặc biệt, rất phổ biến và có nhiều ứng dụng trong thực tế. Các em sẽ được học cách biểu diễn các đơn đồ thị vô hướng, có hướng và hai thuật toán duyệt đồ thị là duyệt theo chiều sâu (DFS) và duyệt theo chiều rộng (BFS).

Các bài học thường được bắt đầu bằng những ví dụ cụ thể, qua đó từng bước dẫn dắt các em đến với kiến thức cần nắm vững, phần cuối bài là các bài luyện tập và vận dụng.

Chúc các em có những giờ phút học tập đầy hứng khởi và thú vị, sáng tạo với cuốn sách này.

CÁC TÁC GIẢ

Mục lục

	Trang
CHUYÊN ĐỀ 1. TÌM HIỂU MỘT VÀI KIỂU DỮ LIỆU TUYẾN TÍNH	**5**
Bài 1. Mô hình dữ liệu ngăn xếp và hàng đợi	5
Bài 2. Kiểu dữ liệu ngăn xếp	9
Bài 3. Thực hành kiểu dữ liệu ngăn xếp	13
Bài 4. Kiểu dữ liệu hàng đợi	16
Bài 5. Thực hành kiểu dữ liệu ngăn xếp và hàng đợi	20
CHUYÊN ĐỀ 2. TÌM HIỂU CÂY TÌM KIẾM NHỊ PHÂN TRONG SẮP XẾP VÀ TÌM KIẾM	**23**
Bài 6. Cây nhị phân	23
Bài 7. Cây tìm kiếm nhị phân	30
Bài 8. Thực hành cây tìm kiếm nhị phân	37
Bài 9. Các thuật toán duyệt trên cây tìm kiếm nhị phân	41
Bài 10. Thực hành tổng hợp với cây tìm kiếm nhị phân	46
CHUYÊN ĐỀ 3. TÌM HIỂU KĨ THUẬT DUYỆT ĐỒ THỊ VÀ ỨNG DỤNG	**49**
Bài 11. Khái niệm đồ thị	49
Bài 12. Biểu diễn đồ thị	56
Bài 13. Thực hành thiết lập đồ thị	62
Bài 14. Kĩ thuật duyệt đồ thị theo chiều sâu	65
Bài 15. Thực hành duyệt đồ thị theo chiều sâu	72
Bài 16. Kĩ thuật duyệt đồ thị theo chiều rộng	75
Bài 17. Thực hành duyệt đồ thị tổng hợp	80
BẢNG GIẢI THÍCH THUẬT NGỮ	**83**

KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

Chuyên đề
1

TÌM HIỂU MỘT VÀI KIỂU DỮ LIỆU TUYẾN TÍNH

BÀI 1 MÔ HÌNH DỮ LIỆU NGĂN XẾP VÀ HÀNG ĐỢI

Sau bài học này em sẽ:

- Biết và mô tả được mô hình dữ liệu ngăn xếp và hàng đợi thông qua cơ chế hoạt động của các kiểu dữ liệu này.

Em hãy quan sát các hình ảnh về đồ vật và hiện tượng trong thực tế trong Hình 1.1 và cho biết:

a) Trong chồng đĩa, đĩa nào được xếp vào sau cùng? Đĩa nào cần được lấy ra đầu tiên?

b) Ai sẽ là người được rút tiền trước tại cây ATM? Người xếp hàng cuối cùng sẽ được rút tiền khi nào?



a) Chồng đĩa



b) Xếp hàng rút tiền tại ATM

Hình 1.1. Một số đồ vật và hiện tượng trong thực tế

1. Mô hình dữ liệu ngăn xếp

Hoạt động 1 Tìm hiểu mô hình dữ liệu ngăn xếp

Đọc, trao đổi và thảo luận để hiểu về mô hình dữ liệu ngăn xếp và cơ chế hoạt động "vào sau, ra trước" (LIFO – Last In, First Out) của mô hình dữ liệu này.

Trong các ví dụ thực tế ở phần khởi động, ví dụ ở Hình 1.1a thuộc mô hình dữ liệu ngăn xếp (stack). Có thể hiểu ngăn xếp là đối tượng dữ liệu, trong đó việc đưa dữ liệu vào và lấy dữ liệu ra ở cùng một đầu, theo cơ chế hoạt động LIFO.

Một ví dụ khác về ngăn xếp là chức năng Undo thường được sử dụng khi soạn thảo văn bản. Lệnh Undo giúp người sử dụng huỷ bỏ kết quả của thao tác gần nhất.

KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

Mô hình quản lí các dữ liệu Undo chính là ngăn xếp. Mỗi khi thực hiện một thao tác mới, trạng thái hiện tại của văn bản được đưa vào đỉnh của ngăn xếp Undo. Khi yêu cầu Undo, trạng thái hiện tại được lấy ra từ đỉnh ngăn xếp và khôi phục lại.

Mô hình dữ liệu ngăn xếp được mô tả như sau:

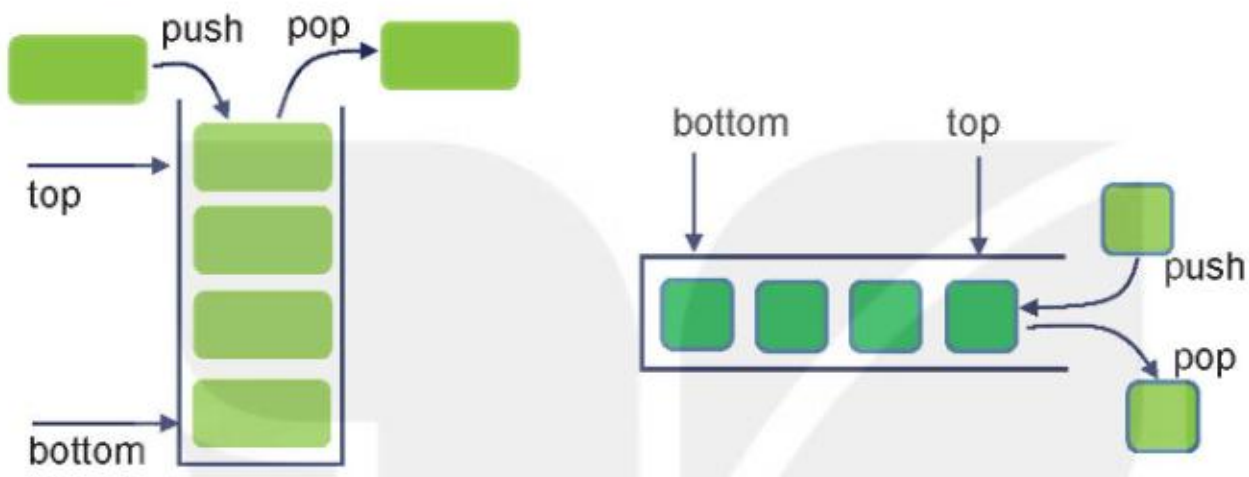
– Ngăn xếp là một dãy tuyến tính các phần tử dữ liệu.

– Ngăn xếp có các thao tác đưa phần tử vào và lấy phần tử ra tại cùng một đầu của ngăn xếp. Thao tác đưa dữ liệu vào là push và lấy dữ liệu ra là pop.

– Quy ước đầu dùng để đưa dữ liệu vào và lấy dữ liệu ra là đỉnh (top) của ngăn xếp. Đầu ngược lại là đáy (bottom) của ngăn xếp.

– Mô hình dữ liệu ngăn xếp hoạt động theo cơ chế LIFO.

Mô hình ngăn xếp với cơ chế hoạt động LIFO có thể được biểu diễn như sau:



Hình 1.2. Mô hình ngăn xếp

Sau đây là bảng các thao tác cơ bản trên dữ liệu ngăn xếp. Có thể thiết lập các lệnh thực hiện những thao tác này đều có độ phức tạp thời gian `O(1)`, tức là hằng số, không phụ thuộc vào độ dài của ngăn xếp.

Bảng 1.1. Các thao tác cơ bản trên dữ liệu ngăn xếp

Số thứ tự	Thao tác	Hàm/Lệnh
1	Tạo một ngăn xếp rỗng.	`S = Stack()`
2	Đưa phần tử x vào đỉnh ngăn xếp S.	`push(S,x)`
3	Lấy ra một phần tử từ đỉnh của ngăn xếp S và trả về phần tử này.	`pop(S)`
4	Kiểm tra ngăn xếp rỗng. Trả về True nếu S rỗng, ngược lại trả về False.	`isEmptyStack(S)`
5	Trả về phần tử tại vị trí đỉnh của ngăn xếp S, S không thay đổi.	`top(S)`

Ví dụ 1. Để tạo một ngăn xếp rỗng có thể thực hiện lệnh sau:

```
S = Stack()
```

Ví dụ 2. Nếu muốn đưa lần lượt các số 2, 1, 5 vào ngăn xếp, cần thực hiện các lệnh sau:

```
push(S,2); push(S,1); push(S,5)
```

Ngăn xếp (stack) thuộc kiểu dữ liệu tuyến tính có các hàm cơ bản: hàm push() để đưa dữ liệu vào và hàm pop() để lấy dữ liệu ra ở cùng một đầu là đỉnh của ngăn xếp. Ngăn xếp hoạt động theo cơ chế "vào sau, ra trước" (LIFO). Một số hàm khác là isEmptyStack(), top().



1. Muốn lấy ra phần tử nằm ở đáy của ngăn xếp thì phải làm như thế nào?

2. Cho S là một ngăn xếp rỗng. Em hãy cho biết, khi thực hiện các lệnh sau thì S sẽ chứa những phần tử nào:

`push(S,1); push(S,5); pop(S); push(S,10).`

2. Mô hình dữ liệu hàng đợi

Hoạt động 2 Tìm hiểu mô hình dữ liệu hàng đợi

Đọc, trao đổi và thảo luận để hiểu về mô hình dữ liệu hàng đợi và cơ chế hoạt động "vào trước, ra trước" (FIFO – First In, First Out) của mô hình dữ liệu này.



Trong các ví dụ thực tế ở phần khởi động, ví dụ ở Hình 1.1b thuộc mô hình dữ liệu hàng đợi (queue). Có thể hiểu hàng đợi là đối tượng dữ liệu trong đó việc đưa dữ liệu vào tại một đầu và lấy dữ liệu ra ở đầu khác, theo cơ chế hoạt động FIFO.

Một ví dụ của mô hình hàng đợi là hàng đợi của máy in. Máy in trong văn phòng thường được sử dụng cho nhiều người, trong đó mỗi người có thể đưa ra các lệnh in bất cứ lúc nào. Máy in sẽ lưu trữ các nội dung in vào một hàng đợi, nội dung nào vào trước sẽ được in trước, nội dung nào vào sau được in sau.

Mô hình dữ liệu hàng đợi (queue) có thể được mô tả như sau:

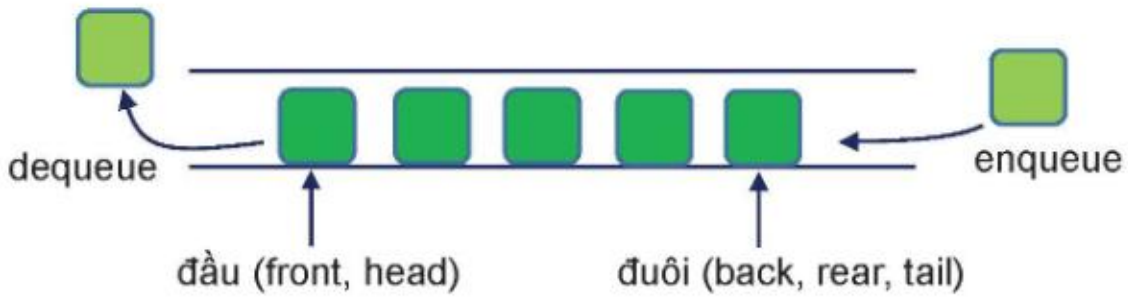
– Hàng đợi là một dãy tuyến tính các phần tử dữ liệu.

– Hàng đợi có các thao tác đưa phần tử vào ở một đầu và lấy phần tử ra tại một đầu khác của hàng đợi. Thao tác đưa dữ liệu vào là enqueue và lấy dữ liệu ra là dequeue.

– Quy ước đầu dùng để đưa dữ liệu vào là đuôi (back, rear, tail) của hàng đợi. Đầu ngược lại dùng để lấy dữ liệu ra là đầu (front, head) của hàng đợi.

– Mô hình dữ liệu hàng đợi hoạt động theo cơ chế FIFO.

Mô hình hàng đợi với cơ chế hoạt động FIFO có thể được biểu diễn như sau:



Hình 1.3. Mô hình hàng đợi

Sau đây là bảng các thao tác cơ bản trên dữ liệu hàng đợi. Có thể thiết lập các lệnh thực hiện những thao tác này đều có độ phức tạp thời gian `O(1)`, tức là hằng số, không phụ thuộc vào độ dài của hàng đợi.

Bảng 1.2. Các thao tác cơ bản làm việc trên dữ liệu hàng đợi

STT	Thao tác	Hàm/Lệnh
1	Tạo một hàng đợi rỗng.	`Q = Queue()`
2	Đưa phần tử x vào cuối của hàng đợi Q.	`enqueue(Q,x)`
3	Lấy ra một phần tử tại đầu của hàng đợi Q và trả về phần tử này.	`dequeue(Q)`
4	Kiểm tra hàng đợi rỗng. Hàm trả về True nếu Q rỗng, ngược lại trả về False.	`isEmptyQueue(Q)`
5	Trả về phần tử đầu của hàng đợi Q và Q không thay đổi.	`front(Q)`

Ví dụ 1. Muốn tạo một hàng đợi rỗng, cần thực hiện lệnh:

```
Q = Queue()
```

Ví dụ 2. Giả sử Q là hàng đợi đã có và chúng ta biết Q không rỗng. Muốn lấy ra phần tử ở đầu hàng đợi và chuyển xuống cuối Q, chúng ta thực hiện các lệnh sau:

```
x = dequeue(Q)
enqueue(Q,x)
```

Hàng đợi (queue) thuộc kiểu dữ liệu tuyến tính có các hàm cơ bản: hàm enqueue() để đưa dữ liệu vào ở đuôi hàng đợi và hàm dequeue() để lấy dữ liệu ra ở đầu hàng đợi. Hàng đợi hoạt động theo cơ chế "vào trước, ra trước" (FIFO). Một số hàm khác là isEmptyQueue(), front().



1. Hãy chỉ ra những điểm giống và khác nhau giữa ngăn xếp và hàng đợi.
2. Sau khi thực hiện các lệnh sau, hỏi trong hàng đợi Q có những giá trị nào?

```
Q = Queue()
enqueue(Q,2); enqueue(Q,10); dequeue(Q); enqueue(Q,1); dequeue(Q).
```



LUYỆN TẬP

1. Cho trước một dãy số, nếu đưa các số này lần lượt từ trái qua phải vào một ngăn xếp, sau đó lại lấy các số này ra từ ngăn xếp và xếp theo thứ tự lấy ra cũng từ trái qua phải, thì sẽ thu được dãy số mới như thế nào?
2. Giả sử cho một dãy các số, ví dụ 2, 5, 1, 0, 10, các số này lần lượt được kiểm tra, nếu là số chẵn sẽ được đưa vào hàng đợi Q, nếu là số lẻ thì đưa vào ngăn xếp S. Sau đó lần lượt lấy tất cả các số từ S và in ra màn hình. Hỏi các số được in ra màn hình lần lượt là các số nào?



VẬN DỤNG

1. Tìm thêm các ví dụ thực tế của ngăn xếp và hàng đợi, mô tả hoạt động của các ví dụ này.
2. Giả sử ngăn xếp S chứa các phần tử theo thứ tự từ đỉnh xuống đáy là 2, 1, 3. Được phép sử dụng một hàng đợi rỗng Q, em hãy sắp xếp các phần tử của ngăn xếp S theo thứ tự 3, 1, 2 (từ đỉnh xuống đáy).

BÀI 2 KIỂU DỮ LIỆU NGĂN XẾP

Sau bài học này em sẽ:

- Biểu diễn được ngăn xếp bằng mảng một chiều.
- Giải thích và viết được các chương trình con sử dụng các hàm cơ bản của kiểu dữ liệu ngăn xếp.

Theo em, những kiểu dữ liệu sau có thể được dùng để thiết lập dữ liệu ngăn xếp không? Tại sao?

a) Sử dụng kiểu mảng có chiều dài cố định N, với số tự nhiên N khá lớn.

b) Sử dụng kiểu dữ liệu danh sách liên kết (đã học ở chương trình Tin học 11 – Định hướng Khoa học máy tính).

c) Sử dụng kiểu dữ liệu list của Python.

1. Biểu diễn ngăn xếp bằng mảng một chiều

Hoạt động 1 Dùng kiểu dữ liệu mảng để biểu diễn ngăn xếp

Quan sát, trao đổi, thảo luận để tìm hiểu cách biểu diễn ngăn xếp bằng mảng một chiều. Trả lời các câu hỏi sau:

– Có thể biểu diễn ngăn xếp bằng mảng một chiều được không?

– Cần có các biến nào để thực hiện các phép toán cơ bản trên ngăn xếp?

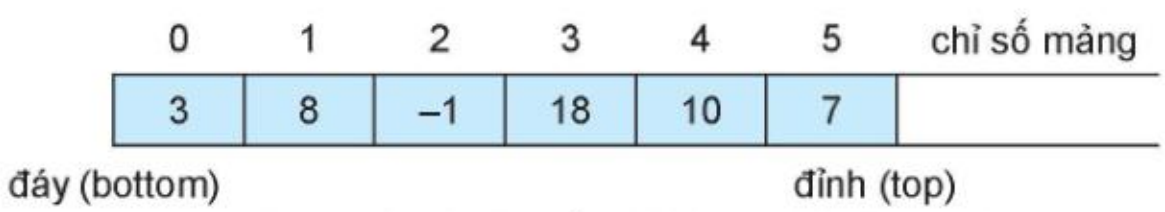
Chúng ta sẽ quan sát ngăn xếp được cài đặt bằng một danh sách (kiểu list của Python). Sau đây là các trường hợp của ngăn xếp.

a) Khởi tạo ngăn xếp S rỗng: Đặt S là danh sách rỗng. Hình 2.1a biểu diễn ngăn xếp rỗng.



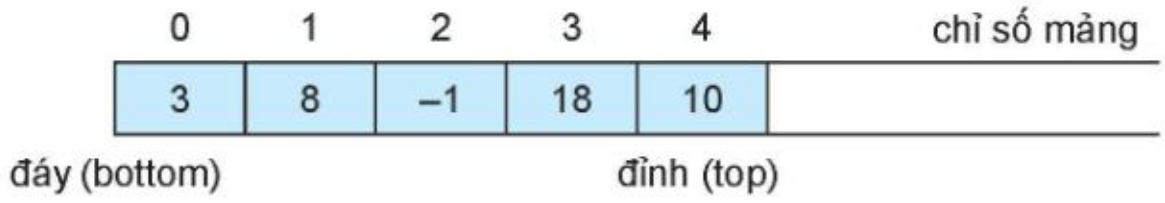
Hình 2.1a. Ngăn xếp rỗng

b) Phép toán `push(S,x)` dùng để thêm x vào đỉnh top của ngăn xếp S, nghĩa là thêm x vào cuối danh sách. Đáy (bottom) của ngăn xếp là phần tử đầu tiên và đỉnh (top) của ngăn xếp là phần tử cuối của danh sách. Hình 2.1b minh hoạ quá trình thêm các giá trị 3, 8, –1, 18, 10, 7 vào ngăn xếp:



Hình 2.1b. Ngăn xếp với phép toán thêm vào

c) Phép toán pop(S) dùng để lấy ra và trả về phần tử ở đỉnh (top) của ngăn xếp S, nghĩa là lấy ra phần tử cuối của danh sách. Hình 2.1c minh hoạ ngăn xếp sau khi lấy ra một phần tử.



Hình 2.1c. Ngăn xếp với phép toán lấy ra

Trong ví dụ trên, chúng ta thấy trong quá trình thêm vào và lấy ra các phần tử của ngăn xếp S thì đỉnh (top) luôn là phần tử cuối của danh sách.

Ngăn xếp được cài đặt bằng mảng một chiều (danh sách thuộc kiểu list của Python). Phép toán push(S, x) thêm x vào cuối mảng. Phép toán pop(S) lấy ra phần tử cuối của mảng. Đỉnh (top) là phần tử cuối của mảng.



1. Dãy các số 1, 2, 3, 4, 5, 6 lần lượt được đưa vào ngăn xếp S bằng lệnh push(). Người thực hiện làm như sau: Cứ thực hiện push(S,x) hai lần thì lại pop(S) một lần. Dãy số kết quả thu được bao gồm những số nào?
2. Giả sử chúng ta lần lượt thực hiện dãy các lệnh sau (ngăn xếp S ban đầu là rỗng).
   push(S,1); push(S,2); pop(S); push(S,3); pop(S); pop(S).
   Dãy các phần tử lần lượt được đưa ra khỏi ngăn xếp là các số nào?

2. Các phép toán của kiểu dữ liệu ngăn xếp

Hoạt động 2 **Tìm hiểu các hàm của kiểu dữ liệu ngăn xếp**

Đọc, trao đổi để biết các hàm cơ bản của ngăn xếp được cài đặt bằng danh sách (kiểu list của Python).



Sau đây là một số hàm cơ bản của ngăn xếp được cài đặt bằng danh sách (kiểu list của Python). Đỉnh (top) của ngăn xếp S luôn là phần tử cuối của danh sách S, nghĩa là biến `top = len(S) - 1`. Do đó không cần có biến top.

a) Hàm Stack() dùng để tạo ngăn xếp rỗng, hàm trả về danh sách rỗng:

```
def Stack():
    return []
```

Lệnh tạo ngăn xếp S rỗng (S là danh sách rỗng):

```
S = Stack()
```

b) Hàm push(S,x) dùng để thêm x vào đỉnh (top) của ngăn xếp, nghĩa là thêm x vào cuối danh sách S bằng hàm append():

```
def push(S,x):
    S.append(x)
```

Lệnh gọi hàm:

```
push(S, x)
```

c) Hàm `isEmptyStack(S)` trả về True nếu ngăn xếp S là rỗng; ngược lại hàm trả về False:

```
1  def isEmptyStack(S):
2      return len(S) == 0
```

d) Hàm `pop(S)` dùng để lấy ra phần tử tại đỉnh (top) của ngăn xếp S (phần tử cuối của danh sách S) và trả về phần tử này. Nếu ngăn xếp S rỗng thì hàm này báo lỗi ngoại lệ ValueError và dừng chương trình:

```
1  def pop(S):
2      if isEmptyStack(S):
3          raise ValueError("Ngăn xếp rỗng!")
4      else:
5          return S.pop()
```

Lệnh gọi hàm:

```
x = pop(S)
```

e) Hàm `top(S)` trả về phần tử tại đỉnh (top) của ngăn xếp S (phần tử cuối: `S[len(S)-1]` hoặc `S[-1]`) và phần tử này vẫn còn trong ngăn xếp S (ngăn xếp S không bị thay đổi). Nếu ngăn xếp S rỗng thì hàm này báo lỗi ngoại lệ ValueError và dừng chương trình:

```
1  def top(S):
2      if isEmptyStack(S):
3          raise ValueError("Ngăn xếp rỗng!")
4      else:
5          return S[len(S)-1]
```

Lệnh gọi hàm:

```
x = top(S)
```

Sau đây là một số ví dụ thiết lập và làm việc với ngăn xếp.

Ví dụ 1. Các lệnh sau thiết lập ngăn xếp rỗng và thêm 10 vào ngăn xếp:

```
S = Stack()
push(S,10)
```

Ví dụ 2. Giả sử cho trước ngăn xếp S, cần in tất cả các phần tử hiện có trong ngăn xếp S. Đoạn mã sau lấy ra và in các phần tử của ngăn xếp S cho đến ngăn xếp S là rỗng:

```
while not isEmptyStack(S):
    x = pop(S)
    print(x)
```

Ngăn xếp có thể được cài đặt bằng danh sách (kiểu list của Python). Các hàm cơ bản trên ngăn xếp S gồm `Stack()`, `isEmptyStack(S)`, `push(S,x)`, `pop(S)` và `top(S)`.

KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

1. Sửa lại hàm `pop(S)` và `top(S)` trong hoạt động trên như sau: Nếu ngăn xếp rỗng thì thông báo: "Ngăn xếp rỗng không thể thực hiện được lệnh này".
2. Vì sao các hàm cơ bản trên ngăn xếp S được cài đặt bằng danh sách (kiểu list của Python) không cần sử dụng biến top và biến bottom?

LUYỆN TẬP

1. Viết hàm `length(S)` trả về số phần tử của ngăn xếp S.
2. Giả sử dãy số ban đầu là 2, 7, 6, 1 và S là ngăn xếp rỗng. Chúng ta lần lượt thực hiện các thao tác `push(S,x)`, `pop(S)` với dãy số trên từ trái sang phải. Kết quả các số lần lượt được đưa ra khỏi ngăn xếp là 6, 7, 1, 2. Hãy viết các lệnh theo trình tự đã thực hiện.

VẬN DỤNG

1. Xâu kí tự được gọi là biểu thức nếu nó là rỗng hoặc chỉ chứa các kí tự "(" và ")". Ví dụ: "((()())())". Xâu biểu thức được gọi là đúng nếu vị trí các dấu ngoặc được sắp xếp hợp lí theo tự nhiên. Ví dụ các xâu sau là biểu thức đúng:

    ()

    (()())

    Ví dụ các xâu biểu thức sau là sai:

    (((())

    ))()()

    Có thể định nghĩa khái niệm biểu thức đúng bằng đệ quy như sau:

    – Xâu rỗng là đúng.

    – Nếu xâu A, B đúng thì xâu AB đúng.

    – Nếu xâu A là đúng thì xâu (A) đúng.

    Cho trước xâu biểu thức A, viết chương trình kiểm tra xem A có là biểu thức đúng hay không. Yêu cầu sử dụng kiểu dữ liệu ngăn xếp.

2. Ngăn xếp S được cài đặt bằng mảng T có N phần tử, phần tử đầu tiên có chỉ số 0. Hãy viết các hàm cơ bản trên ngăn xếp S.

    Lưu ý:

    – Biến topIdx cho biết đỉnh top của ngăn xếp.

    – Ngăn xếp là rỗng thì topIdx = -1. Khi topIdx = N-1 thì ngăn xếp bị tràn (overflow), không thể thêm phần tử mới vào ngăn xếp S.

    – Viết hàm `stackOverflow(S)` trả về True nếu ngăn xếp S bị tràn; ngược lại trả về False. Hàm `stackOverflow(S)` sẽ tạo ngoại lệ `ValueError()`. Sử dụng hàm `stackOverflow(S)` để kiểm tra ngăn xếp S chưa bị tràn trước khi gọi hàm `push(S,x)`.

BÀI 3 THỰC HÀNH KIỂU DỮ LIỆU NGĂN XẾP

Sau bài học này em sẽ:

- Biết cách sử dụng kiểu dữ liệu ngăn xếp để giải quyết các bài toán thực tế.

Trong bài trước, các em đã học cách thiết lập kiểu dữ liệu ngăn xếp. Kiểu dữ liệu ngăn xếp được sử dụng khá phổ biến trong các ứng dụng thực tế. Theo em, có thể sử dụng kiểu dữ liệu này để mô phỏng chức năng quay lại trang web đã duyệt trong các trình duyệt thông dụng như Google Chrome hay Bing được không?

Nhiệm vụ 1: Viết chương trình mô phỏng quá trình duyệt web

Hầu hết các trình duyệt web đều hỗ trợ chức năng quay lại trang web trước (backward). Để thực hiện chức năng này, các trình duyệt web lưu lại lịch sử các trang web đã duyệt trước đó.

Viết chương trình mô phỏng quá trình duyệt web của người dùng bằng cách sử dụng ngăn xếp. Chương trình cho phép người dùng nhấn phím số 1 để nhập vào địa chỉ trang web mới, nhấn phím số 2 để quay trở về trang web vừa duyệt trước đó, nhấn phím số 3 để kết thúc. Với mỗi lựa chọn, chương trình sẽ in ra thông báo về việc đi tới trang web tương ứng.

Hướng dẫn

Phân tích: Trong bài toán này, chúng ta cần sử dụng kiểu dữ liệu phù hợp để lưu trữ lịch sử các trang web đã duyệt. Mỗi lần người dùng duyệt web, cần lưu lại địa chỉ trang web. Khi người dùng chọn quay trở lại trang web trước (backward) thì cần truy xuất lại trang web ngay trước đó, nghĩa là trang web nào được lưu trữ sau cùng sẽ được truy xuất đầu tiên. Như vậy, dữ liệu ngăn xếp là kiểu dữ liệu phù hợp trong bài toán này.

web.py

```
1  from Stack import *
2  backward = Stack()
3  option = 0
4  while option != 3:
5      option = int(input("Hãy nhập vào lựa chọn của bạn:\n"))
6      if option ==1:
7          website = input("Hãy nhập vào địa chỉ website muốn đi đến:\n")
8          push(backward,website)
9          print("Đang đi đến trang web:"+ website)
10     if option ==2:
11         if isEmptyStack(backward):
12             print("Không tồn tại lịch sử duyệt web")
```

```
13          else:
14              website = pop(backward)
15              print("Đang đi đến trang web:"+ website)
```



Nhiệm vụ 2: Viết chương trình kiểm tra các dấu ngoặc trong biểu thức

Hầu hết công cụ hỗ trợ các ngôn ngữ lập trình bậc cao hiện nay đều có chức năng phát hiện và cảnh báo một số lỗi lập trình của người lập trình, ví dụ kiểm tra thứ tự xuất hiện các dấu đóng/mở ngoặc trong các biểu thức có hợp lệ hay không.

Em hãy viết chương trình cho phép người dùng nhập vào một biểu thức toán học và kiểm tra các dấu đóng mở ngoặc trong biểu thức có hợp lệ hay không. Biểu thức có thể chứa hai loại dấu đóng mở ngoặc là dấu "( )" và dấu "[ ]". Một biểu thức hợp lệ là biểu thức mà trong đó mỗi dấu mở ngoặc cần có các dấu đóng ngoặc tương ứng theo trình tự xuất hiện. Ví dụ biểu thức [(5 + 4)/(9 − 3)] được coi là hợp lệ; biểu thức [(5 + 4)/(9 − 3]) là không hợp lệ.

Hướng dẫn

Phân tích: Các dấu đóng mở ngoặc có thể được chia ra làm hai nhóm: nhóm các dấu ngoặc mở bao gồm "(", "[" và nhóm các dấu ngoặc đóng ")", "]". Một biểu thức là hợp lệ nếu số lượng các dấu ngoặc đóng, ngoặc mở phải bằng nhau, thêm vào đó, với mỗi dấu ngoặc đóng, dấu ngoặc mở ngay trước đó phải là dấu cùng loại. Ví dụ với dấu ")" thì dấu ngoặc mở ngay trước đó phải là dấu "(". Nếu dấu ngoặc mở ngay trước dấu ")" là "[" thì biểu thức là không hợp lệ.

Để giải bài toán này, chúng ta sử dụng ngăn xếp. Các bước thực hiện như sau:

– Khởi tạo một ngăn xếp để chứa các dấu ngoặc mở. Tiến hành duyệt qua từng kí tự trong biểu thức.

– Nếu kí tự là dấu ngoặc mở "(", "[", thì đẩy nó vào ngăn xếp.

– Nếu kí tự là dấu ngoặc đóng ")", "]", thì kiểm tra ngăn xếp. Nếu ngăn xếp rỗng hoặc phần tử tại đỉnh ngăn xếp không phải là dấu ngoặc mở tương ứng thì biểu thức không hợp lệ.

– Tiến hành duyệt cho đến hết biểu thức. Sau đó kiểm tra xem ngăn xếp có rỗng không. Nếu ngăn xếp là rỗng, biểu thức hợp lệ. Nếu ngăn xếp không rỗng, tức là còn dấu ngoặc mở chưa được đóng, biểu thức không hợp lệ.

Ví dụ: Xét biểu thức "([()])".

Duyệt kí tự "(" → đẩy vào ngăn xếp.

Duyệt kí tự "[" → đẩy vào ngăn xếp.

Duyệt kí tự "(" → đẩy vào ngăn xếp.

Duyệt kí tự ")" → lấy "(" khỏi ngăn xếp.

Duyệt kí tự "]" → lấy "[" khỏi ngăn xếp.

Duyệt kí tự ")" → lấy "(" khỏi ngăn xếp.

Hết biểu thức, kiểm tra ngăn xếp có rỗng hay không. Lúc này ngăn xếp rỗng nên biểu thức là hợp lệ.

kiemtrabieuthuc.py

```
1  from Stack import *
2  def kiemtrabt(bieuthuc):
```

```
    hople = True
    ngoacmo = Stack()
    for i in range(0, len(bieuthuc)):
        if bieuthuc[i] in {"(","["}:
            push(ngoacmo,bieuthuc[i])
        elif bieuthuc[i] in {")","]"}:
            if isEmptyStack(ngoacmo):
                hople = False
                break
            else:
                tmp = pop(ngoacmo)
                if (bieuthuc[i] == ")" and tmp!="(") or (bieuthuc[i] == "]"
and tmp!="["):
                hople = False
                break
    if not isEmptyStack(ngoacmo):
        hople = False
    return hople
bieuthuc = input("Hãy nhập vào một biểu thức:\n")
hople = kiemtrabt(bieuthuc)
if hople:
    print("Biểu thức hợp lệ")
else:
    print("Biểu thức không hợp lệ")
```

LUYỆN TẬP

1. Hãy sửa chương trình trong Nhiệm vụ 1 để thêm chức năng đi đến trang web kế tiếp (go forward). Sau khi người dùng chọn chức năng trở về trang web trước đó thì có thể sử dụng chức năng go forward để quay lại trang web vừa duyệt.
2. Sửa chương trình trong Nhiệm vụ 2 để in ra màn hình tổng số cặp đóng mở ngoặc của từng loại xuất hiện trong biểu thức.

VẬN DỤNG

1. Hãy viết chương trình mô phỏng quá trình xếp và lấy sách ra khỏi một ngăn tủ.
   Cho trước một số quyển sách, lần lượt xếp các quyển sách này vào ngăn tủ. Khi lấy ra, sách sẽ được lấy ra theo thứ tự quyển nào đưa vào sau sẽ được lấy ra trước. Để lấy được một quyển sách, chúng ta phải lấy các quyển sách ở phía ngoài ra trước. Ví dụ các quyển sách được xếp vào tủ theo thứ tự như sau: [English, Physic, Maths, Chemistry, History, Biology].
   Nếu muốn lấy quyển sách *Maths* ra khỏi ngăn sách thì chúng ta cần lấy các quyển *Biology*, *History*, *Chemistry* ra trước.
   Cho trước tệp chứa tên các quyển sách. Hãy tạo một ngăn xếp và đưa các quyển sách trong tệp vào ngăn xếp. Sau đó cho người dùng nhập vào tên quyển sách muốn lấy ra và in ra màn hình số quyển sách cần lấy ra trước khi lấy được quyển sách muốn lấy.
2. Cải tiến chương trình trong Nhiệm vụ 2 để có thể kiểm tra biểu thức có chứa ba loại dấu đóng mở ngoặc "( )", "[ ]", "{ }".

BÀI 4 KIỂU DỮ LIỆU HÀNG ĐỢI

Sau bài học này em sẽ:

- Biểu diễn được hàng đợi bằng mảng một chiều.
- Giải thích và viết được chương trình con sử dụng các hàm cơ bản của kiểu dữ liệu hàng đợi.

Từ các bài học trước, em đã biết viết chương trình đơn giản để sử dụng các hàm cơ bản của ngăn xếp được cài đặt bằng danh sách (kiểu list của Python). Em hãy trả lời các câu hỏi sau:

a) Có thể cài đặt hàng đợi bằng mảng một chiều tương tự như ngăn xếp được không?

b) Khi cài đặt hàng đợi bằng mảng một chiều, cần có thông tin nào để thực hiện phép toán thêm vào và lấy ra?

1. Biểu diễn hàng đợi bằng mảng một chiều

Hoạt động 1 **Dùng kiểu dữ liệu mảng để biểu diễn hàng đợi**

Quan sát, trao đổi, thảo luận để tìm hiểu cách biểu diễn hàng đợi bằng mảng một chiều. Em hãy trả lời các câu hỏi sau:

1. Có thể biểu diễn hàng đợi bằng mảng một chiều được không?
2. Cần có các biến nào để thực hiện các phép toán thêm vào và lấy ra?

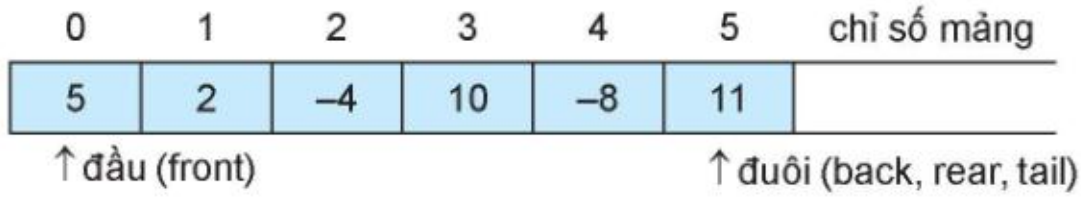
Chúng ta sẽ quan sát hàng đợi được cài đặt bằng một danh sách (kiểu list của Python). Sau đây là các trường hợp của hàng đợi:

a) Khởi tạo hàng đợi Q là rỗng; nghĩa là Q là danh sách rỗng. Hình 4.1a biểu diễn hàng đợi rỗng.



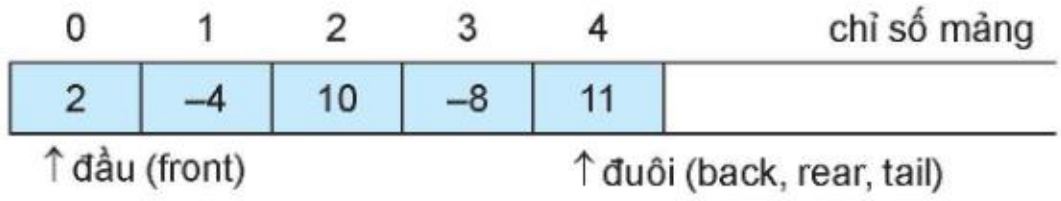
Hình 4.1a. Hàng đợi rỗng

b) Phép toán `enqueue(Q, x)` dùng để thêm x vào đuôi (back) của hàng đợi Q, nghĩa là thêm x vào cuối danh sách. Đầu (front) của hàng đợi là phần tử đầu tiên và đuôi (back, rear, tail) của hàng đợi là phần tử cuối của danh sách. Hình 4.1b cho thấy quá trình thêm các giá trị 5, 2, –4, 10, –8, 11 vào hàng đợi.



Hình 4.1b. Hàng đợi với phép toán thêm vào

c) Phép toán dequeue(Q) dùng để lấy ra và trả về phần tử ở đầu (front) của hàng đợi Q, nghĩa là lấy ra phần tử đầu tiên của danh sách. Hình 4.1c cho thấy hàng đợi sau khi lấy ra một phần tử.



Hình 4.1c. Hàng đợi với phép toán lấy ra

Qua ví dụ trên, chúng ta thấy trong quá trình thêm vào và lấy ra các phần tử của hàng đợi Q thì đầu (front) là phần tử đầu tiên và đuôi (rear) là phần tử cuối của danh sách.

Hàng đợi được cài đặt bằng mảng một chiều (danh sách thuộc kiểu list của Python). Phép toán enqueue(Q, x) thêm x vào cuối mảng. Phép toán dequeue(Q) lấy ra phần tử cuối của mảng. Đầu (front) là phần tử đầu tiên và đuôi (rear) là phần tử cuối của mảng.



1. Khi hàng đợi được cài đặt bằng danh sách (kiểu list của Python), em hãy cho biết cách tính số phần tử của hàng đợi này.
2. Ban đầu, hàng đợi là rỗng. Em hãy cho biết giá trị của phần tử ở đầu (front) và đuôi (rear) sau khi thực hiện tuần tự các phép toán `enqueue(Q,2)`; `enqueue(Q,10)`; `dequeue(Q)`; `enqueue(Q,6)`; `dequeue(Q)`; `enqueue(Q,9)`; `enqueue(Q,1)`.

2. Các phép toán của kiểu dữ liệu hàng đợi

Hoạt động 2 Tìm hiểu các hàm của kiểu dữ liệu hàng đợi

Đọc, trao đổi để biết các hàm cơ bản của hàng đợi được cài đặt bằng danh sách (kiểu list của Python).



Sau đây là các hàm cơ bản của hàng đợi được cài đặt bằng danh sách (kiểu list của Python). Đầu (front) của hàng đợi Q là phần tử đầu tiên của danh sách, nghĩa là biến front = 0. Đuôi (rear) của hàng đợi Q là phần tử cuối của danh sách, nghĩa là biến `rear = len(Q)-1`. Do đó, không cần các biến front và rear.

a) Hàm `Queue()` dùng để tạo hàng đợi rỗng, hàm trả về danh sách rỗng:

```
def Queue():
    return []
```

Lệnh tạo hàng đợi Q rỗng (Q là danh sách rỗng):

```
Q = Queue()
```

b) Hàm `enqueue(Q,x)` dùng để thêm x vào đuôi (rear) của hàng đợi Q, nghĩa là thêm x vào cuối danh sách Q bằng hàm `append()`:

```
def enqueue(Q,x):
    Q.append(x)
```

Lệnh gọi hàm:

```
enqueue(Q,x)
```

c) Hàm `isEmptyQueue(Q)` trả về True nếu hàng đợi Q là rỗng; ngược lại hàm trả về False.

```
def isEmptyQueue(Q):
    return len(Q) == 0
```

d) Hàm `dequeue(Q)` dùng để lấy ra phần tử tại đầu (front) của hàng đợi Q (phần tử đầu tiên của danh sách Q) và trả về phần tử này. Nếu hàng đợi Q rỗng thì hàm này báo lỗi ngoại lệ ValueError và dừng chương trình.

```
def dequeue(Q):
    if isEmptyQueue(Q):
        raise ValueError("Hàng đợi rỗng.")
    else:
        return Q.pop(0)
```

Lệnh gọi hàm:

```
x=dequeue(Q)
```

e) Hàm `front(Q)` trả về phần tử tại đầu (front) của hàng đợi Q (phần tử đầu tiên của danh sách) và phần tử này vẫn còn trong hàng đợi Q (hàng đợi Q không bị thay đổi). Nếu hàng đợi Q rỗng thì hàm này báo lỗi ngoại lệ ValueError và dừng chương trình.

```
def front(Q):
    if isEmptyQueue(Q):
        raise ValueError("Hàng đợi rỗng.")
    else:
        return Q[0]
```

Lệnh gọi hàm:

```
x=front(Q)
```

KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

Ví dụ 1. Các lệnh sau tạo hàng đợi rỗng và bổ sung 5 vào hàng đợi:

```
Q = Queue()
enqueue(Q,5)
```

Ví dụ 2. Cho trước dãy số A. Đoạn mã sau đây thêm các số lớn hơn hoặc bằng 0 vào hàng đợi Q, bắt đầu từ số đầu tiên. Kết thúc khi gặp số âm.

```
Q = Queue()
for x in A:
    if x >= 0:
        enqueue(Q,x)
    else:
        break
```

Hàng đợi có thể được cài đặt bằng một danh sách (kiểu list của Python). Các hàm cơ bản trên hàng đợi Q gồm `Queue()`, `isEmptyQueue(Q)`, `enqueue(Q,x)`, `dequeue(Q)` và `front(Q)`.



1. Khi hàng đợi Q được cài đặt bằng danh sách (kiểu list của Python), em hãy cho biết chỉ số của các phần tử tại đầu (front) và đuôi (rear). So sánh các chỉ số này với chỉ số của các phần tử tại đáy (bottom) và đỉnh (top) của ngăn xếp (cũng được cài đặt bằng danh sách).
2. Em hãy nêu sự giống nhau và khác nhau giữa các hàm của ngăn xếp và hàng đợi được cài đặt bằng danh sách (kiểu list của Python).



LUYỆN TẬP

1. Sửa lại hàm `dequeue(Q)` và `front(Q)` trong chương trình trên như sau: Nếu hàng đợi rỗng thì thông báo: "Hàng đợi rỗng không thể thực hiện được lệnh".
2. Viết hàm `length(Q)` trả về số phần tử của hàng đợi Q.



VẬN DỤNG

1. Hãy giải thích vì sao lệnh `dequeue(Q)` lại có độ phức tạp thời gian là O(n), với n là độ dài của hàng đợi hiện thời.
2. Cho trước mảng T gồm N phần tử T[0], T[1], …, T[N-1]. Hãy viết hàm thiết lập hàng đợi và các thao tác cơ bản với hàng đợi từ mảng T.

   Gợi ý:

   – Để thiết lập dữ liệu hàng đợi từ mảng T cho trước cần có thêm biến backIdx mô tả chỉ số của phần tử đuôi của hàng đợi. Ban đầu thiết lập `backIdx = -1` tương ứng với hàng đợi rỗng.

   – Cần viết thêm hàm `isFullQueue(Q)` kiểm tra xem hàng đợi đã đầy chưa. Hàm trả về True nếu hàng đợi Q đã đầy (`backIdx = N-1`), ngược lại trả về False.

BÀI 5 THỰC HÀNH KIỂU DỮ LIỆU NGĂN XẾP VÀ HÀNG ĐỢI

Sau bài học này em sẽ:

- Biết cách kết hợp các kiểu dữ liệu hàng đợi và ngăn xếp để biểu diễn các loại dữ liệu khác nhau.

Trong bài trước, chúng ta đã sử dụng kiểu dữ liệu hàng đợi và ngăn xếp. Trong nhiều trường hợp ứng dụng trong thực tế chúng ta phải kết hợp cả hai loại dữ liệu này. Em có thể nêu được một ví dụ cần sử dụng cả hai kiểu dữ liệu này không?

Nhiệm vụ: Viết chương trình mô phỏng bếp ăn tập thể

Nhà ăn tập thể của một doanh nghiệp, phục vụ cho người lao động xếp hàng vào chọn suất ăn. Nhà ăn này chỉ có đúng hai loại là cơm gà và cơm bò. Mỗi người khi vào phải xếp hàng và đăng kí món ăn (gà hoặc bò). Thông tin đăng kí suất ăn sẽ được lưu trong tệp input1.inp như hình bên. Trong đó, mỗi hàng tương ứng với lượt đăng kí của một người, số đầu tiên là số định danh (ID của người đăng kí), theo sau là loại suất ăn mà người đó chọn.

input1.inp

```
1 gà
2 bò
3 gà
4 gà
5 gà
6 bò
7 bò
8 gà
```

Căn cứ vào tệp đã đăng kí, người quản lí sẽ cho người lao động xếp thành hai hàng, một hàng gồm toàn bộ những người đã chọn cơm gà, hàng còn lại gồm những người chọn cơm bò.

Do nhà bếp không biết trước thông tin đăng kí của người lao động nên sẽ chuẩn bị trước các suất ăn một cách ngẫu nhiên. Các suất ăn được chuẩn bị sẵn và được đưa vào một ngăn xếp, tổng số lượng các suất ăn bằng với tổng số người lao động. Thông tin các suất ăn đã chuẩn bị sẵn được lưu trong tệp input2.inp có dạng như sau:

gà bò gà bò gà bò bò gà

Quy tắc nhà bếp hoạt động như sau: Mỗi lần người quản lí sẽ lấy một suất ăn trong ngăn xếp và đưa cho người đầu tiên trong hàng đợi tương ứng. Ví dụ, nếu suất ăn được lấy ra là cơm gà thì sẽ đưa cho người đầu tiên trong hàng chọn cơm gà.

Nếu số lượng các loại suất ăn không đúng với số lượng các suất ăn mà mọi người đăng kí, sẽ xảy ra hiện tượng một hàng đợi (ví dụ hàng đợi cơm gà) đã được phục vụ xong nhưng loại đó vẫn còn. Khi đó, người quản lí bắt buộc người đã đăng kí cơm bò phải chuyển sang cơm gà.

Hãy viết chương trình mô phỏng quá trình trên. Nhận đầu vào là hai tệp input1.inp thể hiện việc đăng kí suất ăn của mọi người và tệp input2.inp thể hiện các suất ăn đã được chuẩn bị trước và đưa vào ngăn xếp. Hãy cho biết có người nào buộc phải đổi suất ăn của mình hay không. Nếu có thì hãy in ra số ID của những người đó.

Hướng dẫn

Phân tích: Để xử lí bài toán này, việc đầu tiên là phải tiến hành đọc dữ liệu từ hai file input và đưa chúng vào những kiểu dữ liệu phù hợp. Với dữ liệu người lao động trong hàng đợi, yêu cầu người xếp hàng trước sẽ được phục vụ món ăn trước nên chúng ta sẽ sử dụng kiểu dữ liệu hàng đợi để mô phỏng hai hàng đợi của suất ăn gà và suất ăn bò. Với dữ liệu các suất ăn được đưa vào ngăn xếp, các suất ăn sẽ được lấy lần lượt từ trên xuống dưới nên chúng ta sử dụng dạng dữ liệu ngăn xếp để lưu trữ loại dữ liệu này.

Chúng ta sẽ duyệt lần lượt từng suất ăn trong ngăn xếp, nếu suất ăn là gà, chúng ta kiểm tra nếu vẫn còn người trong hàng đợi suất ăn gà, chúng ta phát suất ăn cho người đầu tiên trong hàng đó (dequeue) và tiếp tục. Nếu đã hết người trong hàng đợi suất ăn gà, chúng ta bắt buộc người đầu tiên trong hàng suất ăn bò đổi sang suất ăn gà, đưa ID của người này vào danh sách những người buộc phải đổi món ăn và tiếp tục. Quy trình tương tự được thực hiện nếu suất ăn lấy ra từ ngăn xếp là suất ăn bò. Chương trình thực hiện bài toán này có thể được thực hiện như sau:

bepan.py

```
from Queue import *
from Stack import *
def bepan(dangkiga, dangkibo, suatan):
    doimon = [] #list chứa ID những người phải đổi món
    while not isEmptyStack(suatan):
        tmp = pop(suatan)
        if tmp == "bò": # nếu suất ăn lấy ra là bò
            if not isEmptyQueue(dangkibo): #còn người đăng kí suất ăn bò
                dequeue(dangkibo)
            else:
                ID_doi = dequeue(dangkiga)
                doimon.append(ID_doi)
        elif tmp == "gà": # nếu suất ăn lấy ra là gà
            if not isEmptyQueue(dangkiga): #còn người đăng kí suất ăn gà
                dequeue(dangkiga)
            else:
                ID_doi = dequeue(dangkibo)
                doimon.append(ID_doi)
    return doimon
dangkiga = Queue()
dangkibo = Queue()
suatan = Stack()
file1= open("input1.inp",encoding="utf8")
for line in file1.readlines(): #đọc thông tin đăng kí và đưa vào 2 queue
    id,dangki = line.split()
    if dangki =="gà":
        enqueue(dangkiga,id)
    elif dangki == "bò":
        enqueue(dangkibo,id)
```

```
30 file1.close()
31 file2 = open("input2.inp", encoding ="utf8")
32 data = file2.read()
33 for item in data.split(' '): # đọc thông tin suất ăn
34     push(suatan,item)
35 file2.close()
36 doimon = bepan(dangkiga, dangkibo, suatan)
37 if len(doimon)==0:
38     print("Không có người nào phải đổi món ăn")
39 else:
40     print("Danh sách những người phải đổi món là:",doimon)
```



LUYỆN TẬP

1. Hãy chạy chương trình với nhiều trường hợp dữ liệu đầu vào khác nhau. Em có nhận xét gì về vị trí của những người phải đổi món ăn?

2. Hãy viết chương trình tính toán thời gian chờ đợi trung bình của mỗi người để nhận được suất ăn của mình, cho biết thời gian lấy một suất ăn ra khỏi ngăn xếp và đưa cho người lao động là 1 giây.



VẬN DỤNG

Bài toán nhà bếp được thay đổi như sau:

Yêu cầu người lao động xếp thành một hàng để nhận đồ ăn, trong đó những người đăng kí suất cơm gà và những người đăng kí suất cơm bò sẽ đứng lẫn với nhau. Các suất ăn vẫn được nhà bếp thực hiện và đưa vào một ngăn xếp để phục vụ người lao động.

Quy tắc chọn suất ăn như sau: Mỗi người đến lượt sẽ nhận suất ăn được đưa ra từ ngăn xếp, nếu suất ăn đó đúng với suất ăn đã đăng kí thì người này sẽ được nhận suất ăn và đưa ra khỏi hàng đợi. Ngược lại nếu suất ăn không đúng với đăng kí thì người này sẽ ra khỏi hàng đợi và xếp lại vào cuối hàng và tiếp tục đợi. Quá trình chọn suất ăn như trên sẽ dừng lại nếu tất cả số người xếp hàng đều nhận được suất ăn đúng của mình hoặc tất cả mọi người trong hàng đợi đều không thể nhận được suất ăn như đã đăng kí.

Em hãy viết chương trình đọc hai tệp dữ liệu chứa thông tin về các suất ăn của người lao động và tính số người không nhận được suất ăn của mình.

Dữ liệu đầu vào gồm hai tệp, tệp input1.inp chứa thông tin về các suất ăn của người lao động trong hàng đợi, tệp input2.inp chứa thông tin về các suất ăn mà nhà ăn đã chuẩn bị và đưa vào trong ngăn xếp. Ví dụ dữ liệu đầu vào và đầu ra như sau:

input1.inp
gà gà gà bò bò gà

input2.inp
gà bò bò bò gà gà

Dữ liệu đầu ra là một số nguyên cho trước chỉ số lượng người xếp hàng không thể chọn được suất ăn của mình. Trong ví dụ trên kết quả đưa ra là 2.

Chuyên đề
2

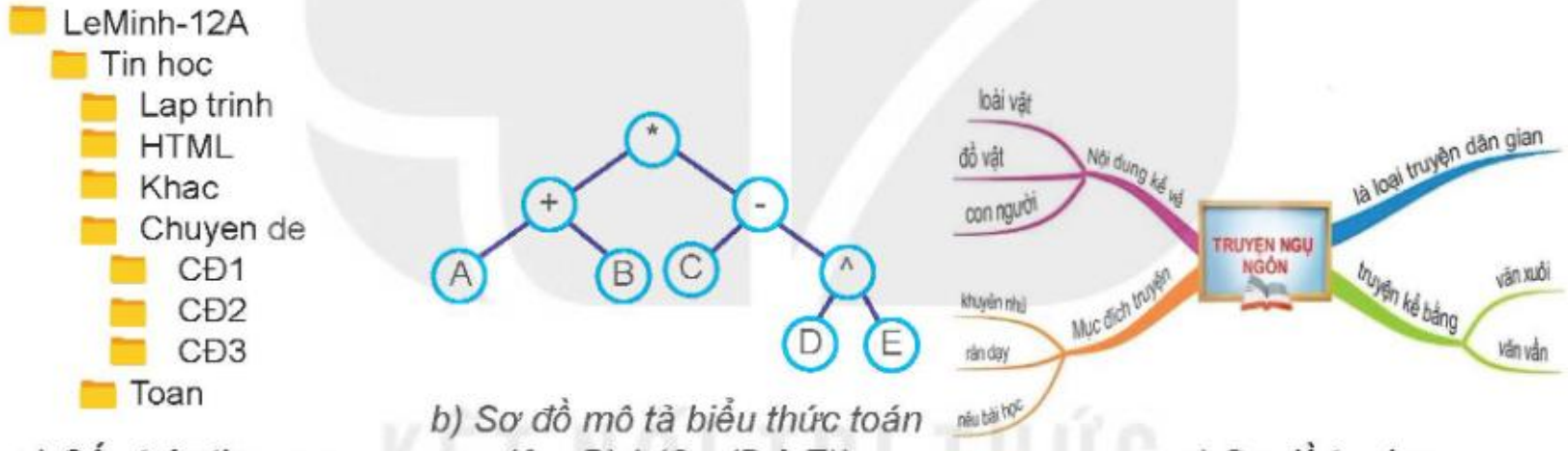
TÌM HIỂU CÂY TÌM KIẾM NHỊ PHÂN TRONG SẮP XẾP VÀ TÌM KIẾM

BÀI 6 CÂY NHỊ PHÂN

Sau bài học này em sẽ:

- Nêu được khái niệm cây, cây nhị phân.
- Biểu diễn được cây nhị phân bằng mảng một chiều.
- Trình bày và mô phỏng được các phép toán duyệt trước, duyệt giữa, duyệt sau cây nhị phân bằng biểu diễn trực quan.

1. Quan sát các sơ đồ biểu diễn thông tin trong Hình 6.1, em có nhận xét gì?
2. Các sơ đồ này có những điểm gì chung?



a) Cấu trúc thư mục

*b) Sơ đồ mô tả biểu thức toán (A + B) * (C - (D ^ E))*

c) Sơ đồ tư duy

Hình 6.1. Một số sơ đồ biểu diễn thông tin

1. Cấu trúc cây và cây nhị phân

Hoạt động 1 **Tìm hiểu cấu trúc cây và cây nhị phân**

Đọc, quan sát, thảo luận về khái niệm và cấu trúc cây. Với mỗi sơ đồ cây đã được mô tả trong hoạt động khởi động, hãy chỉ ra nút gốc, nút nhánh, nút lá và tính chiều cao của cây.

Các sơ đồ trong hoạt động khởi động đều có cấu trúc cây.

Cây (tree) bao gồm một tập hợp các nút (node) chứa thông tin, có kết nối với nhau, thường được gọi là quan hệ cha – con. Mỗi nút cha có thể kết nối với nhiều nút con. Mỗi nút chỉ có thể kết nối với một nút cha. Mỗi cây có một nút đóng vai trò gốc (root). Nút gốc không có nút cha (Hình 6.2).

Một số định nghĩa và khái niệm liên quan đến cấu trúc cây:

– Nút không có nút con được gọi là nút lá (leaf node).

– Nút có nút con được gọi là nút trong (inner node) hay nút nhánh (branch node).

– Mỗi nút cùng với các nút con bắt đầu từ nút đó tạo thành một cây con (sub tree).

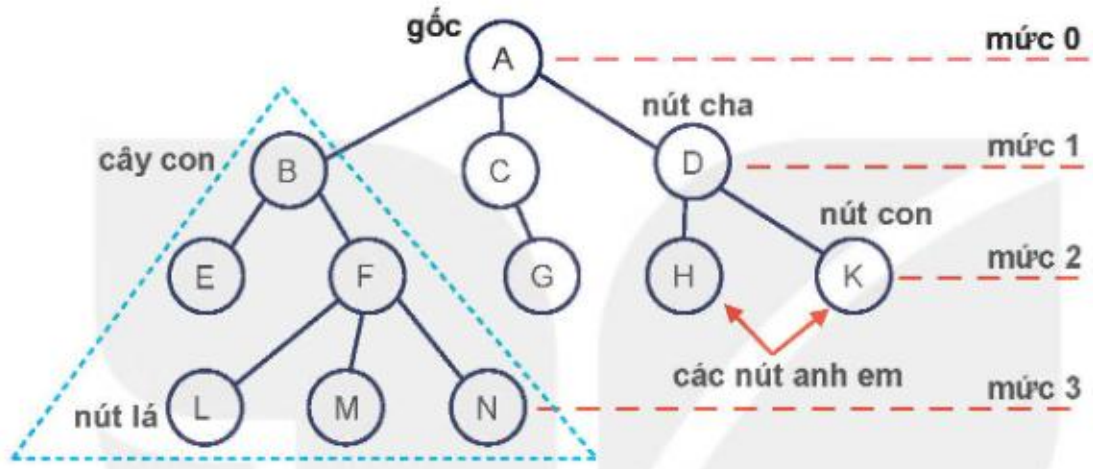
– Với mỗi nút của cây, số cạnh cần đi để về tới nút gốc được gọi là mức (level) của nút đó. Mức của nút gốc là 0. Mức của nút con bằng mức của nút cha cộng 1.

– Các nút có cùng nút cha được gọi là nút anh em (sibling node).

– Chiều cao (height) của cây được định nghĩa là độ dài đường đi đến nút lá sâu nhất, hay chính là mức cao nhất của các nút trên cây.

– Bậc (degree) của một nút là số các nút con của nó. Bậc của cây là bậc lớn nhất của các nút.

Các định nghĩa trên được thể hiện trong Hình 6.2.



Hình 6.2. Mô hình cây

Cây nhị phân (binary tree) là cây mà mọi nút có tối đa hai nút con là nút con trái và nút con phải.

- Cấu trúc cây bao gồm các nút có quan hệ cha – con. Cây có một nút gốc. Một nút có thể có nhiều nút con. Nút gốc không có nút cha, mỗi nút còn lại chỉ có một nút cha.
- Cấu trúc cây có nhiều ứng dụng trên thực tế và trong Khoa học máy tính.
- Cây nhị phân là cây mà mỗi nút có nhiều nhất hai nút con, được gọi là nút con trái và nút con phải.

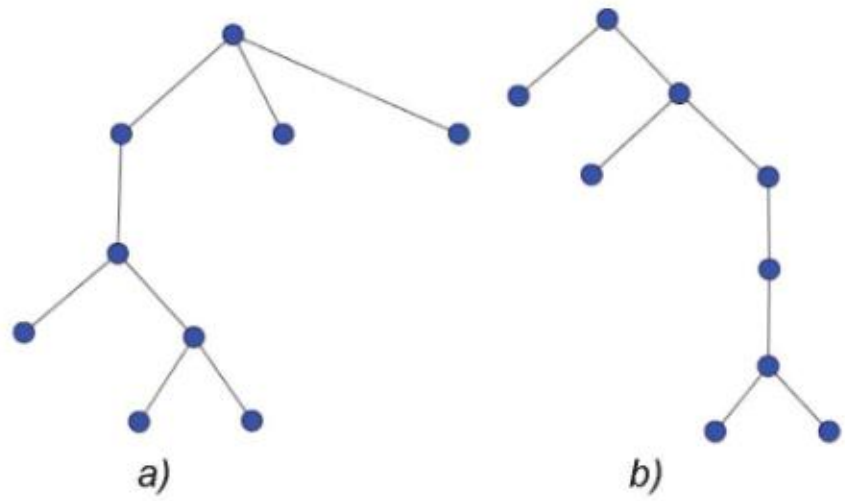
1. Tìm thêm các ví dụ cấu trúc cây.

2. Vẽ sơ đồ cây cho các biểu thức toán học sau:

a) (x + y) * (x – (y + z)/t).

b) x + (y + (z + t)/(u – v)).

3. Tính chiều cao của các cây trong Hình 6.3.



Hình 6.3. Một số sơ đồ cây

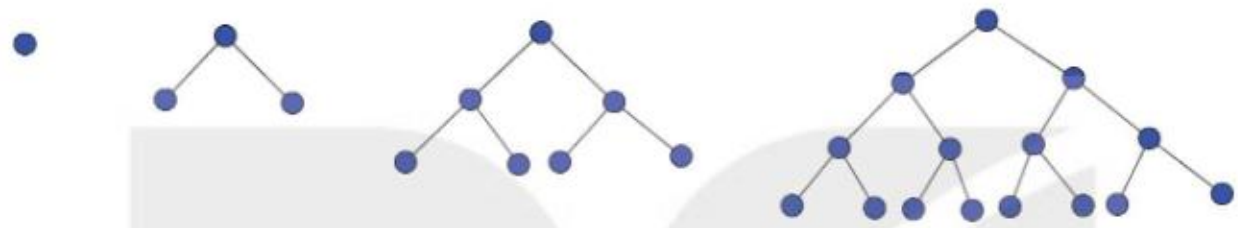
2. Biểu diễn cây nhị phân bằng mảng một chiều

Hoạt động 2 **Tìm hiểu cây nhị phân và cách biểu diễn cây nhị phân**

Đọc và thảo luận nhóm để tìm hiểu phân loại cây nhị phân và một số cách biểu diễn cây nhị phân bằng mảng một chiều hoặc bằng nút liên kết.

Chúng ta xét một số trường hợp đặc biệt của cây nhị phân.

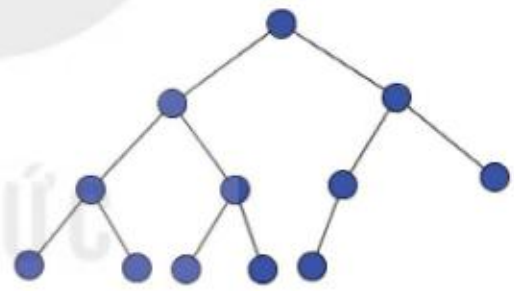
• Cây nhị phân được gọi là hoàn hảo (perfect) nếu mọi nút trong của cây đều có đủ hai nút con và tất cả các nút lá đều cùng mức. Nói cách khác, cây nhị phân là hoàn hảo nếu tất cả các mức của cây đều có đầy đủ tất cả các nút. Hình 6.4 mô tả các cây nhị phân hoàn hảo có chiều cao 0, 1, 2, 3 tương ứng.



Hình 6.4. Cây nhị phân hoàn hảo

Dễ thấy với cây nhị phân hoàn hảo, mỗi mức k sẽ có đủ 2^k nút. Do vậy nếu cây hoàn hảo có chiều cao h thì tổng số nút sẽ là $1 + 2 + 2^2 + ... + 2^h = 2^{h+1} - 1$.

• Cây nhị phân được gọi là hoàn chỉnh (complete) nếu tại mức i ($0 \le i < h$) có 2^i nút và tại mức h thì các nút liên tục tính từ trái qua phải, có thể khuyết một số nút bên phải, với h là chiều cao của cây.



Hình 6.5. Cây nhị phân hoàn chỉnh

Cây nhị phân hoàn hảo là trường hợp riêng của cây hoàn chỉnh, trong đó bậc cao nhất của cây có đủ 2^h nút. Với cây hoàn chỉnh, số lượng nút tại mức h sẽ có từ 1 đến 2^h nút.

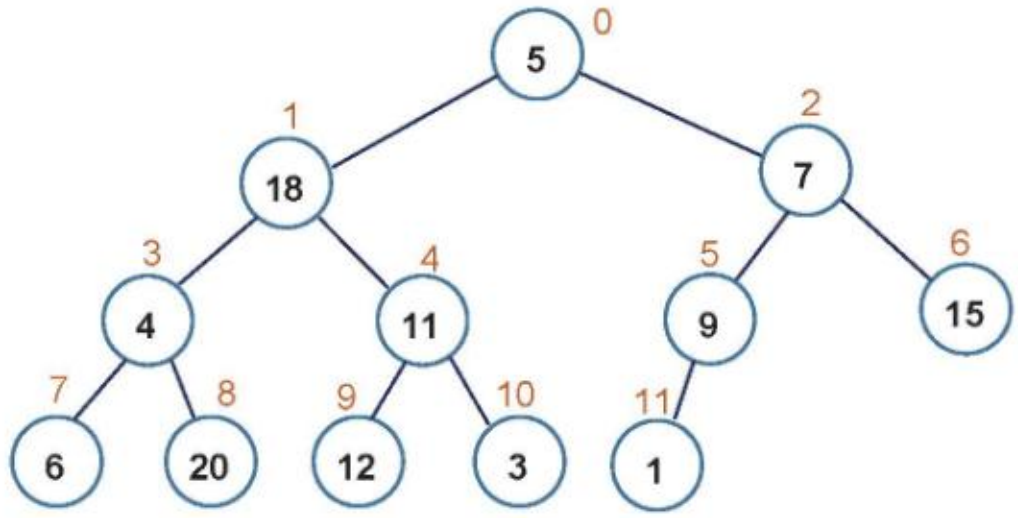
• Có nhiều cách biểu diễn cấu trúc của cây: cây có một nút gốc, mỗi nút có thể có nhiều nút con. Thông thường, cấu trúc của cây là cấu trúc nút liên kết. Đối với cây nhị phân, cấu trúc cây (tree) có thuộc tính root cho biết nút gốc của cây; cấu trúc nút (node) có thuộc tính left cho biết nút con trái và thuộc tính right cho biết nút con phải.

Riêng đối với cây nhị phân hoàn chỉnh, có thể biểu diễn thông tin của cây một cách đơn giản thông qua mảng một chiều. Ngược lại, mỗi mảng dữ liệu một chiều có thể là biểu diễn thông tin của cây nhị phân hoàn chỉnh.

Cây nhị phân hoàn chỉnh được đánh chỉ số như sau: bắt đầu từ 0 (nút gốc), sau đó đánh chỉ số lần lượt theo các nút ở từng mức, từ trái sang phải, cho đến nút cuối cùng của cây (Hình 6.6). Cụ thể như sau:

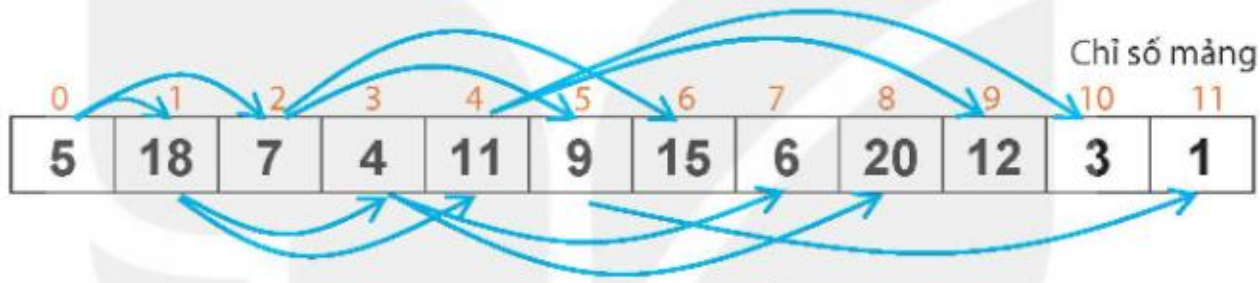
– Cây rỗng tương ứng với mảng rỗng.

– Nút gốc có chỉ số 0. Nếu nút có chỉ số k thì nút con trái có chỉ số 2k+1 và nút con phải có chỉ số 2k+2 và nút cha có chỉ số (k-1)/2.



Hình 6.6. Đánh chỉ số cây nhị phân hoàn chỉnh

Ngược lại, nếu cho trước mảng một chiều bất kì, khi đó có thể dễ dàng thiết lập cây nhị phân hoàn chỉnh tương ứng với mảng này. Nút gốc của cây sẽ tương ứng phần tử đầu tiên của mảng với chỉ số 0. Các phần tử tiếp theo tương ứng với chỉ số các nút của cây theo thứ tự từng mức, từ trái sang phải. Hình 6.7 mô tả quan hệ cha con của các phần tử của mảng nếu biểu diễn bằng cây nhị phân.



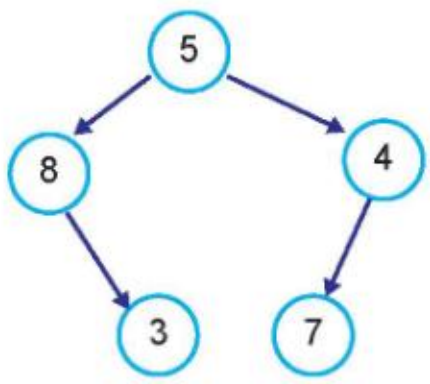
Hình 6.7. Quan hệ cha con giữa các phần tử của mảng

Cây nhị phân hoàn chỉnh có thể được biểu diễn bằng mảng một chiều có số phần tử bằng số nút của cây. Ngược lại, mảng một chiều có thể biểu diễn cây nhị phân hoàn chỉnh.

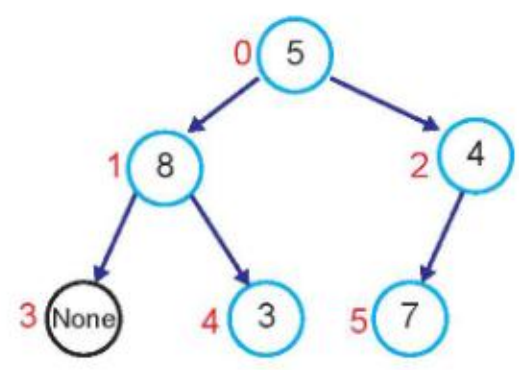
1. Cho mảng A = [2, 1, 8, 10, 0, 5, 9], biểu diễn cây nhị phân hoàn chỉnh. Hãy chỉ ra dãy các nút đi từ nút lá 9 về nút gốc 2.

2. Cho mảng A có 14 phần tử, biểu diễn cây nhị phân hoàn chỉnh. Tính chiều cao của cây nhị phân này.

Lưu ý: Cây nhị phân tổng quát cũng có thể được biểu diễn bằng mảng một chiều bằng cách bổ sung các nút rỗng có giá trị None để tạo thành cây hoàn chỉnh, sau đó biểu diễn mảng như đã nêu trên. Ví dụ sau minh hoạ cho ý tưởng này.



a) Cây nhị phân tổng quát



b) Cây nhị phân hoàn chỉnh biểu diễn cây nhị phân tổng quát

Hình 6.8. Biểu diễn cây nhị phân tổng quát bằng mảng

Cây nhị phân tổng quát đều có thể được biến đổi thành cây nhị phân hoàn chỉnh bằng cách bổ sung các nút giả (nút rỗng) None. Những cây nhị phân như vậy được gọi là cây nhị phân hoàn chỉnh đã biến đổi (có thể có các nút giả None) để phân biệt với cây nhị phân hoàn chỉnh (không có các nút giả None). Như vậy, cây nhị phân hoàn chỉnh có thể được biểu diễn bằng mảng một chiều, còn cây nhị phân tổng quát cũng có thể được biểu diễn bằng mảng một chiều sau khi bổ sung các nút giả None. Trong bài học này chỉ làm việc với cây nhị phân hoàn chỉnh, còn cây nhị phân hoàn chỉnh đã được biến đổi sẽ được đề cập trong các bài học sau.

3. Các thuật toán duyệt cây nhị phân

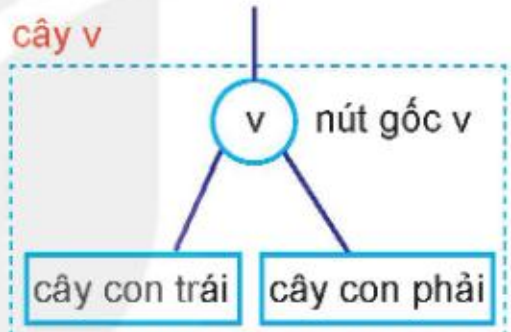
Hoạt động 3 **Tìm hiểu một số thuật toán duyệt cây nhị phân**

Trao đổi, thảo luận và thực hiện các thuật toán duyệt cây nhị phân. Bài toán đặt ra là cần duyệt tất cả các nút của cây nhị phân, mỗi nút duyệt một lần.

Trong phần này tất cả các cây nhị phân đều được hiểu là cây nhị phân hoàn chỉnh và được biểu diễn bằng mảng một chiều A cho trước.

a) Duyệt trước (preorder traversal)

Cây con có nút gốc v được gọi là "cây v" như minh hoạ ở Hình 6.9. Ý tưởng của phương pháp duyệt trước là bắt đầu từ nút gốc, sau đó duyệt cây con trái. Duyệt xong cây con trái thì chuyển sang duyệt cây con phải.



Hình 6.9. Cây v

Đoạn mã giả sau là thuật toán duyệt trước cây v. Lời gọi duyệt chính là `preorder(root)`.

```
preorder(cây v) # Duyệt trước (gốc-trái-phải) cây v
    Nếu cây v khác rỗng
        Duyệt nút v # gốc
        preorder(cây con trái của nút v) # trái
        preorder(cây con phải của nút v) # phải
```

Hàm cài đặt thuật toán duyệt trước trên Python có dạng `preorder(A,k)`, trong đó k là chỉ số nút bắt đầu duyệt, A là mảng biểu diễn cây nhị phân tương ứng.

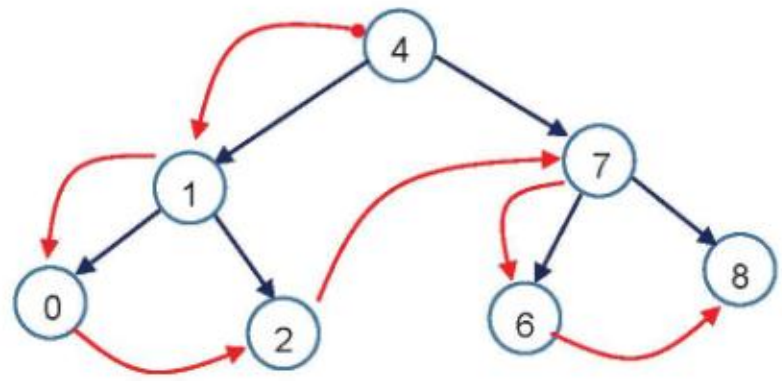
```
def preorder(A,k):
    if k < len(A) and A[k] != None:
        print(A[k], end = " ")
        preorder(A,left(k))
        preorder(A,right(k))
```

Chú ý các hàm `left()`, `right()` đã được xác định từ trước, ví dụ như sau:

```
def left(i):
    return 2*i + 1
def right(i):
    return 2*i+2
```

KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

Xét ví dụ cây nhị phân hoàn chỉnh có biểu diễn là mảng A = [4,1,7,0,2,6,8]. Với ví dụ này thì lệnh duyệt trước bắt đầu từ gốc sẽ duyệt các nút của cây lần lượt theo mũi tên trên Hình 6.10.



Hình 6.10. Thứ tự duyệt các nút theo thuật toán duyệt trước: 4, 1, 0, 2, 7, 6, 8.

b) Duyệt sau (postorder traversal)

Ý tưởng của phương pháp duyệt sau là duyệt toàn bộ cây con trái, sau đó duyệt cây con phải, cuối cùng duyệt nút gốc. Đoạn chương trình mã giả của thuật toán duyệt sau bắt đầu từ nút v như sau:

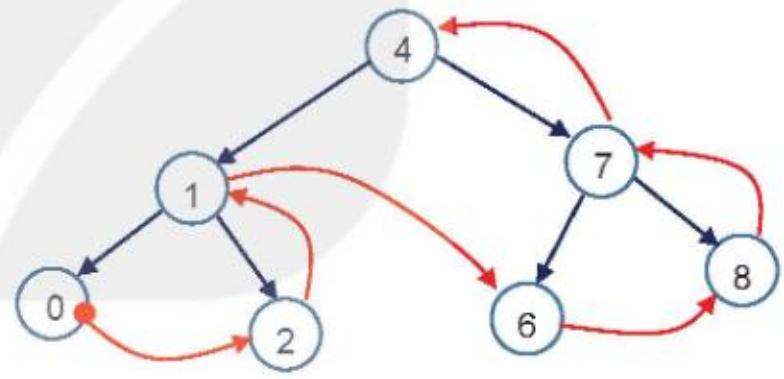
```
postorder(cây v) # Duyệt sau (trái-phải-gốc) cây v
    Nếu cây v khác rỗng
        postorder(cây con trái của nút v) # trái
        postorder(cây con phải của nút v) # phải
        Duyệt nút v # gốc
```

Lời gọi duyệt chính là `postorder(root)`.

Hàm cài đặt trên Python như sau:

```
def postorder(A,k):
    if k < len(A) and A[k]!= None:
        postorder(A,left(k))
        postorder(A,right(k))
        print(A[k], end = " ")
```

Với ví dụ trên thì thuật toán duyệt sau sẽ duyệt các nút của cây bắt đầu từ gốc lần lượt theo mũi tên trên Hình 6.11.



Hình 6.11. Thứ tự duyệt các nút theo thuật toán duyệt sau: 0, 2, 1, 6, 8, 7, 4.

c) Duyệt giữa (inorder traversal)

Ý tưởng của phương pháp duyệt giữa là duyệt cây con trái trước, sau đó duyệt nút gốc, cuối cùng duyệt cây con phải. Đoạn mã giả của thuật toán duyệt giữa bắt đầu từ nút v như sau:

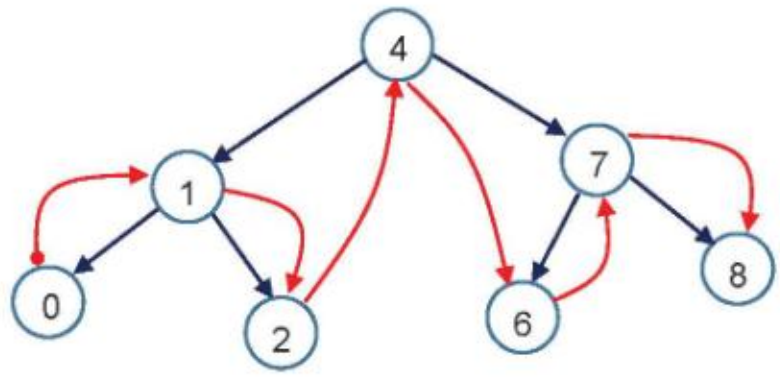
```
inorder(cây v) # Duyệt giữa (trái-gốc-phải) cây v
    Nếu cây v khác rỗng
        inorder(cây con trái của nút v) # trái
        Duyệt nút v # gốc
        inorder(cây con phải của nút v) # phải
```

Lời gọi duyệt chính là `inorder(root)`.

Hàm cài đặt trên Python.

```
def inorder(A,k):
    if k < len(A) and A[k] != None:
        inorder(A,left(k))
        print(A[k], end = " ")
        inorder(A,right(k))
```

Với ví dụ trên thì thuật toán duyệt giữa sẽ duyệt các nút của cây lần lượt theo mũi tên trên Hình 6.12.



Hình 6.12. Thứ tự duyệt các nút theo thuật toán duyệt giữa: 0, 1, 2, 4, 6, 7, 8.

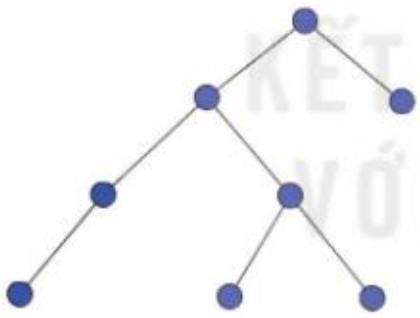
Các thuật toán duyệt cây nhị phân bao gồm duyệt trước, duyệt sau và duyệt giữa.

1. Cho mảng [A, B, C, D, E, F, G, H, I, J] biểu diễn một cây nhị phân. Em hãy cho biết thứ tự duyệt các nút của cây này theo phép duyệt trước (gốc-trái-phải).

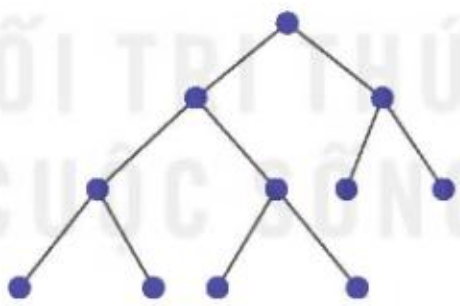
2. Với mảng dữ liệu ở Câu 1, thứ tự duyệt các phần tử sẽ như thế nào nếu thực hiện thuật toán duyệt sau?

LUYỆN TẬP

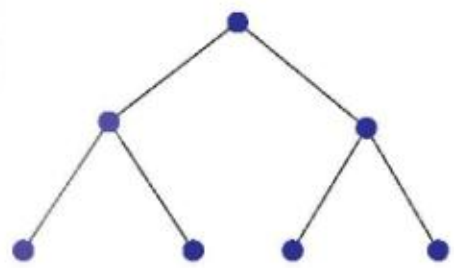
1. Cây nào sau đây là hoàn hảo? Cây nào là hoàn chỉnh? Cây nào không là hoàn hảo và hoàn chỉnh?



a)



b)



c)

2. Cây nhị phân gọi là đầy đủ nếu mỗi nút của nó hoặc là nút lá hoặc có đúng hai nút con. Khẳng định "Cây nhị phân đầy đủ sẽ luôn là hoàn chỉnh hoặc hoàn hảo" là đúng hay sai?

VẬN DỤNG

1. Cho mảng một chiều A biểu diễn cây nhị phân hoàn chỉnh T. Viết hàm `level(k)` trả về mức của nút tương ứng với phần tử A[k] của cây T.

2. Cho cây nhị phân T được biểu diễn bởi mảng một chiều A. Viết các hàm duyệt trước, duyệt giữa và duyệt sau trên cây T.

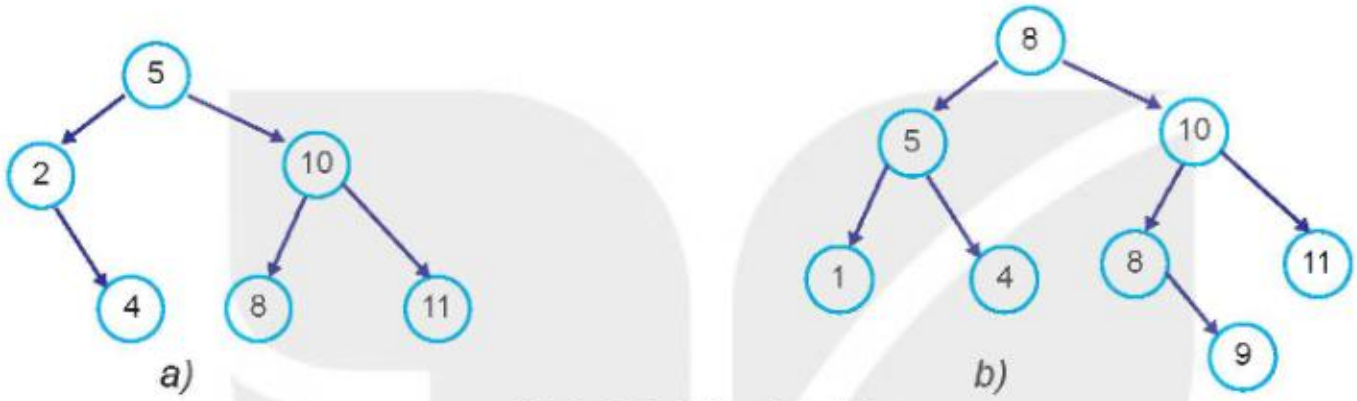
KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

BÀI 7 CÂY TÌM KIẾM NHỊ PHÂN

Sau bài học này em sẽ:

- Trình bày được khái niệm cây tìm kiếm nhị phân.
- Mô phỏng được thuật toán tạo cây tìm kiếm nhị phân từ một tập hợp các số cho trước.
- Biết và thực hiện được thuật toán tìm kiếm một giá trị của cây tìm kiếm nhị phân.

Quan sát các cây nhị phân sau, em có nhận xét gì về giá trị của các nút trên cây?



Hình 7.1. Cây nhị phân

Gợi ý: – Tại mỗi nút, so sánh dữ liệu của các nút của cây con trái và của cây con phải với nút này.

– Tại mỗi nút, so sánh dữ liệu của nút con trái và của nút con phải với nút này.

1. Cây tìm kiếm nhị phân

Cây tìm kiếm nhị phân (BST – Binary Search Tree) là một dạng đặc biệt của cây nhị phân thông thường, được tạo ra với mục đích hỗ trợ thuận tiện cho các bài toán tìm kiếm, chèn, xoá, sắp xếp.

Hoạt động 1 Tìm hiểu cấu trúc cây tìm kiếm nhị phân

Tìm hiểu và thảo luận về tổ chức dữ liệu của cây nhị phân và cây tìm kiếm nhị phân.

a) Mô hình dữ liệu cây nhị phân

Có thể tổ chức dữ liệu cây nhị phân theo hai cách là sử dụng mô hình nút liên kết hoặc mảng một chiều.

Mô hình nút liên kết của cây nhị phân sẽ bao gồm:

– Cấu trúc Node dùng để lưu thông tin của nút, gồm các thuộc tính key chứa dữ liệu của nút, thuộc tính left chứa liên kết nút con trái, thuộc tính right chứa liên kết nút con phải, thuộc tính p chứa liên kết nút cha.

– Cấu trúc Tree có gốc (root) của cây. Thuộc tính root chứa liên kết đến nút gốc của cây nhị phân.



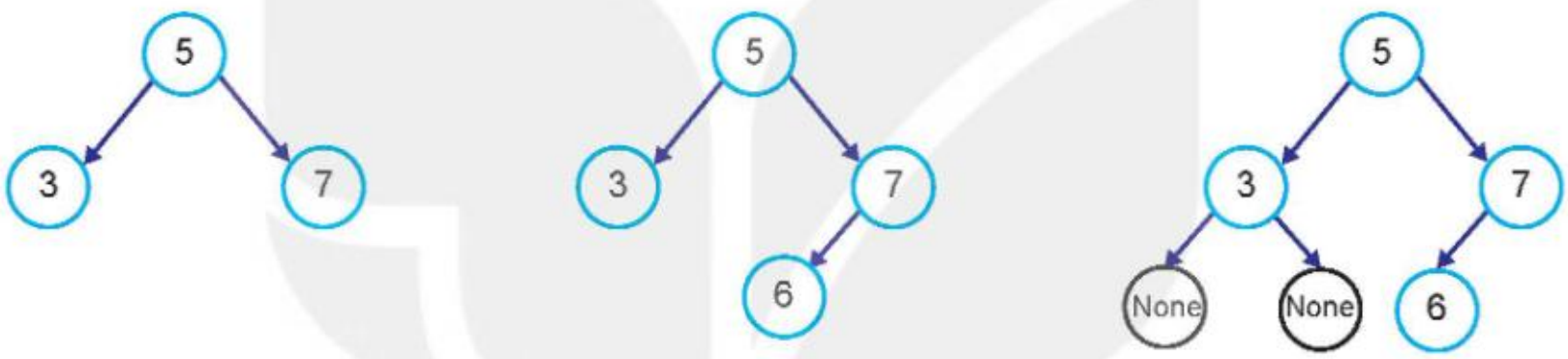
a) Cấu trúc *Node* b) Cấu trúc *Tree*

Hình 7.2. Mô hình nút liên kết cây nhị phân

Trong bài học trước, chúng ta đã biết cách biểu diễn cây nhị phân hoàn chỉnh bằng mảng một chiều và có thể biến đổi cây nhị phân tổng quát thành cây nhị phân hoàn chỉnh đã biến đổi bằng cách thêm các nút giả None. Cây nhị phân hoàn chỉnh đã biến đổi cũng có thể được cài đặt bằng mảng một chiều tương tự như cây nhị phân hoàn chỉnh. Theo cách biểu diễn này, mọi cây nhị phân T đều có thể biểu diễn bằng mảng, hay dãy các phần tử chính là các giá trị (khoá) của các nút của cây T. Nút gốc ứng phần tử T[0] của mảng. Cây rỗng tương ứng với mảng rỗng.

Cây nhị phân trong Hình 7.3b được cài đặt bằng mảng T = [5, 3, 7, 6].

Cây nhị phân tổng quát ở Hình 7.3c được thêm vào các nút giả None để trở thành cây nhị phân hoàn chỉnh và được cài đặt bằng mảng T = [5, 3, 7, None, None, 6].



a) Cây nhị phân hoàn chỉnh b) Cây nhị phân tổng quát c) Cây nhị phân hoàn chỉnh với các nút giả None

Hình 7.3. Biểu diễn cây nhị phân bằng mảng

Để thiết lập cây nhị phân rỗng, chúng ta sử dụng hàm sau:

```
1 def Tree():
2     return []
```

Các hàm `left(k)`, `right(k)` và `parent(k)` trả về chỉ số của nút con trái, nút con phải, nút cha của nút có chỉ số k.

```
1 def left(k):
2     return 2*k+1
3 def right(k):
4     return 2*k+2
5 def parent(k):
6     return (k-1)//2
```

b) Cây tìm kiếm nhị phân

Cây tìm kiếm nhị phân là cây nhị phân, có hai tính chất quan trọng:

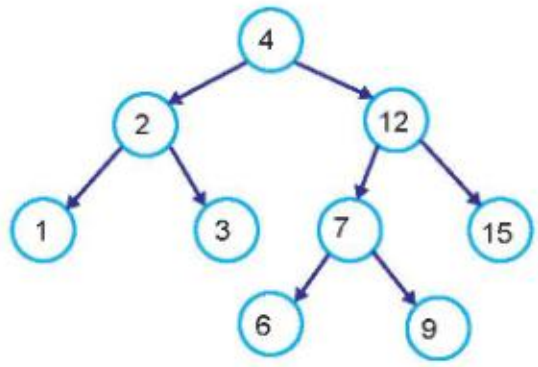
– Khoá của mỗi nút của cây lớn hơn khoá của tất cả các nút thuộc cây con trái và nhỏ hơn khoá của tất cả các nút thuộc cây con phải của nó.

– Hai nút khác nhau có khoá khác nhau.

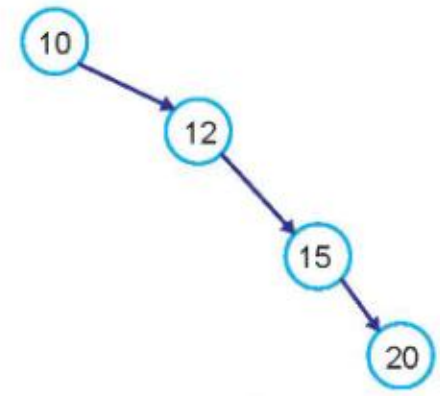
KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

Hình 7.4a là cây tìm kiếm nhị phân mà tại mọi nút thì chiều cao của cây con trái và của cây con phải lệch nhau nhiều nhất là 1. Cây này được gọi là cây cân bằng. Đây là trường hợp tốt nhất, tốn ít thời gian để tìm kiếm một khoá trên cây này, thuật toán tìm kiếm là tìm kiếm nhị phân.

Hình 7.4b là cây tìm kiếm nhị phân có chiều cao lớn nhất, mỗi nút chỉ có tối đa một nút con. Cây này được gọi là cây suy biến. Đây là trường hợp xấu nhất, tốn nhiều thời gian để tìm kiếm một khoá trên cây này, thuật toán tìm kiếm là tìm kiếm tuần tự.



a) Cây tìm kiếm nhị phân với các cây con có chiều cao gần bằng nhau



b) Cây tìm kiếm nhị phân suy biến có chiều cao lớn nhất

Hình 7.4. Cây tìm kiếm nhị phân

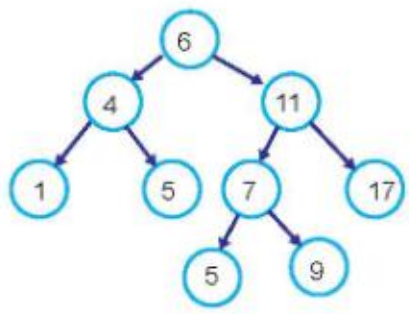
Khi cây tìm kiếm nhị phân được cài đặt bằng mảng T, tại mọi nút k, với mọi nút i thuộc cây con trái và với mọi nút j thuộc cây con phải, ta có bất đẳng thức:

```
T[i] < T[k] < T[j]
```

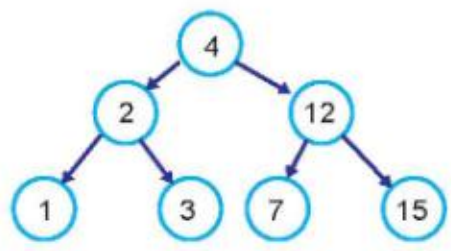
Lưu ý: Nếu cây nhị phân T là cây tìm kiếm nhị phân thì mọi cây con của T cũng là cây tìm kiếm nhị phân.

Cây nhị phân tổng quát có thể được cài đặt bằng cấu trúc nút liên kết hoặc bằng mảng một chiều. Cây tìm kiếm nhị phân là cây nhị phân mà tại mọi nút, khoá của nút này lớn hơn khoá của các nút con thuộc cây con trái và nhỏ hơn khoá của các nút con thuộc cây con phải. Khoá của các nút là duy nhất, nghĩa là hai nút khác nhau có khoá khác nhau.

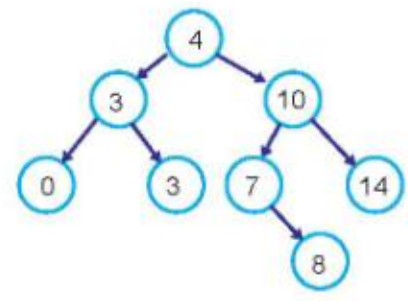
1. Trong Hình 7.5, em hãy cho biết cây nào là cây tìm kiếm nhị phân.



a)



b)



c)

Hình 7.5. Các cây nhị phân

2. Từ các khoá 1, 2, 3 có thể tạo ra được bao nhiêu cây tìm kiếm nhị phân? Hãy vẽ sơ đồ mô tả các cây này.

2. Chèn một khoá vào cây tìm kiếm nhị phân

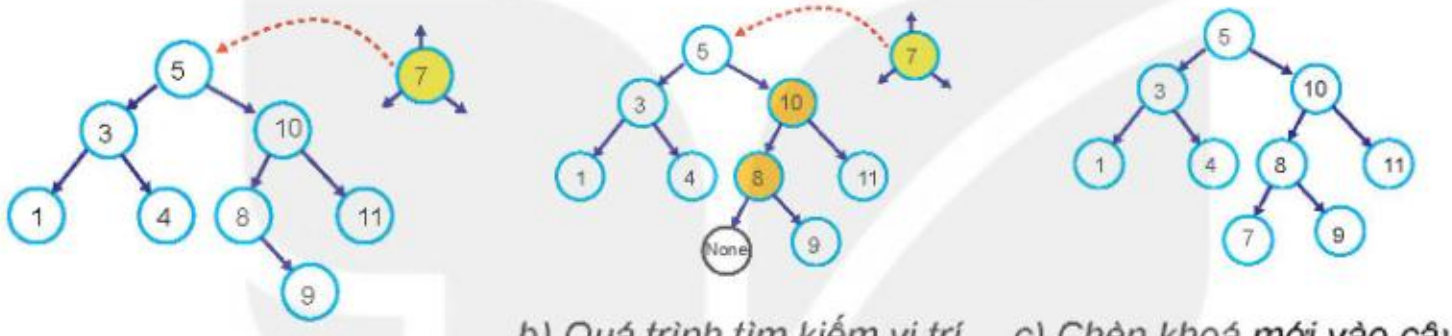
Hoạt động 2 **Thuật toán chèn khoá mới vào cây tìm kiếm nhị phân**

Bài toán: Cho cây tìm kiếm nhị phân T. Yêu cầu chèn khoá v vào cây T sao cho sau khi chèn khoá v thì cây T vẫn là cây tìm kiếm nhị phân.
Quan sát, thảo luận, tìm hiểu thuật toán tìm kiếm khoá 7 trên cây tìm kiếm nhị phân và cách chèn khoá 7 vào cây này.

Quá trình chèn khoá v = 7 vào cây tìm kiếm nhị phân T ở Hình 7.6a như sau:

Bước 1. Tìm vị trí cần chèn khoá v trên cây T (Hình 7.6b). Khoá v lớn hơn khoá 5, đi đến nút con phải. Khoá v nhỏ khoá 10, đi đến nút con trái. Khoá v nhỏ hơn khoá 8, đi đến nút con trái và gặp nút giả None.

Bước 2. Chèn khoá v vào cây T (Hình 7.6c). Trong trường hợp khoá v không có trong cây T thì chèn khoá v vào cây này bằng cách tạo nút thật mới tại nút giả None và gán khoá v cho nút mới này.



a) Cây tìm kiếm nhị phân

b) Quá trình tìm kiếm vị trí cần chèn

c) Chèn khoá mới vào cây tìm kiếm nhị phân

Hình 7.6. Chèn một khoá mới vào cây tìm kiếm nhị phân

Quá trình chèn một khoá v vào cây tìm kiếm nhị phân T gồm hai bước:
Bước 1. Tìm vị trí chính xác cần chèn. Nếu gặp khoá v thì dừng chương trình.
Bước 2. Thực hiện thao tác chèn.

Hàm `Tree_Insert(T, v)` dùng để chèn khoá v vào cây tìm kiếm nhị phân T được cài đặt bằng một danh sách (thuộc kiểu list của Python). Bước 1 thực hiện tìm vị trí cần chèn khoá v bắt đầu từ nút gốc T[0] cho đến khi gặp nút giả T[k] = None hoặc tìm thấy nút T[k] = v thì kết thúc. Bước 2 thực hiện chèn khoá v vào cây T tại nút k. Nếu k ≥ `len(T)` thì ta phải thêm các nút giả None từ chỉ số `len(T)` đến chỉ số k để cây T là cây nhị phân hoàn chỉnh. Số nút giả None phải thêm vào là k - `len(T)` + 1.

```
1  def Tree_Insert(T,v):
2      k = 0
3      while k < len(T) and T[k] != None:
4          if v < T[k]:
5              k = left(k)
6          elif v > T[k]:
7              k = right(k)
8          else:
9              return
```

Bước 1. Tìm vị trí cần chèn. Xuất phát từ gốc, đi theo các nút cho đến khi gặp nút rỗng. Nếu gặp nút có khoá v thì dừng chương trình.

```
10      if k >= len(T):
11          T.extend([None]*(k - len(T) + 1))
12      T[k] = v
```

← Bước 2. Thực hiện thao tác chèn, đặt v vào phần tử thứ k.

Đoạn chương trình sau thực hiện việc tạo cây tìm kiếm nhị phân từ một tập hợp các phần tử cho trước.

```
1  A = [7,1,9,0,5,10,2]
2  T = Tree()
3  for x in A:
4      Tree_Insert(T,x)
```



1. Cho trước dãy các số A = [10,1,2,11,8,15,20,9,0].

Hãy mô tả và vẽ sơ đồ cây nhị phân biểu diễn dãy số trên sau khi thực hiện thao tác chèn như đã mô tả trong hoạt động.

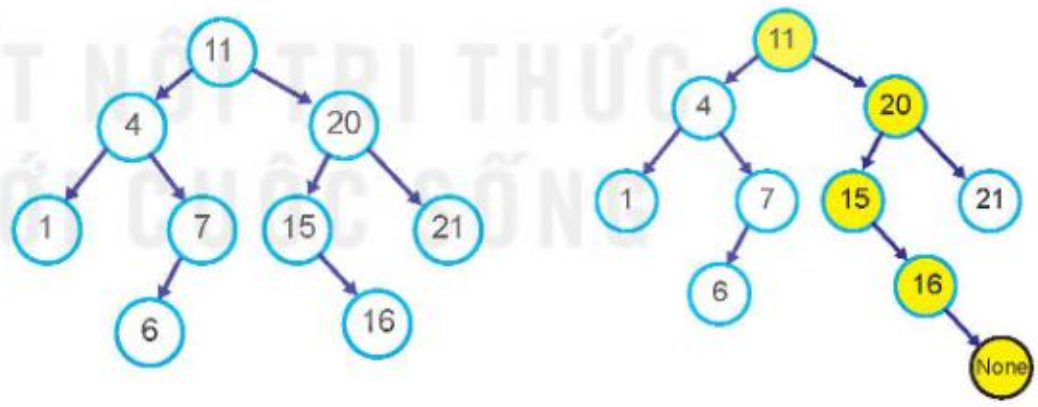
2. Với cây nhị phân đã có ở Câu 1, em hãy vẽ sơ đồ cây sau khi chèn khoá 14 và cho biết vị trí của khoá này ở trong cây.

3. Thuật toán tìm kiếm trên cây tìm kiếm nhị phân

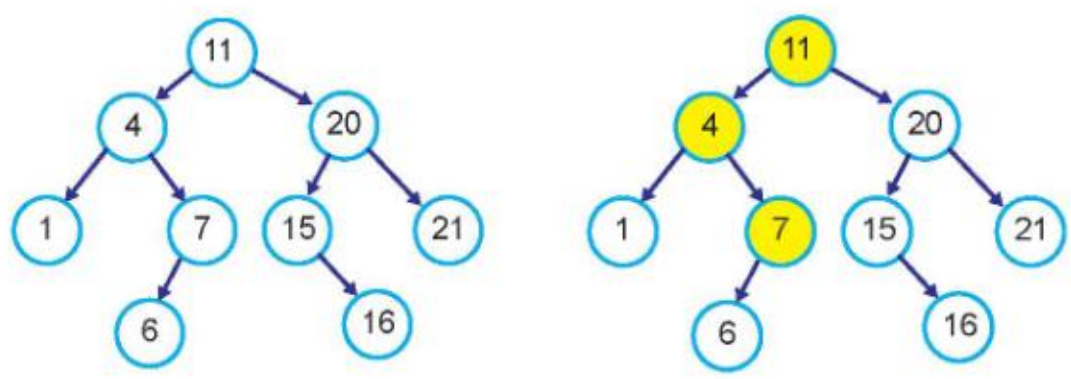
Hoạt động 3 **Tìm hiểu thuật toán tìm kiếm trên cây tìm kiếm nhị phân**

Quan sát quá trình tìm kiếm khoá trên cây tìm kiếm nhị phân thông qua các ví dụ cụ thể, thảo luận về thuật toán đã thực hiện.

a) Tìm kiếm khoá 18. Trình tự tìm kiếm: **11 20 15 16** None (không tìm thấy).



b) Tìm kiếm khoá 7. Trình tự tìm kiếm: **11 4** 7 (tìm thấy).



Hình 7.7. Tìm kiếm một khoá trên cây tìm kiếm nhị phân

Chúng ta sẽ thiết lập quá trình tìm kiếm một nút với khoá v được thực hiện bắt đầu từ nút có chỉ số k trên cây tìm kiếm nhị phân T. Nếu tìm thấy thì hàm trả về chỉ số của nút có giá trị v, ngược lại trả về –1.

– Bắt đầu tìm kiếm từ nút có chỉ số k.

– Việc tìm kiếm được thực hiện như sau:

+ Nếu k nằm ngoài khoảng chỉ số của T hoặc T[k] = None thì trả về –1.

+ Nếu T[k] = v thì dừng tìm kiếm và trả về k.

+ Nếu T[k] ≠ v thì sẽ tìm tiếp trong cây con trái nếu v < T[k], ngược lại tìm tiếp trong cây con phải nếu T[k] < v.

Từ suy luận trên, chúng ta thiết lập được hai cách tìm kiếm, một cách sử dụng kĩ thuật đệ quy và một cách không sử dụng kĩ thuật đệ quy. Cả hai cách này đều có độ phức tạp thời gian O(h) với h là chiều cao của cây tìm kiếm nhị phân.

Hàm tìm kiếm sử dụng đệ quy:

```
def search(T,k,v):
    if k >= len(T) or T[k] == None: # Nút T[k] là nút giả
        return -1
    else:
        if v == T[k]: # Tìm thấy khoá v
            return k  # Trả về chỉ số của nút
        elif v < T[k]:
            return search(T,left(k),v) # Tìm khoá v trên cây con trái
        else:
            return search(T,right(k),v) # Tìm khoá v trên cây con phải
```

Hàm tìm kiếm không sử dụng kĩ thuật đệ quy:

```
def search(T,k,v):     # Trong khi nút có chỉ số k thuộc cây T và
  # có khoá khác khoá v
  while k < len(T) and T[k] != None and T[k] != v:
    if v < T[k]:
      k = left(k)                    # k đến nút con trái
    else:
      k = right(k)                   # k đến nút con phải
  if k >= len(T) or T[k] == None:     # Không tìm thấy khoá v
    return -1
  else:                              # Tìm thấy khoá v
    return k                         # Trả về chỉ số của nút
```

Ví dụ. Đoạn mã sau đây tìm khoá v trên cây tìm kiếm nhị phân T và trả về chỉ số k >= 0 nếu tìm thấy; sau đó hiển thị kết quả tìm kiếm.

```
k = search(T,0,v) # Tìm khoá v trên cây tìm kiếm nhị phân T
if k >= 0:
    print("Tìm thấy nút có khoá", T[k])
else:
    print("Không tìm thấy khoá", v)
```

Thuật toán tìm kiếm khoá K trong cây tìm kiếm nhị phân có độ phức tạp thời gian O(h), với h là chiều cao của cây, hay tương đương trong trường hợp trung bình là O(logn) với n là số nút của cây.

1. Khi nào việc tìm kiếm trên cây tìm kiếm nhị phân là:

a) nhanh nhất? b) chậm nhất?

2. Cây tìm kiếm nhị phân T được thiết lập bằng cách chèn lần lượt các phần tử 3, 1, 6, 5, 0, 2, 4. Dùng sơ đồ mô tả các bước tìm kiếm giá trị khoá là:

a) 4. b) 10. c) 0.

LUYỆN TẬP

1. Thay đổi thứ tự chèn các phần tử vào cây nhị phân có tạo ra các cây tìm kiếm nhị phân khác nhau hay không? Cho ví dụ minh hoạ.

2. Nếu dãy số được đưa vào cây tìm kiếm nhị phân là tăng dần (hoặc giảm dần) thì cây tìm kiếm nhị phân tương ứng có dạng như thế nào?

VẬN DỤNG

1. Dữ liệu đầu vào là danh sách học sinh trong lớp và điểm trung bình các môn. Danh sách được cho trong tệp văn bản có dạng như bảng bên.

Viết chương trình đọc tệp dữ liệu đầu vào trên và liên tục thực hiện các thao tác sau:

Data.inp
Nguyễn Văn Năm 9.3
Bùi Văn Hai 9.0
Trần Quang Bảy 8.8

a) Nhập thêm vào danh sách học sinh và điểm trung bình.

b) Tìm kiếm với yêu cầu nhập họ tên học sinh và đưa ra kết quả họ tên học sinh, điểm trung bình hoặc thông báo "không tìm thấy".

Chương trình kết thúc khi nhập vào một xâu rỗng. Yêu cầu giải bài này bằng cây tìm kiếm nhị phân.

2. Viết hàm chèn khoá v vào cây tìm kiếm nhị phân T sử dụng kĩ thuật đệ quy.

3. Cho trước dãy A bao gồm các số nguyên và các giá trị None. Viết chương trình kiểm tra xem A có phải là biểu diễn của một cây nhị phân hoàn chỉnh đã biến đổi hay không?

Ví dụ:

Dãy [10, 7, 0, 5, None, 3] là biểu diễn của cây nhị phân hoàn chỉnh đã biến đổi.

Dãy [1, 6, None, 2, 3, None, 4] không là biểu diễn của cây nhị phân tổng quát nào.

4. Cho trước dãy A bao gồm các số nguyên và các giá trị None. Viết chương trình kiểm tra xem A có phải là biểu diễn của một cây tìm kiếm nhị phân hay không.

Ví dụ:

Dãy [5, 3, 6, None, 4, None, 10] là biểu diễn của cây tìm kiếm nhị phân.

Dãy [2, 1, 5, None, 3, 4, 10] không là biểu diễn của cây tìm kiếm nhị phân (mặc dù dãy này là biểu diễn của cây nhị phân hoàn chỉnh đã biến đổi).

BÀI 8 THỰC HÀNH CÂY TÌM KIẾM NHỊ PHÂN

Sau bài học này em sẽ:

- Thực hành thiết lập cây tìm kiếm nhị phân và sử dụng để tra cứu tìm kiếm dữ liệu trong các trường hợp cụ thể.

Trong Bài 7, cây tìm tìm kiếm nhị phân được cài đặt bằng mảng một chiều và mỗi nút của cây có khoá là một thuộc tính. Trong thực tế, một đối tượng có thể có nhiều thuộc tính. Ví dụ, với bài toán quản lí các món trong thực đơn, mỗi món có hai thuộc tính là tên và giá tiền. Trong trường hợp này, cây tìm kiếm nhị phân biểu diễn danh sách các món được cài đặt bằng mảng như thế nào và làm thế nào để mỗi nút của cây chứa hai thuộc tính là tên và giá tiền?

Nhiệm vụ: Viết chương trình quản lí thực đơn

Em có nhiệm vụ quản lí thực đơn các món ăn hoặc uống (gọi chung là món) của một nhà hàng. Mỗi món đều có tên (không trùng nhau) và giá tiền. Dữ liệu được nhập từ tệp văn bản menu.inp, mỗi dòng ứng với một món, có tên và giá tiền cách nhau bởi dấu phẩy.

menu.inp

```
Cơm suất cá thu sốt, 50000
Cơm suất cá trắm, 40000
Cơm sườn cốt lết, 85000
Phở xào bò, 35000
Phở tái chín, 40000
Bún chả, 60000
Cà phê đen, 30000
Cà phê nâu, 35000
Nước ngọt, 15000
```

Em hãy viết chương trình nhập thực đơn từ tệp menu.inp và lưu trữ vào cây tìm kiếm nhị phân được cài đặt bằng mảng, sau đó cho phép người dùng:

a) Tra cứu giá theo tên món, ví dụ khi nhập *Bún chả* thì chương trình thông báo giá 60 000.

b) Bổ sung thêm món hoặc cập nhật giá tiền. Ví dụ nhập *Cà phê đen, 35 000* thì chương trình hiểu là cập nhật lại giá *Cà phê đen* thành *35 000*, nếu nhập *Nước chanh, 20 000* thì chương trình sẽ bổ sung thêm món *Nước chanh*.

Hướng dẫn

Chương trình gồm hai bước: (1) cài đặt cây tìm kiếm nhị phân bằng mảng và (2) xây dựng chương trình đọc dữ liệu từ tệp, lưu trữ dữ liệu vào cây tìm kiếm nhị phân, cho phép người dùng tra cứu, bổ sung, cập nhật các món.

Bước 1. Cài đặt cây tìm kiếm nhị phân.

Phân tích: Mỗi món có hai thuộc tính là tên món và giá tiền. Mỗi nút của cây tìm kiếm nhị phân ứng với một món. Do đó, cấu trúc dữ liệu của nút, gồm có hai thuộc tính là

tên món và giá tiền, là mảng gồm hai phần tử [tên món, giá tiền]. Trong bài toán này, cây tìm kiếm nhị phân được cài đặt bằng một mảng, mỗi phần tử lại là một mảng gồm [tên món, giá tiền]. Ví dụ, nếu danh sách món trong tệp menu.inp gồm hai món *Bún chả, 60 000* và *Cà phê đen, 30 000* thì mảng biểu diễn cây tìm kiếm nhị phân là *[[Bún chả, 60 000], [Cà phê đen, 30 000]]*. Nếu cây tìm kiếm nhị phân được cài đặt bằng mảng T thì T[k] biểu diễn nút có chỉ số k, T[k][0] là tên món, T[k][1] là giá tiền. Hàm `Tree_Insert(T,name,price)` dùng để thêm món (name, price) vào cây tìm kiếm nhị phân T. Hàm `search(T,k,name)` dùng để tìm chỉ số của nút có tên là name, bắt đầu từ chỉ số k trong mảng T. Để thuận tiện cho việc bổ sung món hoặc cập nhật giá tiền, cần thêm hàm `Tree_Insert_Update(T,name,price)` gọi hàm `search` để kiểm tra xem món ăn đó đã tồn tại chưa, nếu đã tồn tại thì cập nhật giá, nếu chưa thì gọi hàm `Tree_Insert`.

Các hàm `Tree_Insert`, `search` và `Tree_Insert_Update` được viết trong tệp **BST_menu.py** như sau:

```
1  def Tree_Insert(T, name, price):
2      k = 0
3      while k< len(T) and T[k] != None:
4          if name < T[k][0]:
5              k = left(k)
6          elif name > T[k][0]:
7              k = right(k)
8          else:
9              return
10     if k >= len(T):
11         T.extend([None]*(k - len(T) + 1))
12     T[k] = [name, price] #Chèn thông tin món ăn dưới dạng một mảng vào phần tử k
13
14 def search(T, k, name):
15     if k > len(T) or T[k] == None:
16         return -1
17     else:
18         if name == T[k][0]:
19             return k
20         elif name < T[k][0]:
21             return search(T,left(k), name)
22         else:
23             return search(T,right(k), name)
24
25 def Tree_Insert_Update(T, name, price):
26     k = search(T, 0, name)
```

```
    if k == -1:
        Tree_Insert(T, name, price)
        return False #Thể hiện đã thêm món mới
    else:
        T[k][1] = price #Cập nhật giá món ăn
        return True #Thể hiện đã cập nhật
```

Bước 2. Xây dựng chương trình hoàn chỉnh.

Chương trình có bảng chọn ba chức năng tương ứng với thoát chương trình; tra cứu giá theo tên món; cập nhật giá tiền hoặc bổ sung món mới như sau:

```
0: Thoát chương trình
1: Tra cứu giá
2: Cập nhật hoặc thêm món
```

Để thực hiện tính năng này, em có thể dùng một vòng lặp trong đó mỗi vòng lặp thực hiện hỏi người dùng lựa chọn sau đó ứng với số được chọn, thực hiện các đoạn mã gọi các chức năng tương ứng. Mã nguồn chương trình như sau:

```
from BST_menu import *  #Khai báo thư viện cây tìm kiếm nhị phân

#Khởi tạo cây tìm kiếm và đọc dữ liệu từ tệp
T = Tree()
f = open("menu.inp", "r", encoding="utf8")
lines = f.readlines()
for line in lines:
    l = line.strip()
    l = l.split(",", maxsplit=2)
    Tree_Insert(T, l[0], l[1])
f.close()
#Hiển thị các chức năng chương trình và xử lí theo lựa chọn
options = ["Thoát chương trình", "Tra cứu giá", "Cập nhật hoặc thêm món"]
selected = -1
while selected != 0:
    for choice in range(len(options)):
        print(choice, "-", options[choice] )
    selected = int(input("Chọn chức năng: "))
    if selected <0 or selected>2:
        print("Chức năng không hợp lệ, hãy nhập số 0, 1 hoặc 2")
        continue
```

```
    #Kiểm tra lựa chọn và xử lí
    if selected == 1:
        item = input("Nhập tên món: ")
        k = search(T, 0, item)
        if k != -1:
                print("Giá ", item, ": ", T[k][1]) #T[k][1] là giá của
phần tử thứ k
        else:
            print("Món", item, "không có trong thực đơn")
    elif selected == 2:
        key = input("Nhập tên món:")
        price = int(input("Nhập giá tiền (số nguyên dương):"))
        if Tree_Insert_Update(T, key, price):
            print("Đã cập nhật giá", key)
        else:
            print("Đã thêm món", key, "vào thực đơn")
```



LUYỆN TẬP

1. Vẽ cây tìm kiếm nhị phân ứng với tệp menu.inp trong nhiệm vụ thực hành, lưu ý mỗi nút gồm hai thuộc tính *name* và *price*.

2. Mô tả quá trình tra cứu giá tiền món *Bún chả* thực hiện trên cây tìm kiếm nhị phân đã vẽ ở Luyện tập 1.



VẬN DỤNG

1. Sử dụng cây tìm kiếm nhị phân để viết ứng dụng quản lí tài khoản ngân hàng. Mỗi một tài khoản gồm mã tài khoản (duy nhất) và số dư tài khoản. Ứng dụng cho phép thêm tài khoản, sửa số dư tài khoản, tìm kiếm tài khoản theo mã tài khoản.

2. Sử dụng cây tìm kiếm nhị phân để viết chương trình quản lí danh sách học sinh của một trường trung học phổ thông. Mỗi một học sinh gồm các thông tin: mã học sinh (duy nhất), họ tên, ngày sinh. Chương trình cho phép thêm một học sinh vào danh sách với các trường thông tin kể trên, tìm kiếm học sinh theo mã và sửa đổi họ tên và ngày sinh ứng với một mã học sinh.

BÀI 9 CÁC THUẬT TOÁN DUYỆT TRÊN CÂY TÌM KIẾM NHỊ PHÂN

Sau bài học này em sẽ:

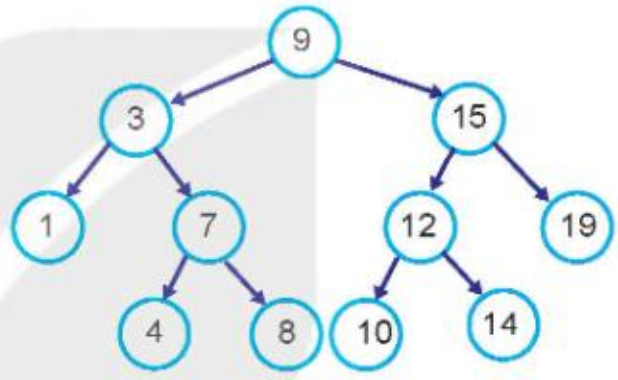
- Thực hiện được các thuật toán duyệt trên cây tìm kiếm nhị phân.
- Tìm kiếm và sắp xếp dãy số dựa trên cây tìm kiếm nhị phân.

Quan sát cây tìm kiếm nhị phân trong Hình 9.1, cùng trao đổi, thảo luận các câu hỏi sau:

a) Nếu thực hiện thuật toán duyệt giữa (trái – gốc – phải) thì nút đầu tiên được duyệt là nút nào?

b) Nút cuối cùng được duyệt là nút nào?

c) Thứ tự các nút được duyệt theo thuật toán duyệt giữa sẽ theo thứ tự nào? Em có nhận xét gì về kết quả đạt được? Giải thích vì sao.



Hình 9.1. Cây tìm kiếm nhị phân

1. Các thuật toán duyệt cây tìm kiếm nhị phân

Hoạt động 1 **Tìm hiểu các thuật toán duyệt cây tìm kiếm nhị phân**

Quan sát cách cài đặt các thuật toán duyệt trên cây tìm kiếm nhị phân và trao đổi về ý nghĩa và sự khác biệt khi thực hiện các thuật toán này so với các thuật toán duyệt cây nhị phân đã học trong Bài 6.

Từ Bài 6 chúng ta đã biết ba phương pháp duyệt cây nhị phân là duyệt trước, duyệt sau và duyệt giữa được cài đặt trên cây nhị phân hoàn chỉnh biểu diễn bằng mảng một chiều (kiểu dữ liệu list của Python). Để thực hiện các thuật toán này trên cây tìm kiếm nhị phân, cần nhớ lại cây tìm kiếm nhị phân muốn biểu diễn bằng mảng một chiều cần bổ sung thêm các nút giả None để trở thành cây nhị phân hoàn chỉnh trước khi có thể biểu diễn bằng mảng. Do vậy, nếu cây tìm kiếm nhị phân được biểu diễn bằng mảng T thì cần kiểm tra điều kiện `k < len(T)` và `T[k] ≠ None` để nút tại chỉ số k không là nút giả None.

Sau đây là các hàm duyệt cây tìm kiếm nhị phân. Chương trình in ra các khoá của các nút đã duyệt.

a) Thuật toán duyệt trước

Thuật toán duyệt trước bắt đầu từ nút k:

```
def preorder(T,k):
    if k < len(T) and T[k] != None:
        print(T[k], end = " ")
        preorder(T,left(k))
        preorder(T,right(k))
```

Lệnh duyệt trước toàn bộ cây tìm kiếm nhị phân T là:

```
preorder(T,0)
```

b) Thuật toán duyệt sau

Thuật toán duyệt sau bắt đầu từ nút k:

```
def postorder(T,k):
    if k < len(T) and T[k] != None:
        postorder(T,left(k))
        postorder(T,right(k))
        print(T[k], end = " ")
```

Lệnh duyệt sau toàn bộ cây tìm kiếm nhị phân T là:

```
postorder(T,0)
```

c) Thuật toán duyệt giữa

Thuật toán duyệt giữa bắt đầu từ nút k:

```
def inorder(T,k):
    if k < len(T) and T[k] != None:
        inorder(T,left(k))
        print(T[k], end = " ")
        inorder(T,right(k))
```

Lưu ý: Thuật toán duyệt này sẽ duyệt các nút lần lượt theo thứ tự tăng dần của khoá.

Lệnh duyệt giữa và in ra màn hình toàn bộ các khoá cây tìm kiếm nhị phân T theo thứ tự tăng dần là:

```
inorder(T,0)
```

KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

d) Thuật toán duyệt ngược

Thuật toán duyệt ngược bắt đầu từ nút k:

```
def reverseorder(T,k):
    if k < len(T) and T[k] != None:
        reverseorder(T,right(k))
        print(T[k], end = " ")
        reverseorder(T,left(k))
```

Lưu ý: Thuật toán duyệt này sẽ duyệt các nút lần lượt theo thứ tự giảm dần của khoá.

Lệnh duyệt ngược và in ra màn hình toàn bộ các khoá cây tìm kiếm nhị phân T theo thứ tự giảm dần là:

```
reverseorder(T,0)
```

Các thuật toán duyệt chính trên cây tìm kiếm nhị phân bao gồm duyệt trước, duyệt giữa, duyệt sau và duyệt ngược. Thuật toán duyệt giữa sẽ duyệt các nút của cây theo thứ tự tăng dần của khoá. Thuật toán duyệt ngược sẽ duyệt các nút của cây theo thứ tự giảm dần của khoá.

1. Cho trước dãy số A = [2,1,9,0,2,1,5]. Tạo cây tìm kiếm nhị phân T từ dãy A và thực hiện thuật toán duyệt giữa trên cây T. Em hãy cho biết kết quả duyệt là dãy các khoá có thứ tự như thế nào.

2. Với cây T như Câu 1, nếu thực hiện thuật toán duyệt ngược thì thứ tự các khoá thể hiện trên màn hình như thế nào?

2. Sắp xếp dãy số bằng cây tìm kiếm nhị phân

Hoạt động 2 Tìm hiểu thuật toán sắp xếp dãy số bằng cây tìm kiếm nhị phân

Trao đổi, thảo luận để giải bài toán sau:
Cho trước dãy số A. Thiết kế thuật toán sắp xếp lại dãy A theo thứ tự tăng dần hoặc giảm dần.

Bài toán sắp xếp các phần tử của một dãy số đã được trình bày trong các thuật toán sắp xếp, ví dụ:

– Các thuật toán sắp xếp đơn giản như sắp xếp chèn, sắp xếp chọn, sắp xếp nổi bọt.

– Sắp xếp trộn sử dụng kĩ thuật chia để trị.

Có một cách sắp xếp khác sử dụng cây tìm kiếm nhị phân. Cách này thực hiện dựa trên các thuật toán duyệt trên cây tìm kiếm nhị phân. Đoạn mã giả của thuật toán sắp xếp như sau:

```
1  BSTSort(A)
2  Thiết lập cây tìm kiếm nhị phân T từ dãy A.
3  Thực hiện thuật toán duyệt giữa (inorder) để in các nút theo thứ tự
   tăng dần, cập nhật các thay đổi vào dãy A.
```

Theo định nghĩa thì cây tìm kiếm nhị phân không cho phép khoá trùng nhau, do vậy nếu thực hiện theo đúng thuật toán đã mô tả trên thì sẽ chỉ sắp xếp được các phần tử khác nhau của dãy A. Muốn xử lí chính xác các khoá trùng nhau của cây tìm kiếm nhị phân chúng ta sẽ cần thêm công cụ để xử lí trường hợp khoá trùng nhau trên cây tìm kiếm nhị phân T. Có nhiều cách xử lí yêu cầu này, ví dụ:

Cách 1. Bổ sung biến để lưu thông tin về số lần lặp của các khoá trên cây T. Bên cạnh mảng T, cần có thêm mảng C với ý nghĩa như sau: C[k] = số lần lặp của khoá T[k]. Mảng C có độ dài bằng T và được cập nhật đồng thời với T.

Cách 2. Định nghĩa lại cây tìm kiếm nhị phân T cho phép khoá có giá trị trùng nhau. Theo cách này việc sắp xếp lại dãy dựa trên cây tìm kiếm nhị phân sẽ rất tự nhiên và dễ dàng.

Chúng ta sẽ thiết lập theo cách 2.

Với cây nhị phân được hiểu theo cách mới cho phép các khoá trùng nhau thì thuật toán chèn khoá hoàn toàn tương tự thuật toán đã trình bày trong Bài 7, chỉ có một điểm khác biệt duy nhất là chương trình sẽ không dừng lại khi gặp trường hợp trùng khoá. Hàm chèn khoá v vào cây tìm kiếm nhị phân T sẽ như sau:

```
1  def Tree_Insert(T,v):
2      k = 0
3      while k < len(T) and T[k] != None:
4          if v < T[k]:
5              k = left(k)
6          else:
7              k = right(k)
8      if k >= len(T):
9          T.extend([None]*(k - len(T) + 1))
10     T[k] = v
```

Chúng ta cần viết lại thuật toán duyệt giữa bằng hàm `new_inorder(T,k,A)`. Hàm sẽ thực hiện duyệt giữa trên cây tìm kiếm nhị phân T bắt đầu từ nút k, trong khi duyệt sẽ đưa các giá trị khoá của các nút được duyệt vào mảng A.

```
1  def new_inorder(T,k,A):
2      if k < len(T) and T[k] != None:
3          new_inorder(T,left(k),A)
4          A.append(T[k])
5          new_inorder(T,right(k),A)
```

Đoạn mã giả mô tả thuật toán sắp xếp dãy trên có thể được viết lại trên mô hình cây tìm kiếm nhị phân mới như sau. Chương trình sẽ sử dụng thư viện cây tìm kiếm nhị phân **BST.py**.

```
1 from BST import *
2 def BSTSort(A):
3     T = []
4     for x in A:
5         Tree_Insert(T,x)
6     A.clear()
7     new_inorder(T,0,A)
```

Có thể thiết lập thuật toán sắp xếp danh sách theo kĩ thuật sử dụng cây tìm kiếm nhị phân bằng cách duyệt giữa trên cây tìm kiếm nhị phân được tạo bởi danh sách.



1. Viết lại hàm `BSTSort(A)` thực hiện sắp xếp dãy số A theo thứ tự tăng dần nhưng kết quả không cập nhật vào A. Hàm trả lại dãy số mới là dãy vừa được sắp xếp (gồm các phần tử của dãy A).
2. Nếu không cần cập nhật vào dãy A mà chỉ cần in ra màn hình các phần tử của A theo thứ tự tăng dần thì cần sửa lại chương trình sắp xếp trên như thế nào?



LUYỆN TẬP

1. Dựa trên hàm `BSTSort(A)` đã biết, viết chương trình sắp xếp dãy số giảm dần theo kĩ thuật sử dụng cây tìm kiếm nhị phân.
2. Thuật toán sắp xếp dãy sử dụng cây tìm kiếm nhị phân có độ phức tạp thời gian là bao nhiêu?



VẬN DỤNG

1. Dựa trên tính chất của cây tìm kiếm nhị phân, hãy viết hàm `minimum(T)` và `maximum(T)` tính giá trị khoá nhỏ nhất và lớn nhất của cây tìm kiếm nhị phân T.
2. Viết hàm `height(T)` tính chiều cao của cây tìm kiếm nhị phân T.

BÀI 10 THỰC HÀNH TỔNG HỢP VỚI CÂY TÌM KIẾM NHỊ PHÂN

Sau bài học này em sẽ:

- Thực hành các thuật toán duyệt trên cây tìm kiếm nhị phân: các thao tác thêm, tìm kiếm, duyệt dữ liệu với cây tìm kiếm nhị phân.

Trong Bài 9, chúng ta đã học thao tác duyệt cây. Với bài toán thực tế quản lí danh bạ điện thoại, làm thế nào để sử dụng các thao tác đó vào cây tìm kiếm nhị phân để thêm, tìm kiếm, hiển thị toàn bộ các liên hệ theo thứ tự sắp xếp của tên liên hệ trong danh bạ?

Nhiệm vụ: Viết chương trình quản lí danh bạ điện thoại

Các thiết bị máy tính, điện thoại hiện nay đều tích hợp ứng dụng quản lí danh bạ. Em hãy sử dụng cấu trúc dữ liệu cây tìm kiếm nhị phân để viết ứng dụng quản lí danh bạ đơn giản. Mỗi liên hệ trong danh bạ gồm các thông tin: tên liên hệ (duy nhất), tên đầy đủ, số điện thoại. Khi chạy chương trình, dữ liệu được đọc từ tệp contacts.inp với mỗi dòng ứng với một liên hệ có dạng như hình trên.

contacts.inp

```
Anh An, Nguyễn Văn An, 0901.000.159
Bố, Nguyễn Văn Hoàng, 0983 000 131
Mẹ, Hoàng Thị Hồng, 0962 000 481
ICTLab Station, Số cố định tại ICTLab,
024 124 000 313
```

Các chức năng chính của chương trình:

a) Hiển thị danh sách liên hệ theo thứ tự sắp xếp tên theo thứ tự từ điển.

b) Tìm kiếm liên hệ theo tên.

c) Thêm, sửa các liên hệ.

Hướng dẫn

Xây dựng chương trình gồm hai bước: (1) Cài đặt cây tìm kiếm nhị phân bằng mảng để lưu thông tin của các liên hệ, mỗi liên hệ gồm ba thông tin là tên liên hệ, tên đầy đủ, số điện thoại; (2) Xây dựng chương trình hoàn chỉnh đọc dữ liệu từ tệp contacts.inp, lưu dữ liệu vào cây tìm kiếm nhị phân, cho phép người dùng tra cứu, bổ sung, cập nhật danh bạ và in danh bạ theo thứ tự từ điển.

Bước 1. Cài đặt cây tìm kiếm nhị phân bằng mảng.

Phân tích: Các cấu trúc dữ liệu của cây tìm kiếm nhị phân lưu trữ danh bạ tương tự như mã nguồn mẫu từ Bài 7 với hàm định nghĩa cấu trúc Tree, hàm `left`, `right`, `parent` được giữ nguyên, chỉ có hàm `Tree_Insert` và `Tree_Insert_Update` cần được sửa đổi để tích hợp thêm các trường dữ liệu *key* (khoá tìm kiếm), *fullName* (tên đầy đủ của liên hệ) và *phoneNumber* (số điện thoại) vào mỗi nút của cây. Để duyệt

các nút từ nhỏ đến lớn với khoá tìm kiếm được sắp xếp theo thứ tự từ điển, em hãy dùng hàm `inorder` được giới thiệu ở Bài 9 nhưng cần chỉnh sửa để in các thông tin trong khi duyệt.

Các hàm `Tree_Insert`, `Tree_Insert_Update`, và `inorder` được sửa như sau:

```
1  def Tree_Insert(T, key, fullName, phoneNumber):
2      k = 0
3      while k < len(T) and T[k] != None:
4          if key < T[k][0]:
5              k = left(k)
6          elif key > T[k][0]:
7              k = right(k)
8          else:
9              return
10     if k >= len(T):
11         T.extend([None]*(k - len(T) + 1))
12     T[k] = [key, fullName, phoneNumber]
13 def Tree_Insert_Update(T, key, fullName, phoneNumber):
14     k = search(T, 0, key)
15     if k == -1:
16         Tree_Insert(T, key, fullName, phoneNumber)
17         return False #Thể hiện đã thêm món mới
18     else:
19         T[k][1] = fullName #Cập nhật tên
20         T[k][2] = phoneNumber #Cập nhật số điện thoại
21         return True #Thể hiện đã cập nhật liên hệ
22 def inorder(T, k):
23     if k < len(T) and T[k] != None:
24         inorder(T,left(k))
25         print(T[k][0], "|", T[k][1], "|", T[k][2], end = " ")
26         inorder(T,right(k))
```

Bước 2. Xây dựng chương trình hoàn chỉnh.

Để xây dựng chương trình hoàn chỉnh, trước tiên cần đọc dữ liệu từ tệp contacts.inp rồi chèn vào cây tìm kiếm nhị phân. Tương tự Bài 8, chương trình có bảng chọn cho phép người dùng nhập số ứng với các lựa chọn như sau:

```
0: "Thoát chương trình"
1: "Hiển thị các liên hệ theo thứ tự từ điển"
2: "Tra cứu liên hệ"
3: "Cập nhật hoặc thêm một liên hệ"
```

Mã nguồn chương trình có thể như sau:

```
1  from BST_phonebook import * #Khai báo thư viện cây tìm kiếm nhị phân
2
3  #Khởi tạo cây tìm kiếm và đọc dữ liệu từ tệp
4  T = Tree()
5  f = open("contacts.inp", "r", encoding="utf8")
6  lines = f.readlines()
7  for line in lines:
8      l = line.strip()
9      l = l.split(",", maxsplit=3)
10     Tree_Insert(T, l[0], l[1], l[2])
```

```
f.close()
#Hiển thị các chức năng chương trình và xử lí theo lựa chọn
options = ["Thoát chương trình", "Hiển thị các liên hệ theo thứ tự từ điển", "Tra cứu liên hệ", "Cập nhật hoặc thêm một liên hệ"]
selected = -1 #Chức năng người dùng chọn lựa từ menu, khởi tạo bằng -1
while selected != 0:
    for choice in range(len(options)):
        print(choice, "-", options[choice] )
    selected = int(input("Chọn chức năng: "))
    if selected < 0 or selected > 4:
        print("Chức năng không hợp lệ, hãy nhập số 0 đến 4")
        continue
    if selected == 1: #In danh bạ
        print("Danh bạ theo thứ tự từ điển:")
        inorder(T, 0)
    elif selected == 2: #Tìm liên hệ
        item = input("Nhập tên liên hệ: ")
        k = search(T, 0, item)
        if k >= 0:
            print("Tên đầy đủ:", T[k][1], "| Số điện thoại:", T[k][2])
        else:
            print("Tên liên hệ", item, "không có trong danh bạ")
    elif selected == 3: #Thêm hoặc cập nhật liên hệ
        key = input("Nhập tên liên hệ: ")
        fullName = input("Nhập tên đầy đủ: ")
        phoneNumber = input("Nhập số điện thoại: ")
        if Tree_Insert_Update(T, key, fullName, phoneNumber):
            print("Đã cập nhật liên hệ", key)
        else:
            print("Đã thêm liên hệ", key, "vào danh bạ")
```

LUYỆN TẬP

1. Hãy vẽ cây tìm kiếm nhị phân ứng với:
   a) Dữ liệu tệp contacts.inp ở trong phần thực hành.
   b) Từ cây nhận được ở ý a, thêm liên hệ "Anh, Nguyễn Văn Tùng, 0982 000 134".
2. Tiếp tục với ứng dụng quản lí danh bạ, chức năng hiển thị danh sách liên hệ theo thứ tự từ điển. Do hạn chế của màn hình, mỗi trang chỉ hiển thị được 20 liên hệ. Hãy thêm tính năng in các liên hệ ở trang n bất kì do người dùng nhập vào, điều kiện n nguyên, lớn hơn 0 và nhỏ hơn hoặc bằng tổng số trang có thể hiển thị.

VẬN DỤNG

1. Sử dụng cây tìm kiếm nhị phân để viết chương trình quản lí danh sách học sinh của một lớp. Thông tin mỗi học sinh gồm mã (duy nhất), tên đầy đủ, ngày sinh. Chương trình cho phép thêm mới thông tin các học sinh, in danh sách sắp xếp theo mã từ nhỏ đến lớn và từ lớn đến nhỏ, tìm kiếm học sinh theo mã.
2. Sử dụng cây tìm kiếm nhị phân để hiển thị các món trong tệp menu.inp ở Bài 8 theo thứ tự giá tiền tăng dần. Mỗi dòng in ra gồm tên món và giá tiền. Nếu có hai hoặc nhiều món cùng giá tiền thì các món đó được hiển thị theo thứ tự xuất hiện trong tệp menu.inp.

Chuyên đề
3

TÌM HIỂU KĨ THUẬT DUYỆT ĐỒ THỊ VÀ ỨNG DỤNG

BÀI 11 KHÁI NIỆM ĐỒ THỊ

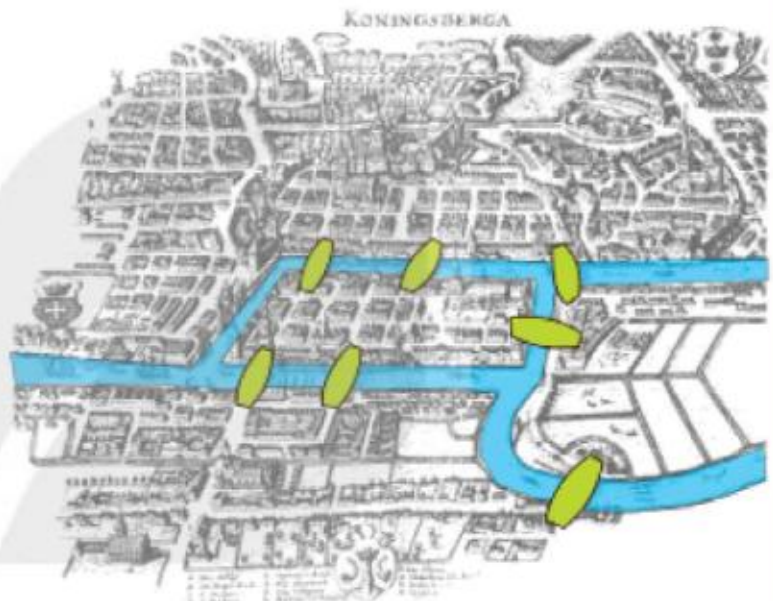
Sau bài học này em sẽ:

- Biết và trình bày được khái niệm đồ thị và các định nghĩa có liên quan.
- Mô tả được cấu trúc và ý nghĩa của ma trận kề và danh sách kề.

Năm 1736, nhà bác học Euler đưa ra bài toán, được gọi là bài toán 7 cây cầu ở Königsberg. Tại thành phố cổ Königsberg của nước Phổ cũ (nay thuộc nước Nga) có dòng sông Pregel vắt ngang qua, chia thành phố thành các vùng riêng biệt. Bài toán Euler đặt ra là làm sao đi qua tất cả 7 cây cầu này, mỗi cầu chỉ được phép đi qua đúng một lần.

Em hãy giải bài toán trên.

Có thể dùng mô hình dữ liệu nào để mô phỏng bài toán này?



Hình 11.1 Sơ đồ các cây cầu trong thành phố cổ Königsberg

1. Khái niệm đồ thị

Hoạt động 1 **Tìm hiểu khái niệm đồ thị**

1. Trao đổi, thảo luận về mô hình đồ thị, các khái niệm cơ bản của đồ thị và trả lời câu hỏi: Bài toán trong phần khởi động có thể biểu diễn được bằng mô hình đồ thị không?

2. Em hãy tìm một số bài toán thực tế khác có thể biểu diễn được bằng đồ thị.

a) Khái niệm đồ thị

Đồ thị G = (V, E) là tập hợp hữu hạn các đỉnh V và tập hợp các cạnh E nối các đỉnh V với nhau. Giả sử đồ thị G = (V, E) có n đỉnh và m cạnh, ta có các kí hiệu sau:

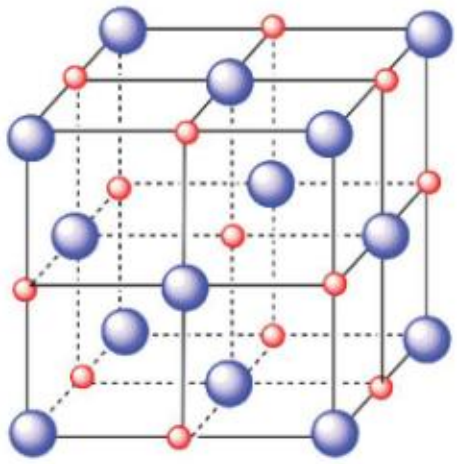
– Tập hợp các đỉnh $V = \{v_1, v_2, v_3, \ldots, v_n\}$;

– Tập hợp các cạnh $E = \{e_1, e_2, e_3, \ldots, e_m\}$.

Trong đó, các cạnh có dạng $e = (v_i, v_j)$ nối hai đỉnh v_i và v_j.

Có rất nhiều mô hình đồ thị trong thực tế (Hình 11.2).

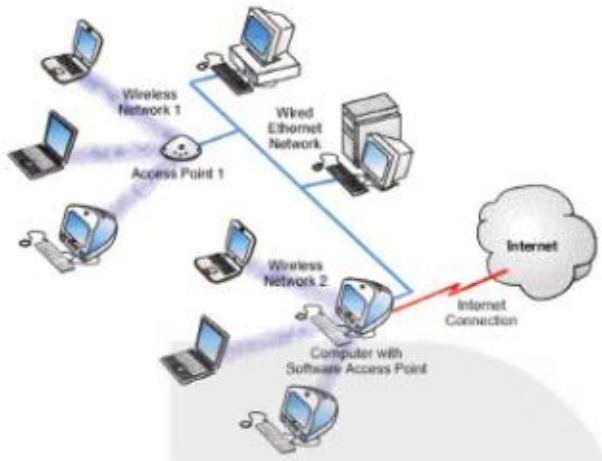
KẾT NỐI TRI THỨC VỚI CUỘC SỐNG



a) Cấu trúc tinh thể liên kết ion của muối ăn



b) Sơ đồ các tuyến xe buýt trong một thành phố



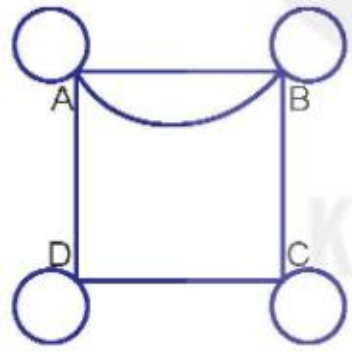
c) Mạng Internet kết nối máy tính toàn cầu



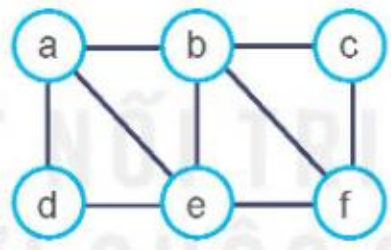
d) Mô hình liên kết siêu văn bản của các trang web trên Internet

Hình 11.2. Một số mô hình đồ thị trong thực tế

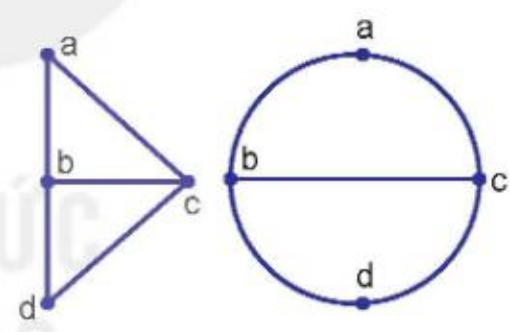
Đồ thị được mô tả bằng cách vẽ các nút để mô tả các đỉnh và các đường nối giữa các đỉnh để mô tả các cạnh của đồ thị. Ví dụ một số đồ thị như Hình 11.3.



a) Đồ thị với 4 đỉnh có tên A, B, C, D



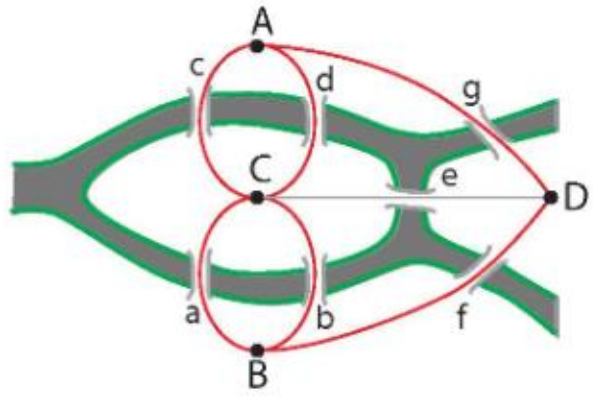
b) Các đỉnh của đồ thị có tên a, b, c, d, e, f



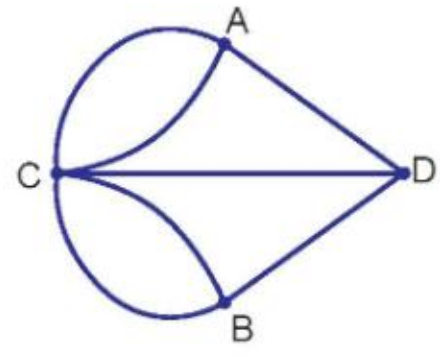
c) Hai sơ đồ trên là tương đương, biểu diễn cùng một đồ thị

Hình 11.3. Một số đồ thị

Mô hình thành phố và các cây cầu của bài toán 7 cây cầu ở Königsberg (Hình 11.4a) được mô phỏng lại để dễ quan sát hơn với các vùng đất được kí hiệu là A, B, C, D và các cây cầu đóng vai trò các cạnh nối những vùng đất này. Mô hình đồ thị được biểu diễn như Hình 11.4b. Như vậy, bài toán 7 cây cầu có thể phát biểu lại như sau:



a)



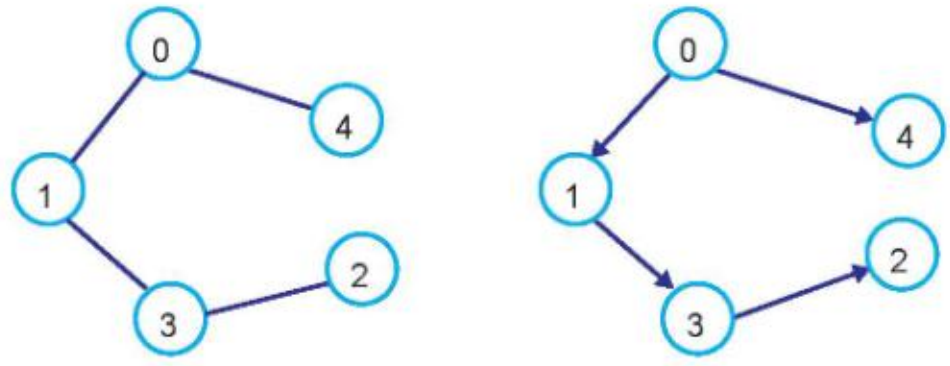
b)

Hình 11.4. Mô hình đồ thị bài toán 7 cây cầu ở Königsberg

Cho mô hình đồ thị như Hình 11.4b, hãy tìm đường đi qua tất cả các cạnh, mỗi cạnh đi qua đúng một lần.

b) Đồ thị vô hướng và đồ thị có hướng

Đồ thị vô hướng có các cạnh nối không phân biệt hướng, có thể đi được hai chiều, các cạnh được biểu diễn bằng các đoạn thẳng (Hình 11.5a). Đồ thị có hướng có mũi tên chỉ hướng trên các cạnh, chỉ đi theo hướng có mũi tên.



a) Đồ thị vô hướng *b) Đồ thị có hướng*

Hình 11.5. Đồ thị vô hướng và đồ thị có hướng

c) Đơn đồ thị

Nếu đồ thị có cạnh e = (v, v), tức là xuất phát và kết thúc tại một đỉnh, thì được gọi là có khuyên (Hình 11.6a). Nếu giữa hai đỉnh u, v có nhiều hơn một cạnh nối thì được gọi là có cạnh song song (Hình 11.6b, c).



a) Khuyên *b) Cạnh song song* *c) Cạnh song song có hướng*

Hình 11.6. Khuyên và cạnh song song trong đồ thị

Đồ thị G = (V, E) được gọi là đơn đồ thị nếu đồ thị không có khuyên và không có cạnh song song. Với đơn đồ thị, giữa hai đỉnh bất kì của đồ thị có nhiều nhất một cạnh nối. Trong phạm vi cuốn sách này, chúng ta chỉ xét các đơn đồ thị.

Trong Python, chúng ta sẽ sử dụng kiểu dữ liệu list để mô tả V và E. Mỗi cạnh là một cặp hai chỉ số mô tả cặp đỉnh tương ứng. Nếu đồ thị vô hướng thì sử dụng tập hợp để mô tả cạnh. Nếu đồ thị có hướng thì dùng list hoặc tuple để mô tả cạnh có hướng giữa hai đỉnh.

Ví dụ đồ thị vô hướng (Hình 11.5a) được biểu diễn trong Python như sau:

```
V = [0, 1, 2, 3, 4]; E = [{0,1}, {0,4}, {1,3}, {2,3}].
```

Với đồ thị có hướng (Hình 11.5b) thì danh sách các cạnh E được mô tả như sau:

```
E = [(0,1), (0,4), (1,3), (3,2)].
```

> Đồ thị G là tập hợp hữu hạn các đỉnh V và tập hợp các cạnh E nối các đỉnh của V với nhau, kí hiệu là G = (V, E). Có hai loại đồ thị là đồ thị vô hướng (các cạnh không phân biệt hướng, có thể đi theo hai chiều) và đồ thị có hướng (các cạnh có hướng chỉ đi theo chiều mũi tên).

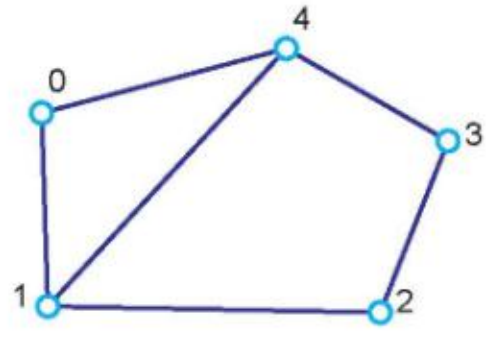


1. Cây, cây nhị phân và cây tìm kiếm nhị phân có là mô hình đồ thị không?

2. Vẽ đồ thị vô hướng G = (V, E) sau:

```
V = [0, 1, 2, 3, 4]
E = [{0,1}, {0,4}, {1,2}, {1,3}, {2,4}]
```

3. Mô tả tập hợp đỉnh V và tập hợp cạnh E của đồ thị vô hướng trong Hình 11.7.



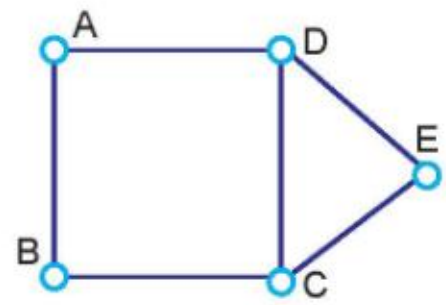
Hình 11.7. Đồ thị vô hướng

2. Một số khái niệm liên quan đến đồ thị

Hoạt động 2 **Tìm hiểu một số khái niệm, định nghĩa liên quan đến đồ thị**

Đọc, trao đổi và thảo luận các khái niệm, định nghĩa liên quan đến đồ thị. Quan sát đồ thị ở Hình 11.8 và thực hiện các yêu cầu sau:

1. Kể tên các đỉnh kề với D.
2. Hãy cho biết bậc của đỉnh A.
3. Liệt kê một vài đường đi từ đỉnh A đến đỉnh E.



Hình 11.8. Đồ thị vô hướng

Cho đơn đồ thị G = (V, E), có thể vô hướng hoặc có hướng.

Nếu từ đỉnh u có cạnh nối đến đỉnh v thì chúng ta gọi v là đỉnh kề của u. Nếu G là vô hướng thì nếu v là đỉnh kề của u thì u cũng là đỉnh kề của v. Nếu cạnh e từ u đến v thì chúng ta sẽ kí hiệu e: $u \to v$, hay $u \xrightarrow{e} v$.

Bậc (degree) của đỉnh u, kí hiệu deg(u), là số lượng các đỉnh kề với u.

Nếu G là đồ thị có hướng chúng ta sẽ có các định nghĩa sau:

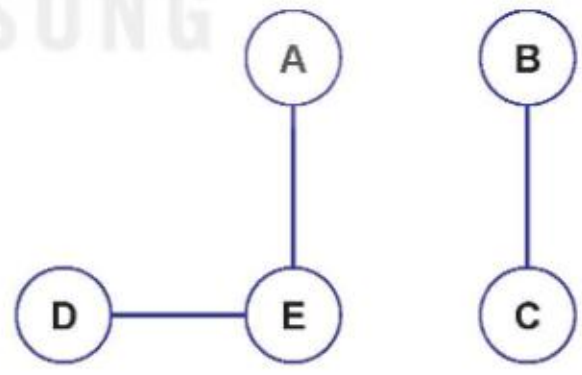
Bậc ra (out degree) của đỉnh u, kí hiệu deg+(v) là số lượng các đỉnh kề với u.

Bậc vào (in degree) của đỉnh u, kí hiệu deg-(v), là số các đỉnh có cạnh nối đến u.

Đường đi (path) từ đỉnh s đến đỉnh t là dãy các cạnh kề nhau e_1, e_2,..., e_k nối từ đỉnh s đến đỉnh t và thoả mãn điều kiện: tồn tại dãy các đỉnh kề nhau và khác nhau từng đôi một v_0, v_1, v_2,..., v_k, sao cho e_i là cạnh nối đỉnh v_{i-1} đến v_i (i = 1, 2,..., k), $v_0 = s$, và $v_k = t$.

Vì các đỉnh v_0, v_1,..., v_k khác nhau từng đôi một, do đó các cạnh e_1, e_2,..., e_k cũng khác nhau từng đôi một. Trong trường hợp đỉnh v_0 trùng với đỉnh v_k thì dãy các cạnh kề nhau e_1, e_2,..., e_k ở trên là chu trình (cycle).

• Đồ thị G = (V, E) được gọi là liên thông nếu với mọi đỉnh u, v ∈ V thì tồn tại đường đi từ u đến v. Nếu G không là liên thông thì tập hợp V có thể được chia thành các tập hợp con rời nhau mà mỗi đồ thị con xác định bởi các tập hợp này là liên thông. Người ta gọi các tập hợp con đó là các thành phần liên thông của G (Hình 11.9). Theo quy ước, nếu đồ thị G chỉ có một đỉnh thì đồ thị này là liên thông.



Hình 11.9. Đồ thị có hai thành phần liên thông là [A, D, E] và [B, C].

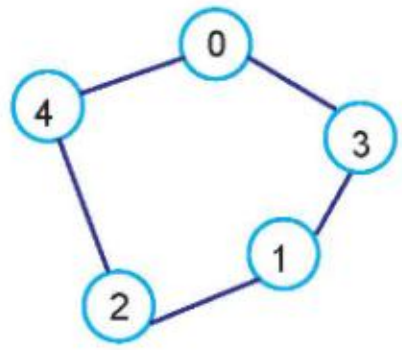
• Cho đồ thị G = (V, E), V có n đỉnh. Các đỉnh V được kí hiệu 0, 1, 2,..., n – 1. Khi đó ma trận kề (Adjacency Matrix) của đồ thị G là ma trận có kích thước n × n, được định nghĩa như sau:

$$A = [a_{ij}], a_{ij} = \begin{cases} 1 & \text{nếu } (i,j) \in E \\ 0 & \text{nếu } (i,j) \notin E \end{cases}$$

Lưu ý: Ma trận kề được định nghĩa cho cả đồ thị vô hướng và có hướng. Nếu G là đồ thị vô hướng thì ma trận kề là đối xứng, tức là $a_{ij} = a_{ji}$ với mọi i, j.

Với mỗi $u \in V$, danh sách các đỉnh kề của u là $Adj(u) = \{v \mid (u,v) \in E\}$. Danh sách kề (Adjacency List) của đồ thị G, kí hiệu là Adj, là tập hợp danh sách đỉnh kề của các đỉnh của G.

Với đồ thị vô hướng trong Hình 11.10a, ta có ma trận kề và danh sách kề như Hình 11.10b và Hình 11.10c.



a) Đồ thị G = (V,E)

$$\begin{pmatrix} 0&0&0&1&1\\ 0&0&1&1&0\\ 0&1&0&0&1\\ 1&1&0&0&0\\ 1&0&1&0&0 \end{pmatrix}$$

b) Ma trận kề

$$\begin{pmatrix} 0 \to 3,4\\ 1 \to 2,3\\ 2 \to 1,4\\ 3 \to 0,1\\ 4 \to 0,2 \end{pmatrix}$$

c) Danh sách kề

Hình 11.10. Đồ thị và ma trận kề, danh sách kề tương ứng

Một số khái niệm, định nghĩa quan trọng liên quan đến đồ thị bao gồm bậc của các đỉnh, đường đi từ một đỉnh đến đỉnh khác, ma trận kề và danh sách kề. Tất cả các khái niệm này được định nghĩa cho cả hai kiểu đồ thị vô hướng và có hướng.

1. Khi nào một đỉnh của đồ thị có bậc bằng 0?

2. Xác định ma trận kề và danh sách kề của các đồ thị ở Hình 11.11.

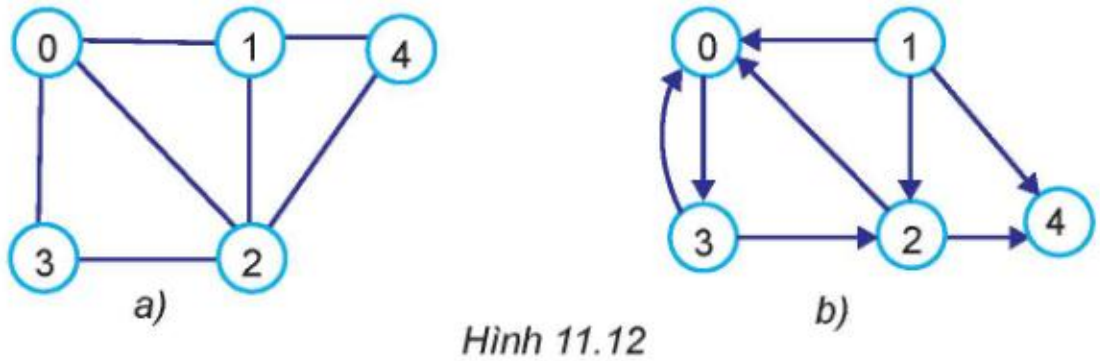


Hình 11.11. Các đồ thị

3. Biểu diễn đồ thị

Hoạt động 3 **Tìm hiểu cách biểu diễn dữ liệu đồ thị trên máy tính**

Tìm hiểu một số cách biểu diễn dữ liệu đồ thị trên máy tính. Thảo luận xem cách nào là hợp lí nhất. Hãy biểu diễn dữ liệu của các đồ thị ở Hình 11.12.



Hình 11.12

Cho đồ thị G = (V, E). Ta cần biểu diễn dữ liệu đồ thị G trên máy tính như thế nào?

• Bộ dữ liệu của đồ thị được cho bởi dãy các đỉnh V và dãy các cạnh E. Các đỉnh của đồ thị được đánh số từ 0 đến n – 1, ta có dãy các đỉnh V như sau:

$$V = \{v_0, v_1, \ldots, v_{n-1}\}.$$

Mỗi cạnh là một cặp hai đỉnh hoặc hai chỉ số tương ứng của hai đỉnh. Nếu G là đồ thị vô hướng thì cạnh là hai chiều, nếu G là đồ thị có hướng thì mỗi cạnh là cặp có thứ tự các đỉnh hoặc chỉ số của các đỉnh. Khi đó ta có dãy các cạnh E như sau:

$$E = \{e_0, e_1, \ldots, e_k\}.$$

Mỗi cạnh có dạng $e = (v_i, v_j)$ hoặc $e = (i, j)$.

Ma trận kề A và danh sách kề Adj đã được định nghĩa trong mục 2.

• Để thiết lập và biểu diễn dữ liệu đồ thị trên máy tính, thực hiện như sau:

– Tập các đỉnh V được đánh số từ 0 đến n – 1:

```
V = [0, 1, 2,..., n - 1]
```

Với đồ thị ở Hình 11.12a ta có n = 5 và V được xác định như sau:

```
V = [0, 1, 2, 3, 4]
```

– Tập các cạnh E là dãy (danh sách) các cạnh, mỗi cạnh sẽ là một cặp hai chỉ số tương ứng với hai đỉnh. Mỗi cạnh của đồ thị vô hướng E được biểu diễn như một tập hợp, ví dụ e = {i, j}. Mỗi cạnh của đồ thị có hướng là cặp hai chỉ số có thứ tự, ví dụ kiểu tuple là e = (i, j) hoặc list là e = [i, j].

Với hai đồ thị trong Hình 11.12, E được định nghĩa như sau:

a) `E = [{0,1},{0,2},{0.3},{1,2},{1,4},{2,4}]`

b) `E = [(0,3),(1,0),(1,2),(1,4),(2,0),(2,4),(3,0),(3,2)]`

– Ma trận kề A của đồ thị là mảng hai chiều kích thước n × n được định nghĩa như sau: A[i][j] = 1 khi và chỉ khi tồn tại cạnh nối từ đỉnh v_i đến v_j. Ma trận kề của các đồ thị trong Hình 11.12 tương ứng như Hình 11.13.

$$\begin{pmatrix} 0&1&1&1&0\\ 1&0&1&0&1\\ 1&1&0&1&1\\ 1&0&1&0&0\\ 0&1&1&0&0 \end{pmatrix} \qquad \begin{pmatrix} 0&0&0&1&0\\ 1&0&1&0&1\\ 1&0&0&0&1\\ 1&0&1&0&0\\ 0&0&0&0&0 \end{pmatrix}$$

a) *b)*

Hình 11.13. Ma trận kề

– Danh sách kề Adj của đồ thị được định nghĩa như sau: A[i] là danh sách các đỉnh kề với đỉnh v_i. Với định nghĩa này, ta có các bảng biểu diễn Adj của các đồ thị trong Hình 11.12 tương ứng như Hình 11.14.

Đỉnh	Danh sách kề
0	1 2 3
1	0 2 4
2	0 1 3 4
3	0 2
4	1 2

a) Danh sách kề của đồ thị trong Hình 11.10a

Đỉnh	Danh sách kề
0	3
1	0 2 4
2	0 4
3	0 2
4	<rỗng>

b) Danh sách kề của đồ thị trong Hình 11.10b

Hình 11.14. Danh sách kề

Có nhiều cách thể hiện dữ liệu Adj trên máy tính. Cách thứ nhất là dùng kiểu dữ liệu *danh sách liên kết* (Linked List). Theo cách này, Adj là dãy các phần tử, phần tử thứ i, Adj[i] là một Linked List các phần tử là đỉnh kề với đỉnh thứ i của đồ thị.

Cách thứ hai, sử dụng list trong Python. Theo cách này thì danh sách kề Adj của các đồ thị ở Hình 11.12 tương ứng như Hình 11.15.

```
Adj = [[1,2,3],
[0,2,4],
[0,1,3,4],
[0,2],
[1,2]]
```

a)

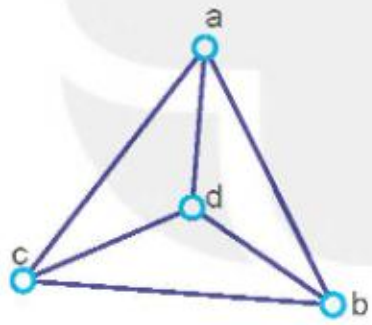
```
Adj = [[3],
[0,2,4],
[0,4],
[0,2],
[]]
```

b)

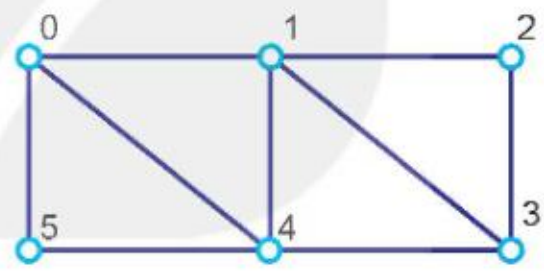
Hình 11.15. Danh sách kề của các đồ thị ở Hình 11.12

Để biểu diễn dữ liệu đồ thị trong máy tính, người ta thường sử dụng Ma trận kề hoặc Danh sách kề. Ma trận kề là mảng hai chiều. Danh sách kề thường được biểu diễn bằng danh sách liên kết. Trong Python, ma trận kề và danh sách kề là các danh sách (kiểu list).

Thiết lập bộ dữ liệu biểu diễn gồm (n, V, E, A, Adj) cho các đồ thị **sau**:



a)



b)

Hình 11.16. Các đồ thị

LUYỆN TẬP

1. Đồ thị ứng với mô hình bài toán 7 cây cầu ở Königsberg có phải là đơn đồ thị không? Tính bậc của các đỉnh của đồ thị đó.

2. Cho đồ thị G vô hướng với ma trận kề như hình bên.

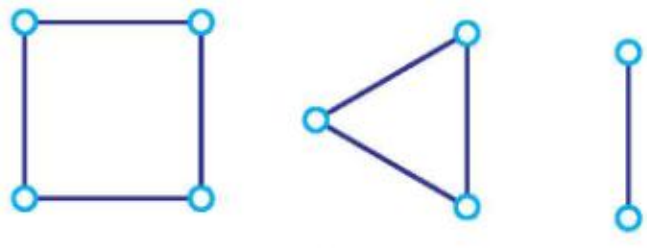
Hãy vẽ đồ thị trên.

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

VẬN DỤNG

1. Đồ thị vô hướng G được gọi là đầy đủ nếu giữa hai đỉnh bất kì (khác nhau) đều có cạnh nối. Hãy vẽ và thiết lập ma trận kề của đồ thị đầy đủ với số đỉnh n = 2, 3, 4.

2. Đồ thị trong Hình 11.17 có bao nhiêu thành phần liên thông?



Hình 11.17. Thành phần liên thông

KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

BÀI 12 BIỂU DIỄN ĐỒ THỊ

Sau bài học này em sẽ:

- Thiết lập và biểu diễn được đồ thị bằng ma trận kề và danh sách kề.

Quan sát đồ thị Hình 12.1 và cho biết mỗi tệp dữ liệu sau có ý nghĩa gì.

```
5
0 1
0 4
1 4
2 3
2 4
3 4
```

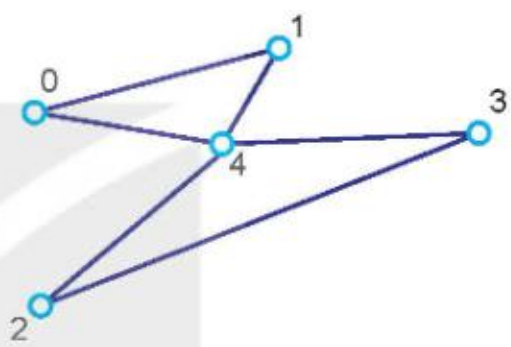
Tệp 1

```
5
0 1 0 0 1
1 0 0 0 1
0 0 0 1 1
0 0 1 0 1
1 1 1 1 0
```

Tệp 2

```
5
0 1 4
1 0 4
2 3 4
3 2 4
4 0 1 2 3
```

Tệp 3



Đồ thị

Hình 12.1. Đồ thị và dữ liệu mô tả đồ thị

1. Mô hình dữ liệu đồ thị

Hoạt động 1 **Tìm hiểu các mô hình dữ liệu đồ thị (vô hướng hoặc có hướng)**

Tìm hiểu, thảo luận về các cách biểu diễn dữ liệu của một đồ thị G.

Có nhiều cách biểu diễn dữ liệu của một đồ thị, trong đó thường gặp ba cách sau:

– Sử dụng danh sách các cạnh của đồ thị.

– Sử dụng ma trận kề kích thước n × n.

– Sử dụng danh sách kề.

Trong các cách trên yêu cầu có thêm giá trị n là số đỉnh hay kích thước của đồ thị. Sau đây, chúng ta cùng tìm hiểu tệp dữ liệu tương ứng với các cách biểu diễn đồ thị khác nhau.

a) Dữ liệu danh sách các cạnh của đồ thị

Tệp dữ liệu loại này (Hình 12.2) có dạng như sau:

– Dòng đầu tiên là số n.

– Các dòng tiếp theo, mỗi dòng là hai chỉ số mô tả một cạnh của đồ thị.

```
n
i1 j1
i2 j2
.....
im jm
```

Hình 12.2. Tệp danh sách các cạnh

b) Dữ liệu ma trận kề của đồ thị

Tệp dữ liệu loại này (Hình 12.3) có dạng sau:

– Dòng đầu tiên là số n.

– n dòng tiếp theo là dữ liệu ma trận kề A.

```
n
a11 a12 ... a1n
a21 a22 ... a2n
.....
an1 an2 ... ann
```

Hình 12.3. Tệp ma trận kề

c) Dữ liệu danh sách kề của đồ thị

Tệp dữ liệu loại này (Hình 12.4) có dạng sau:

– Dòng đầu tiên là số n.

– n dòng tiếp theo là dữ liệu danh sách kề Adj, cụ thể như sau: Dòng thứ i sẽ bắt đầu bằng số i, sau đó danh sách các đỉnh là kề của i, mỗi đỉnh ghi số thứ tự của đỉnh, cách nhau bởi dấu cách.

```
n
0 s0 t0 ... u0
1 s1 t1 ... u1
.....
n-1 sn-1 tn-1 ... un-1
```

Hình 12.4. Tệp danh sách kề

Lưu ý: Yêu cầu thành phần đầu tiên của dòng thứ i là số i chỉ có ý nghĩa hình thức. Tuy nhiên cách định nghĩa này là cần thiết khi đỉnh i của đồ thị là biệt lập, tức là không có các đỉnh kề. Khi đó dòng thứ i chỉ có đúng một giá trị.

Có nhiều cách thiết lập tệp dữ liệu biểu diễn đồ thị. Các cách thường dùng là tệp dữ liệu danh sách các cạnh, ma trận kề hoặc danh sách kề.



1. Vẽ đồ thị có tệp dữ liệu ma trận kề Hình 12.5.
2. Có thể có hai tệp dữ liệu dạng danh sách kề khác nhau nhưng biểu diễn hai đồ thị hoàn toàn giống nhau không?

KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

```
4
0 0 1 1
0 0 1 1
1 1 0 1
1 1 1 0
```

Hình 12.5. Tệp ma trận kề

2. Thiết lập đồ thị từ tệp ma trận kề và tệp danh sách kề

Hoạt động 2 **Tìm hiểu cách thiết lập đồ thị từ ma trận kề và danh sách kề**

Tìm hiểu, thảo luận cách thiết lập đồ thị (dữ liệu của đồ thị) trong trường hợp tệp dữ liệu biểu diễn là ma trận kề hoặc danh sách kề.

a) Thiết lập đồ thị từ tệp ma trận kề

Tệp dữ liệu ma trận kề có dòng đầu tiên là n (số đỉnh của đồ thị), n dòng tiếp theo mô tả ma trận kề của đồ thị. Chương trình sau đây đọc tệp dữ liệu này để tạo dữ liệu biểu diễn đồ thị theo ma trận kề. Chương trình sẽ áp dụng cho cả đồ thị vô hướng và đồ thị có hướng. Hàm `BuildGraph(fname)` đọc dữ liệu từ tệp `fname` và trả về bộ dữ liệu V, A với V là danh sách các đỉnh, A là ma trận kề.

Chương trình 2a. Thiết lập đồ thị từ tệp dữ liệu ma trận kề. Áp dụng cho đồ thị vô hướng và đồ thị có hướng.

```
1 def BuildGraph(fname):
2     f = open(fname)
3     n = int(f.readline())
4     V = [i for i in range(n)]
5     A = [] # thiết lập A là list rỗng
6     for line in f:
7         A.append([int(ch) for ch in line.split()]) # bổ sung vào A
8     f.close()
9     return V,A
```

b) Thiết lập đồ thị từ tệp danh sách kề

Tệp dữ liệu danh sách kề có dòng đầu tiên là n (số đỉnh của đồ thị), n dòng tiếp theo mô tả danh sách kề của đồ thị. Chương trình sau đây đọc tệp dữ liệu đầu vào và tạo bộ dữ liệu biểu diễn đồ thị theo danh sách kề. Chương trình sẽ áp dụng cho cả đồ thị vô hướng và có hướng. Hàm `BuildGraph(fname)` sẽ đọc dữ liệu từ tệp có tên `fname` và trả về bộ dữ liệu V, Adj với V là danh sách các đỉnh, Adj là danh sách kề.

Chương trình 2b. Thiết lập đồ thị từ tệp dữ liệu danh sách kề. Áp dụng cho đồ thị vô hướng và đồ thị có hướng.

```python
def BuildGraph(fname):
    f = open(fname)
    n = int(f.readline())
    V = [i for i in range(n)]
    Adj = [[] for i in range(n)] # thiết lập Adj gồm n dãy rỗng
    for i in range(n):
        line = [int(x) for x in f.readline().split()]
        line = line[1: ]          # Lấy dãy các số từ vị trí thứ 2
        Adj[i].extend(line)       # bổ sung vào hàng thứ i của Adj
    f.close()
    return V,Adj
```

Có thể thiết lập đồ thị từ tệp dữ liệu biểu diễn là ma trận kề hoặc danh sách kề. Các chương trình này có thể áp dụng cho cả đồ thị vô hướng và đồ thị có hướng.



1. Khẳng định dãy Adj[i] có số lượng phần tử bằng số các phần tử có giá trị 1 của hàng thứ i của ma trận kề A là đúng hay sai?
2. Khi nào ma trận kề A chỉ gồm toàn số 0?

3. Thiết lập đồ thị từ danh sách các cạnh

Hoạt động 3 **Tìm hiểu cách thiết lập dữ liệu đồ thị từ tệp dữ liệu danh sách các cạnh**

Tìm hiểu, thảo luận cách thiết lập dữ liệu của đồ thị trong trường hợp tệp dữ liệu biểu diễn danh sách các cạnh.



Tệp dữ liệu danh sách các cạnh có dòng đầu tiên là n (số đỉnh của đồ thị), các dòng tiếp theo mô tả danh sách các cạnh, mỗi dòng có hai số i, j cách nhau bởi dấu cách. Mỗi dòng ứng với một cạnh nối đỉnh i đến đỉnh j của đồ thị. Nếu đồ thị có hướng thì cần cập nhật một lần vào ma trận kề và danh sách kề của đồ thị. Nếu đồ thị vô hướng thì cần cập nhật hai lần. Ví dụ với dòng ghi i, j thì cần cập nhật cho ma trận kề là A[i][j] = 1 và A[j][i] = 1, cập nhật cho danh sách kề là `A[i].append(j)` và `A[j].append(i)`. Do vậy, chương trình cho đồ thị vô hướng khác với chương trình cho đồ thị có hướng.

a) Trường hợp đồ thị vô hướng

Dữ liệu đầu vào của đồ thị được cho bởi tệp fname lưu thông tin danh sách các cạnh của đồ thị. Hàm `BuildGraph(fname)` đọc dữ liệu từ tệp fname và trả lại dữ liệu của đồ thị.

Chương trình 3.1a. Thiết lập đồ thị biểu diễn bởi ma trận kề từ tệp dữ liệu danh sách các cạnh, áp dụng cho đồ thị vô hướng.

```python
def BuildGraph(fname):
    f = open(fname)
    n = int(f.readline())
    V = [i for i in range(n)]
    A = [[0 for i in range(n)] for j in range(n)] # thiết lập ma trận n x n số 0
    for line in f:
        edge = [int(ch) for ch in line.split()]
        i,j = edge[0],edge[1]  # Đây là cạnh nối i → j
        A[i][j] = 1
        A[j][i] = 1
    f.close()
    return V,A
```

Chương trình 3.2a. Thiết lập đồ thị biểu diễn bởi danh sách kề từ tệp dữ liệu danh sách các cạnh, áp dụng cho đồ thị vô hướng.

```python
def BuildGraph(fname):
    f = open(fname)
    n = int(f.readline())
    V = [i for i in range(n)]
    Adj = [[] for i in range(n)] # thiết lập Adj gồm n dãy rỗng
    for line in f:
        edge = [int(ch) for ch in line.split()]
        i,j = edge[0],edge[1] # Đây là cạnh nối i → j
        Adj[i].append(j)   # Bổ sung j vào hàng thứ i của Adj
        Adj[j].append(i)   # Bổ sung i vào hàng thứ j của Adj
    f.close()
    return V,Adj
```

b) Trường hợp đồ thị có hướng

Chương trình 3.1b. Thiết lập đồ thị biểu diễn bởi ma trận kề từ tệp dữ liệu danh sách các cạnh, áp dụng cho đồ thị có hướng.

```python
def BuildGraph(fname):
    f = open(fname)
    n = int(f.readline())
    V = [i for i in range(n)]
    A = [[0 for i in range(n)] for j in range(n)] # thiết lập ma trận n x n số 0
    for line in f:
        edge = [int(ch) for ch in line.split()]
        i,j = edge[0],edge[1] # Đây là cạnh nối i → j
        A[i][j] = 1
    f.close()
    return V,A
```

Chương trình 3.2b. Thiết lập đồ thị biểu diễn bởi danh sách kề từ tệp dữ liệu danh sách các cạnh, áp dụng cho đồ thị có hướng.

```
1  def BuildGraph(fname):
2      f = open(fname)
3      n = int(f.readline())
4      V = [i for i in range(n)]
5      Adj = [[] for i in range(n)] # thiết lập Adj gồm n dãy rỗng
6      for line in f:
7          edge = [int(ch) for ch in line.split()]
8          i,j = edge[0],edge[1] # Đây là cạnh nối i → j
9          Adj[i].append(j)      # Bổ sung j vào hàng thứ i của Adj
10     f.close()
11     return V,Adj
```

Có thể thiết lập và biểu diễn đồ thị thông qua tệp dữ liệu danh sách các cạnh đồ thị. Các chương trình này được áp dụng riêng biệt cho hai trường hợp đồ thị vô hướng và đồ thị có hướng.

1. Một đơn đồ thị, vô hướng có n đỉnh, có thể có số cạnh lớn nhất là bao nhiêu?

2. Khi nào thì tất cả các phần tử của Adj đều rỗng?

LUYỆN TẬP

1. Bổ sung thêm đoạn chương trình kiểm tra khi đọc dữ liệu danh sách các cạnh đồ thị của Hoạt động 3 như sau: Với mỗi dòng dữ liệu, nếu hai chỉ số i = j thì bỏ qua dòng này.

2. Từ ma trận kề A của đồ thị G có thể tính được số các cạnh của đồ thị không? Nếu được thì tính bằng cách nào?

VẬN DỤNG

1. Cho ma trận kề A của đồ thị vô hướng G. Viết hàm `GraphEdge(A)` trả lại danh sách E các cạnh của đồ thị G.

2. Cho danh sách kề Adj của đồ thị G. Viết hàm `GraphEdge(Adj)` trả lại danh sách E các cạnh của đồ thị G. Viết chương trình cho hai trường hợp riêng biệt, G là đồ thị vô hướng và G là đồ thị có hướng.

BÀI 13 THỰC HÀNH THIẾT LẬP ĐỒ THỊ

Sau bài học này em sẽ:

- Thực hành thiết lập đồ thị theo các bộ dữ liệu khác nhau.

Em hãy trao đổi, thảo luận và trả lời một số câu hỏi sau:

– Nếu đồ thị là vô hướng thì ma trận kề có đặc điểm gì?

– Phân biệt sự giống nhau và khác nhau giữa ma trận kề và danh sách kề?

– Khái niệm bậc của các đỉnh có gì khác nhau trong trường hợp đồ thị là vô hướng, có hướng?

Nhiệm vụ 1: Viết chương trình hiển thị danh sách kề

Đơn đồ thị vô hướng G = (V, E) được cho bởi ma trận kề A. Ma trận A được cho trong tệp văn bản có dạng như hình bên.

Yêu cầu xác định danh sách kề của đồ thị trên, kết quả thể hiện ra màn hình, ví dụ như sau:

```
Đỉnh kề với 0: 1 2
Đỉnh kề với 1: 0 2 3
Đỉnh kề với 2: 0 1 3
Đỉnh kề với 3: 1 2
```

Data.inp

```
4
0 1 1 0
1 0 1 1
1 1 0 1
0 1 1 0
```

Hình 13.1. Tệp ma trận kề

Hướng dẫn

Dữ liệu biểu diễn của đồ thị là ma trận kề trong tệp Data.inp. Trong bài trước, chúng ta đã thiết lập hàm `BuildGraph(fname)` lấy dữ liệu từ tệp ma trận kề và trả về cặp dữ liệu V, A là danh sách đỉnh và ma trận kề của đồ thị. Hàm sau thể hiện danh sách kề trên màn hình theo đúng yêu cầu trên với tham số đầu vào là ma trận kề A.

```
1 def In_danh_sach_dinh_ke(A):
2     n = len(A)
3     for i in range(n):
4         print("Đỉnh kề với",i,":", end = " ")
5         for j in range(n):
6             if A[i][j] == 1:
7                 print(j, end = " ")
8         print()
```

Phần chương trình chính của lời giải bài toán như sau:

```
1 fi = "Data.inp"
2 V,A = BuildGraph(fi)
3 In_danh_sach_dinh_ke(A)
```

Nhiệm vụ 2: Viết chương trình hiển thị ma trận kề, danh sách kề và bậc của đồ thị

Đơn đồ thị vô hướng G = (V,E) được cho bởi danh sách các cạnh. Danh sách các cạnh được cho trong tệp văn bản, trong đó dòng đầu tiên là số các đỉnh của đồ thị, các dòng tiếp theo mỗi dòng mô tả một cạnh của đồ thị.

Yêu cầu tính ma trận kề, danh sách kề và bậc của tất cả các đỉnh của đồ thị G.

Kết quả đưa ra màn hình được thể hiện như sau:

Edges.inp
4
0 1
1 3
0 2
2 3
1 2

Hình 13.2. Tệp danh sách các cạnh

Ma trận kề	Danh sách kề	Bậc của các đỉnh của đồ thị
0 1 1 0	0 1 2	Đỉnh 0: 2
1 0 1 1	1 0 3 2	Đỉnh 1: 3
1 1 0 1	2 0 3 1	Đỉnh 2: 3
0 1 1 0	3 1 2	Đỉnh 3: 2

Hướng dẫn

Trong nhiệm vụ này, dữ liệu đầu vào là tệp danh sách các cạnh của đồ thị vô hướng. Trong bài trước, chúng ta đã thiết lập hàm `BuildGraph(fname)` lấy dữ liệu từ tệp danh sách các cạnh và trả về cặp dữ liệu V, Adj là danh sách các đỉnh và danh sách kề của đồ thị.

– Nhiệm vụ yêu cầu tính và đưa ra màn hình ma trận kề của đồ thị. Chúng ta thiết lập hàm `AdjacencyMatrix(Adj)`, đầu vào là danh sách kề Adj, đầu ra là ma trận kề A của đồ thị như sau:

```
1  def AdjacencyMatrix(Adj):
2      n = len(Adj)
3      A = [[0 for i in range(n)] for j in range(n)]
4      for i in range(n):
5          for k in range(len(Adj[i])):
6              j = Adj[i][k]
7              A[i][j] = 1
8      return A
```

– Chúng ta thiết lập hàm `show(A,op)` đưa ra màn hình ma trận kề và danh sách kề. Hàm này có hai lựa chọn cho tham số op. Nếu op = 0 thì dùng cho trường hợp A là ma trận kề, hàm sẽ in ra màn hình ma trận kề. Nếu op = 1 thì sử dụng đầu vào là danh sách kề và in ra danh sách kề.

```
1  def show(A,op):
2      n = len(A)
3      for i in range(n):
4          if op == 1:
5              print(i,end = " ")
6          for j in range(len(A[i])):
7              print(A[i][j], end = " ")
8          print()
```

– Hàm show_deg(Adj) sau đây tính và in ra lần lượt bậc các đỉnh của đồ thị. Đầu vào của hàm này là danh sách kề Adj.

```
def show_deg(Adj):
    n = len(Adj)
    for i in range(n):
        print("Đỉnh",i,":",len(Adj[i]))
```

Phần chương trình chính của bài toán như sau:

```
fi = "Edges.inp"
V,Adj = BuildGraph(fi)
A = AdjacencyMatrix(Adj)
print("Ma trận kề")
show(A,0)
print("Danh sách kề")
show(Adj,1)
print("Bậc của các đỉnh của đồ thị")
show_deg(Adj)
```



LUYỆN TẬP

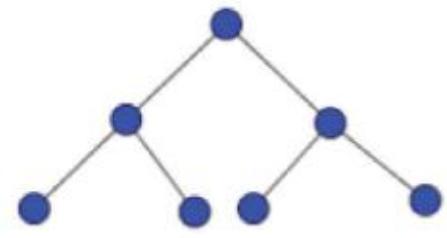
1. Trong Nhiệm vụ 1, hàm In_danh_sach_dinh_ke() lấy thông tin từ ma trận kề A. Có thể sử dụng hàm này với dữ liệu là danh sách kề Adj được không? Nếu có thì viết lại hàm này với dữ liệu đầu vào là danh sách kề Adj.

2. Trong Nhiệm vụ 2, chúng ta có thể thấy các đỉnh kề không được in ra theo thứ tự tăng dần của chỉ số trong biểu diễn danh sách kề. Em hãy giải thích tại sao. Có thể chỉnh sửa chương trình để in ra các đỉnh kề theo thứ tự chỉ số tăng dần được không?



VẬN DỤNG

1. Cây nhị phân có thể được coi là đồ thị vô hướng, các nút của cây sẽ tương ứng là đỉnh, còn quan hệ cha-con là cạnh nối của đồ thị. Với cây nhị phân hoàn chỉnh, các đỉnh được đánh số theo chỉ số của mảng biểu diễn tương ứng của cây. Hãy tính ma trận kề của đồ thị tương ứng cây nhị phân ở hình bên.



2. Viết lại các hàm thiết lập đồ thị BuildGraph(fname) với tệp dữ liệu đầu vào là danh sách các cạnh của đồ thị. Đầu ra của hàm là dãy các giá trị V, E, A, Adj. Viết hàm cho cả hai trường hợp đồ thị vô hướng và đồ thị có hướng.

BÀI 14 KĨ THUẬT DUYỆT ĐỒ THỊ THEO CHIỀU SÂU

Sau bài học này em sẽ:

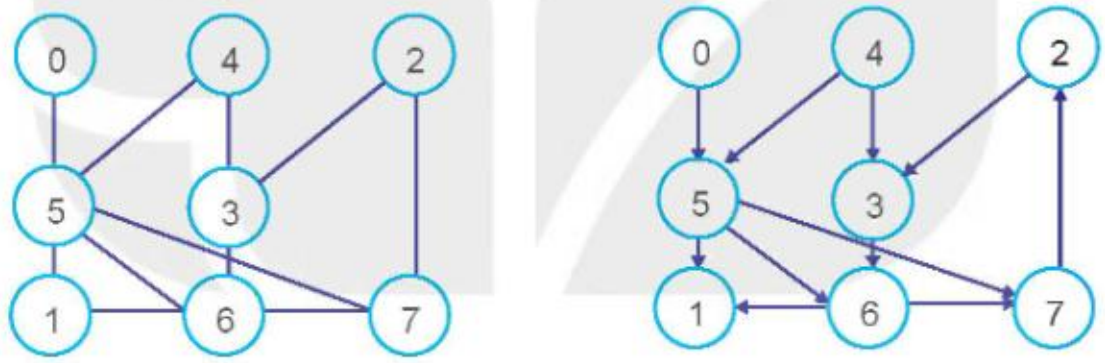
- Trình bày được ý tưởng của duyệt đồ thị theo chiều sâu.
- Mô phỏng được thuật toán duyệt theo chiều sâu.

Chúng ta đã biết bài toán và thuật toán duyệt (tìm kiếm) dữ liệu trên các cấu trúc dữ liệu khác nhau:

– Kiểu dữ liệu mảng (một hoặc hai chiều).

– Kiểu dữ liệu cây (cây nhị phân và cây tìm kiếm nhị phân).

Vậy với dữ liệu của đồ thị, việc duyệt các đỉnh của đồ thị sẽ được thực hiện như thế nào? Quan sát hai đồ thị ở Hình 14.1 và thảo luận với bạn cách thực hiện duyệt trên các đỉnh của đồ thị đó.



Hình 14.1. Duyệt đồ thị

1. Bài toán duyệt đồ thị

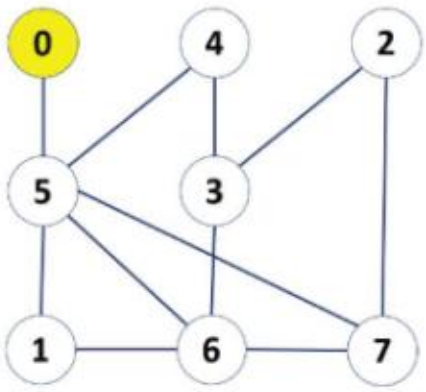
Bài toán duyệt đồ thị là cần duyệt (đánh dấu) tất cả các đỉnh bằng cách đi theo các cạnh của đồ thị. Có hai cách (thuật toán) duyệt đồ thị là duyệt theo chiều sâu và duyệt theo chiều rộng. Chúng ta sẽ lần lượt tìm hiểu các thuật toán này.

Hoạt động 1 Tìm hiểu ý tưởng của thuật toán duyệt đồ thị theo chiều sâu

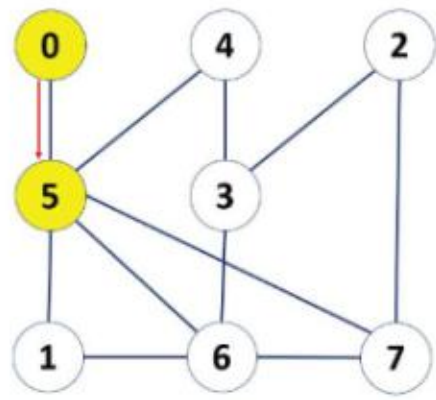
Tìm hiểu ý tưởng của thuật toán duyệt đồ thị theo chiều sâu (DFS – Depth First Search).

Ý tưởng của cách duyệt này, như tên của thuật toán, là mỗi khi xuất phát từ một đỉnh chưa được duyệt, cần đi dọc theo các cạnh theo hướng "sâu" nhất có thể, tức là luôn cố gắng duyệt theo hướng "ra xa" khỏi đỉnh ban đầu, đi tới đỉnh nào thì đánh dấu (duyệt) đỉnh đó, duyệt cho đến khi không đi được nữa thì quay lại đỉnh trước để tìm cách đi khác, cứ như vậy cho đến khi không tìm được đường đi nào nữa thì dừng lại. Quá trình như vậy lặp lại cho đến khi tất cả các đỉnh của đồ thị đã được đánh dấu.

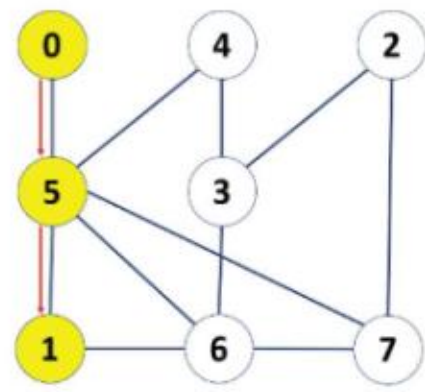
Chúng ta tìm hiểu thuật toán duyệt theo chiều sâu qua ví dụ đồ thị vô hướng trong Hình 14.1a. Giả sử bắt đầu duyệt từ đỉnh 0. Mũi tên màu đỏ chỉ hướng đi theo cạnh.



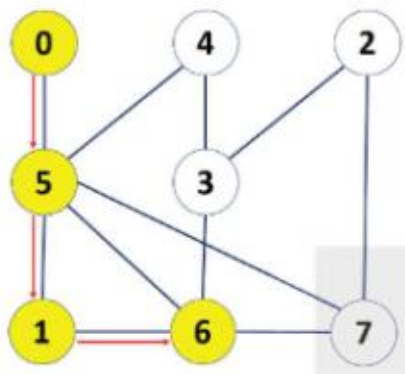
1. Bắt đầu từ đỉnh 0. Duyệt (đánh dấu) đỉnh **0**.



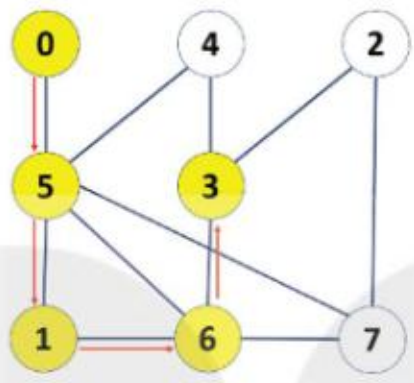
2. Danh sách kề của 0 là [5]. Tìm được 5 chưa duyệt. Duyệt đỉnh **5**.



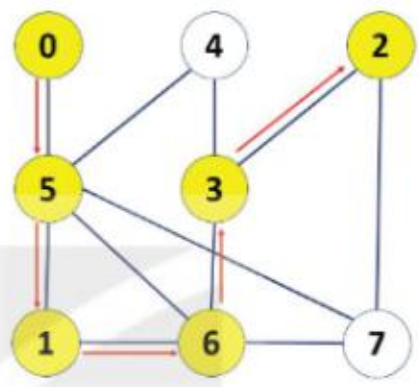
3. Danh sách kề của 5 là [0,1,4,6,7]. Tìm được 1 chưa duyệt. Duyệt đỉnh **1**.



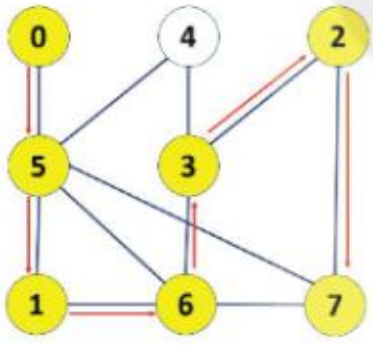
4. Danh sách kề của 1 là [5,6]. Tìm được 6 chưa duyệt. Duyệt đỉnh **6**.



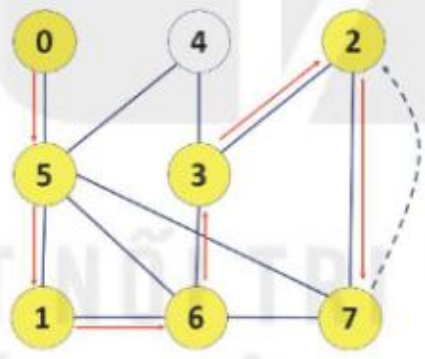
5. Danh sách kề của 6 là [1,3,5,7]. Tìm được 3 chưa duyệt. Duyệt đỉnh **3**.



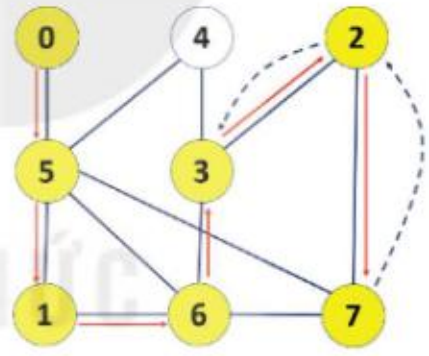
6. Danh sách kề của 3 là [2,4,6]. Tìm được 2 chưa duyệt. Duyệt đỉnh **2**.



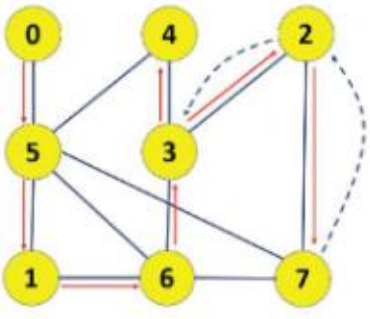
7. Danh sách kề của 2 là [3,7]. Tìm thấy 7 chưa duyệt. Duyệt đỉnh **7**.



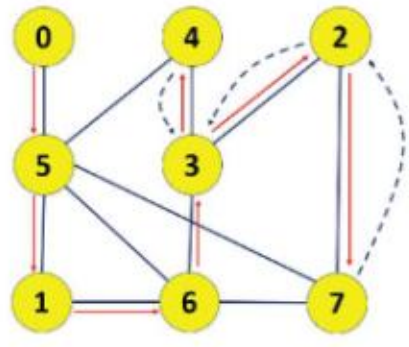
8. Danh sách kề của 7 là [2,5,6]. Không tìm thấy. Quay lại đỉnh 2.



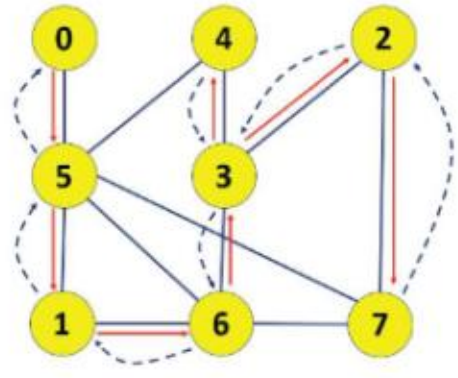
9. Danh sách kề của 2 là [3,7]. Không tìm thấy. Quay lại đỉnh 3.



10. Danh sách kề của 3 là [2,4,6]. Tìm thấy 4 chưa duyệt. Duyệt đỉnh **4**.



11. Danh sách kề của 4 là [3,5]. Không tìm thấy. Quay lại đỉnh 3.



12. Từ đỉnh 3, không tìm thấy, lần lượt quay lại 6, quay lại 1, quay lại 5, quay lại 0.

Hình 14.2. Chi tiết các bước duyệt theo chiều sâu, bắt đầu từ đỉnh 0

Chi tiết các bước của duyệt đồ thị trên được mô tả trong Bảng 14.1.

Bảng 14.1. Các bước duyệt đồ thị

STT	Đỉnh hiện thời	Danh sách đỉnh kề	Duyệt (đánh dấu)	Tìm đỉnh chưa duyệt trong Adj	Trạng thái
1	0	**5**	**0**	**Tìm thấy 5**	Duyệt tiếp
2	5	0 **1** 4 6 7	**5**	**Tìm thấy 1**	Duyệt tiếp
3	1	5 **6**	**1**	**Tìm thấy 6**	Duyệt tiếp
4	6	1 **3** 5 7	**6**	**Tìm thấy 3**	Duyệt tiếp
5	3	**2** 4 6	**3**	**Tìm thấy 2**	Duyệt tiếp
6	2	3 **7**	**2**	**Tìm thấy 7**	Duyệt tiếp
7	7	2 5 6	**7**	Không thấy	Quay lại 2
8	2	3 7		Không thấy	Quay lại 3
9	3	2 **4** 6		Tìm thấy 4	Duyệt tiếp
10	4	3 5 7	**4**	Không thấy	Quay lại 3
11	3	2 4 6		Không thấy	Quay lại 6
12	6	1 3 5 7		Không thấy	Quay lại 1
13	1	5 6		Không thấy	Quay lại 5
14	5	0 1 4 6 7		Không thấy	Quay lại 0
15	0	5			

Lưu ý: Màu đỏ là các đỉnh đã được duyệt (đánh dấu) trước đó. In đậm là đỉnh đầu tiên trong danh sách kề chưa được duyệt, đỉnh này được đánh dấu "tìm thấy" để thực hiện bước đi tiếp theo. Nếu tất cả các đỉnh trong danh sách đỉnh kề đều đã duyệt (màu đỏ) thì thực hiện quay lại đỉnh của bước trước đó.

Như vậy, thứ tự các đỉnh được duyệt là: 0 5 1 6 3 2 7 4.

Thuật toán duyệt như trên được gọi là duyệt theo chiều sâu (DFS – Depth First Search). Thuật toán duyệt này áp dụng cho cả đồ thị vô hướng và có hướng.

Duyệt đồ thị theo chiều sâu (DFS) bắt đầu từ việc "duyệt theo chiều sâu từ đỉnh u" chưa được duyệt: duyệt đỉnh u, sau đó lần lượt xét các đỉnh kề v của đỉnh u, nếu có đỉnh v chưa được duyệt thì "duyệt theo chiều sâu từ đỉnh v", ngược lại quay lui về bước duyệt trước đó. Lặp lại cách duyệt này cho đến khi tất cả các đỉnh của đồ thị đã được duyệt.



1. Thứ tự các đỉnh trong danh sách kề có ảnh hưởng đến thứ tự các đỉnh được duyệt của thuật toán DFS không?
2. Mô tả quá trình duyệt theo chiều sâu của đồ thị có hướng trong Hình 14.1b nếu xuất phát từ đỉnh 4.

2. Thuật toán duyệt theo chiều sâu DFS

Hoạt động 2 **Tìm hiểu thuật toán duyệt DFS**

Quan sát, thảo luận và tìm hiểu thuật toán duyệt theo chiều sâu trên đồ thị bất kì.

Cho trước đồ thị G = (V, E) vô hướng hoặc có hướng. Chúng ta sẽ thiết kế thuật toán duyệt đồ thị theo ý tưởng thực hiện trong Hoạt động 1. Ban đầu, tất cả các đỉnh của đồ thị là chưa đánh dấu. Thuật toán `DFS(Adj,u)` thực hiện công việc duyệt đồ thị theo chiều sâu bắt đầu từ đỉnh u chưa đánh dấu.

Đoạn mã giả sau thực hiện các công việc: thiết lập tất cả các đỉnh của đồ thị là chưa đánh dấu, sau đó bắt đầu duyệt đồ thị G theo chiều sâu có sử dụng thuật toán `DFS(Adj,u)` để bắt đầu duyệt từ đỉnh u.

```
DFS_Traversal(G):
1  Thiết lập tất cả các đỉnh u thuộc V là chưa đánh dấu.
2  Với mỗi đỉnh u thuộc V
3      Nếu u chưa đánh dấu
4          DFS(Adj,u)
```

Hàm đệ quy `DFS(Adj,u)` thực hiện thuật toán duyệt theo chiều sâu bắt đầu từ đỉnh u chưa đánh dấu như sau:

```
DFS(Adj,u):
1  Đánh dấu đỉnh u
2  Với mỗi đỉnh v là đỉnh kề của đỉnh u
3      Nếu đỉnh v chưa đánh dấu
4          DFS(Adj,v)
```

Giả sử đồ thị G = (V, E) được biểu diễn bằng danh sách kề Adj. Mảng mark[] dùng để đánh dấu các đỉnh: mark[v] = False nghĩa là đỉnh v chưa được đánh dấu, mark[v] = True nghĩa là đỉnh v đã được đánh dấu. Ban đầu, mark[v] = False với mọi đỉnh v.

Hàm đệ quy `DFS(Adj,u)` trong Python dùng để duyệt đồ thị theo chiều sâu bắt đầu từ đỉnh u chưa đánh dấu như sau:

```
1  def DFS(Adj,u):
2      mark[u] = True # Đánh dấu đỉnh u
3      for v in Adj[u]:
4          if not mark[v]:  # Đỉnh v chưa đánh dấu
5              DFS(Adj,v)
```

Giải thích: Dòng lệnh 2 cho biết đỉnh u đã được đánh dấu. Dòng lệnh 3 kiểm tra các đỉnh kề v của đỉnh u. Nếu có đỉnh v chưa được đánh dấu thì gọi đệ quy cho đỉnh này. Đây chính là ý tưởng của duyệt đồ thị theo chiều sâu đã mô tả trong Hoạt động 1. Khi kết thúc thực hiện hàm `DFS(Adj,u)` thì tất cả các đỉnh mà có đường đi từ đỉnh u đều được đánh dấu.

Hàm `DFS_Traversal(V,Adj)` sẽ duyệt đồ thị theo chiều sâu với bộ dữ liệu (V, Adj) như sau:

```
1 def DFS_Traversal(V, Adj): # Duyệt đồ thị (V, Adj) theo chiều sâu
2   mark = [False] * len(V)   # Tất cả các đỉnh là chưa được đánh dấu
3   for u in V:  # Duyệt các đỉnh của đồ thị
4     if not mark[u]:        # Đỉnh u chưa đánh dấu
5       DFS(Adj, u)          # Duyệt theo chiều sâu bắt đầu từ u
```

Chương trình sau đây sẽ tạo đồ thị từ tệp danh sách kề, sau đó duyệt đồ thị này theo chiều sâu dùng hàm `DFS_Traversal()`:

```
1 fname = "graph.inp"
2 V, Adj = BuildGraph(fname) # Tạo đồ thị từ tệp danh sách kề
3 DFS_Traversal(V,Adj) # Duyệt đồ thị theo chiều sâu
```

Phân tích thời gian của DFS:

– Theo cách thiết lập tại dòng 2 thì mỗi đỉnh của đồ thị G chỉ có thể duyệt đúng một lần, do đó độ phức tạp thời gian duyệt các đỉnh của đồ thị sẽ là O(|V|). Chúng ta sẽ kí hiệu đơn giản O(V).

– Thời gian tìm đỉnh cần đi tiếp trong danh sách kề tại dòng 3, 4 sẽ tính như sau: Vì tổng tất cả các việc tìm đỉnh kề tại dòng 3 sẽ bằng chính số các cạnh của đồ thị G, tức là bằng |E|. Với đồ thị có hướng thì mỗi cạnh được duyệt một lần trong lệnh 3, còn nếu G là đồ thị vô hướng thì số lần duyệt các đỉnh kề sẽ bằng 2|E| vì mỗi cạnh sẽ được tìm duyệt hai lần.

Từ đó suy ra kết quả sau: Độ phức tạp thời gian duyệt theo chiều sâu là O(V+E) nếu đồ thị có hướng và là O(V+2E) nếu đồ thị vô hướng.

Thuật toán duyệt đồ thị theo chiều sâu DFS có thể mô tả bằng hàm đệ quy DFS thực hiện duyệt theo chiều sâu từ đỉnh s.



1. Nếu đồ thị G chỉ bao gồm các đỉnh biệt lập, không có cạnh nào thì thuật toán duyệt sâu DFS sẽ được thực hiện như thế nào?

2. Chỉnh sửa hàm `DFS()` bổ sung lệnh in thông tin của các đỉnh khi duyệt. Ví dụ hàm `DFS()` có thể viết lại như sau:

```
1  def DFS(Adj,u):
2      mark[u] = True # Đánh dấu đỉnh u
3      print(u) # In đỉnh u
4      for v in Adj[u]:
5          if not mark[v]:
6              DFS(Adj,v)
```

Sử dụng hàm trên áp dụng duyệt các phần tử của đồ thị Hình 14.1a trong phần khởi động. Kiểm tra thứ tự các đỉnh đã duyệt có trùng khớp với thứ tự các đỉnh đã duyệt (bằng tay) trong Hoạt động 1 hay không.

Hoạt động 3 Tìm hiểu thuật toán duyệt không đệ quy DFS

Tìm hiểu một cách cài đặt khác của thuật toán duyệt theo chiều sâu DFS không sử dụng kĩ thuật đệ quy.

Thuật toán không đệ quy DFS sử dụng ngăn xếp để duyệt theo chiều sâu, bắt đầu từ đỉnh u như sau:

```
def DFS(Adj,u):
    S = Stack()  # khởi tạo Stack rỗng
    push(S,u)
    while not isEmptyStack(S):
        v = pop(S)
        if not mark[v]:
            mark[v] = True # Đánh dấu đỉnh v
            for w in Adj[v]:
                push(S,w)
```

Chương trình sau sẽ đọc dữ liệu từ tệp đầu vào và thực hiện lệnh duyệt theo chiều sâu bắt đầu từ đỉnh 0:

```
from Stack import *
fname = "graph.inp"
V,Adj = BuildGraph(fname)
mark = [False]*len(V)
DFS(Adj,0)
```

Với dữ liệu của danh sách kề Adj ở Bảng 14.1, khi thực hiện hàm duyệt `DFS(Adj,0)`, kết quả thứ tự duyệt các đỉnh của đồ thị là: 0 5 7 6 3 4 2 1.

Có thể thiết lập hàm duyệt theo chiều sâu áp dụng tổng quát đồ thị G với bộ dữ liệu (V, Adj) như sau:

```
def DFS_Traversal(V,Adj):
    mark = [False]*len(V)
    for u in V:
        if not mark[u]:
            DFS(Adj,u)
```

Chương trình đọc dữ liệu từ tệp đầu vào và duyệt đồ thị theo chiều sâu như sau:

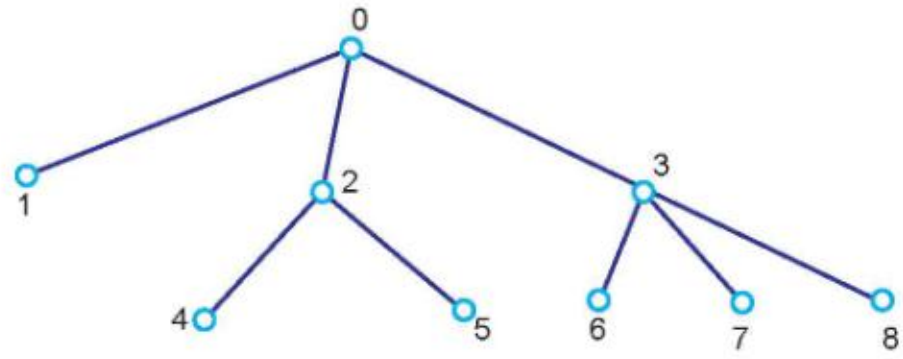
```
from Stack import *
fname = "graph.inp"
V,Adj = BuildGraph(fname)
mark = [False] * len(V)
DFS_Traversal(V,Adj)
```

Thuật toán duyệt không đệ quy theo chiều sâu sử dụng ngăn xếp tương tự thuật toán đệ quy duyệt theo chiều sâu. Điểm khác biệt là tại mỗi bước duyệt, thuật toán tìm đỉnh chưa duyệt theo thứ tự ngược lại của danh sách kề Adj.

KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

1. Nếu áp dụng thuật toán duyệt DFS cho đồ thị có hướng Hình 14.1b trong phần khởi động thì thứ tự các đỉnh được đánh dấu sẽ theo thứ tự nào?

2. Với đồ thị Hình 14.3 thì thứ tự các đỉnh đã duyệt theo chiều sâu, bắt đầu từ đỉnh 0 sẽ như thế nào? (theo cả hai cách đệ quy và không đệ quy).



Hình 14.3. Đồ thị dạng cây

LUYỆN TẬP

1. Sửa chương trình của thuật toán DFS không đệ quy sao cho chương trình trong khi duyệt sẽ in ra các bước với thông tin sau:

   – Thông tin ngăn xếp (hiện thời).

   – Phần tử được lấy ra từ ngăn xếp.

   – Phần tử được đánh dấu để chuẩn bị cho bước sau (mark).

2. Hàm `DFS(Adj,u)` có thể viết theo một cách khác, việc kiểm tra đỉnh u đã đánh dấu chưa được đưa vào bên trong hàm như sau:

```
1  def DFS(Adj,u):
2      if not mark[u]:
3          mark[u] = True # Đánh dấu đỉnh s
4          for v in Adj[u]:
5              DFS(Adj,v)
```

a) Trong trường hợp này, phần chương trình chính cần viết lại như thế nào?

b) Viết lại toàn bộ chương trình duyệt đồ thị theo chiều sâu theo hàm mô tả trên. Cách duyệt này có tương đương với cách đã thực hiện trong Hoạt động 2 không?

VẬN DỤNG

1. Viết chương trình in ra thứ tự các đỉnh đã được duyệt bằng thuật toán DFS theo cả hai cách đệ quy và không đệ quy, áp dụng cho đồ thị có hướng Hình 14.1b trong phần Khởi động.

2. Các thuật toán DFS đã mô tả trong các phần trên đều thực hiện trên đồ thị được biểu diễn bằng danh sách kề Adj. Hãy viết lại hàm `DFS()` được thực hiện trên đồ thị được biểu diễn bằng ma trận kề A.

3. Cho đồ thị G = (V, E) và hai đỉnh s, t bất kì. Chứng minh tính chất: Tồn tại đường đi từ s đến t khi và chỉ khi quá trình duyệt theo chiều sâu từ đỉnh s sẽ duyệt qua đỉnh t, hay nói cách khác, quá trình duyệt theo chiều sâu từ đỉnh s sẽ đi qua tất cả các đỉnh nằm trong thành phần liên thông chứa đỉnh s, trong đó có đỉnh t.

BÀI 15 THỰC HÀNH DUYỆT ĐỒ THỊ THEO CHIỀU SÂU

Sau bài học này em sẽ:

- Thực hành duyệt đồ thị theo chiều sâu.

Trong lí thuyết đồ thị, chu trình được định nghĩa là một đường đi không tầm thường khép kín, tức là đường đi có số cạnh lớn hơn 1 và đỉnh xuất phát trùng với đỉnh kết thúc. Làm cách nào để kiểm tra một đồ thị cho trước có chu trình hay không?

Nhiệm vụ: Hệ thống chuyên đề học tập

Trường em từ năm học này sẽ tổ chức mở rất nhiều chuyên đề học tập cho học sinh lựa chọn, các chuyên đề sẽ được học trong các thời gian khác nhau. Các chuyên đề được đánh số từ 0 đến n – 1 với n là số tự nhiên. Tuy nhiên giữa các chuyên đề này có quan hệ ràng buộc kiến thức, ví dụ quan hệ (i, j) chỉ ra rằng muốn học chuyên đề i thì cần học trước chuyên đề j.

Dữ liệu đầu vào dưới dạng tệp văn bản Data.inp như sau:

Data.inp

```
5
1 2
2 4
4 1
1 0
```

– Dòng đầu tiên là số tự nhiên n (số các chuyên đề học tập của trường em).

– Các dòng tiếp theo mô tả quan hệ dạng (i, j), mỗi dòng là hai số i và j cách nhau bởi dấu cách.

Hệ thống các chuyên đề của nhà trường được gọi là hợp lí nếu về nguyên tắc mỗi học sinh đều có thể đăng kí để học tất cả các chuyên đề.

Viết chương trình nhập bộ dữ liệu trên, kiểm tra và thông báo hệ thống chuyên đề có hợp lí hay không.

Với bộ dữ liệu trên thì hệ thống không hợp lí vì nếu em muốn học chuyên đề 1, em phải học chuyên đề 2 trước (dòng 2); muốn học chuyên đề 2 thì cần học chuyên đề 4 trước (dòng 3). Nhưng muốn học chuyên đề 4 thì cần học chuyên đề 1 (dòng 4), điều này mâu thuẫn. Vậy hệ thống chuyên đề trên không hợp lí.

Hướng dẫn

Nếu gọi mỗi chuyên đề là một đỉnh được đánh số từ 0 đến n – 1 và quan hệ ràng buộc kiến thức (i, j) như một cạnh có hướng từ đỉnh i đến đỉnh j thì tập hợp các chuyên đề học tập của trường em sẽ trở thành một mô hình đồ thị có hướng. Trong đồ thị này dễ thấy đồ thị sẽ không có chu trình tương đương với tính hợp lí của hệ thống các chuyên đề.

Vậy với bài toán trên chúng ta cần kiểm tra xem đồ thị các chuyên đề có chu trình hay không.

Ý tưởng của việc kiểm tra này sẽ được thực hiện bằng cách duyệt theo chiều sâu của đồ thị, bắt đầu từ một đỉnh bất kì. Để thực hiện được việc này chúng ta sẽ đưa vào mảng tổng thể các trạng thái status[] của đồ thị có ý nghĩa như sau:

status[v] = 0 nếu đỉnh v chưa được xét (hoặc duyệt).

status[v] = 1 chỉ ra đỉnh này đang trong quá trình duyệt.

status[v] = 2 nếu đỉnh này đã được duyệt xong.

Công việc kiểm tra chu trình được thực hiện thông qua hai bước sau:

– Hàm `DFS_acyclic(Adj,s)` kiểm tra trong quá trình duyệt bắt đầu từ đỉnh s có gặp chu trình hay không. Việc kiểm tra này khá đơn giản, bắt đầu duyệt từ đỉnh s có trạng thái bằng 0, nếu trong quá trình duyệt gặp đỉnh có trạng thái 1, tức là đã có chu trình.

– Tiến hành thực hiện kiểm tra trên toàn bộ các đỉnh của đồ thị.

Hàm `DFS_acyclic(Adj,s)` sẽ trả lại True nếu vùng duyệt từ s không có chu trình, ngược lại nếu có chu trình sẽ trả về False.

```
1  def DFS_acyclic(Adj,s):
2      status[s] = 1 # Đang duyệt
3      for v in Adj[s]:
4          if status[v] == 1:
5              return False
6          elif status[v] == 0:
7              if not DFS_acyclic(Adj,v):
8                  return False
9      status[s] = 2 # Đã duyệt xong
10     return True
```

Hàm `Acyclic(V,Adj)` sẽ kiểm tra trên toàn bộ đồ thị và trả về True nếu đồ thị không có chu trình, ngược lại trả về False.

```
1  def Acyclic(V,Adj):
2      for u in V:
```

```
            if status[u] == 0:
                if not DFS_acyclic(Adj,u):
                    return False
        return True
```

Bộ dữ liệu đầu vào trên chính là tệp danh sách các cạnh của đồ thị có hướng, chúng ta đã biết cách thiết lập hàm `BuildGraph(fname)` đọc dữ liệu này và trả về tập các đỉnh V và danh sách kề Adj trong Bài 12.

Phần chương trình chính của lời giải bài toán sẽ như sau:

```
# Chương trình chính
fi = "Data.inp"
V,Adj = BuildGraph(fi)
status = [0]*len(V)
if Acyclic(V,Adj):
    print("Hệ thống chuyên đề học tập hợp lí.")
else:
    print("Hệ thống chuyên đề học tập không hợp lí.")
```

LUYỆN TẬP

1. Hàm kiểm tra chu trình của đồ thị trên còn đúng không nếu đồ thị ban đầu là vô hướng?

2. Viết lại hàm kiểm tra chu trình `DFS_acyclic(Adj,s)` trong chương trình trên nhưng sử dụng phương án không đệ quy của thuật toán DFS.

VẬN DỤNG

1. Sửa lại chương trình bổ sung thông báo nếu hệ thống chuyên đề không hợp lí thì thông báo dãy các chuyên đề có mâu thuẫn về kiến thức.

2. Mở rộng bài tập trên cho đồ thị có hướng bất kì G = (V, E), được biểu diễn bởi ma trận kề A hoặc danh sách kề Adj. Viết hàm kiểm tra xem đồ thị G có chu trình hay không, nếu có thì hiển thị trên màn hình chu trình đó, bao gồm dãy các đỉnh tham gia vào chu trình.

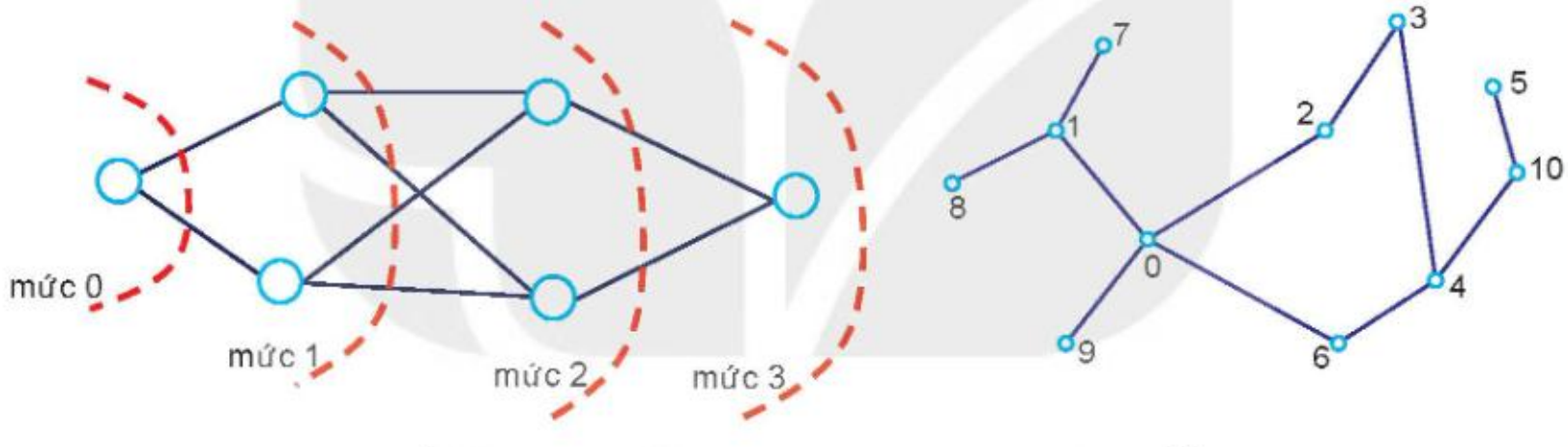
KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

BÀI 16 KĨ THUẬT DUYỆT ĐỒ THỊ THEO CHIỀU RỘNG

Sau bài học này em sẽ:

- Trình bày được ý tưởng của duyệt đồ thị theo chiều rộng.
- Mô phỏng được thuật toán duyệt theo chiều rộng.

Chúng ta đã làm quen với thuật toán duyệt đồ thị theo chiều sâu, quá trình duyệt đi "sâu" nhất có thể theo các cạnh của đồ thị. Ngoài ra còn có cách duyệt đồ thị theo chiều rộng, được hình dung như khi đổ nước xuống một sàn nhà phẳng, nước sẽ lan toả ra xung quanh theo các hình tròn đồng tâm. Cách duyệt theo chiều rộng có thể được mô phỏng như Hình 16.1a.



a) b)

Hình 16.1. Duyệt đồ thị lan toả theo các mức.

Giả sử ta bắt đầu duyệt từ đỉnh 0 của đồ thị Hình 16.1b theo chiều rộng. Theo em, chúng ta sẽ duyệt các đỉnh theo nguyên tắc nào và duyệt theo thứ tự nào?

1. Ý tưởng duyệt đồ thị theo chiều rộng

Hoạt động 1 Làm quen ý tưởng của thuật toán duyệt đồ thị theo chiều rộng

Thực hiện công việc duyệt theo chiều rộng của đồ thị Hình 16.1b, bắt đầu từ đỉnh 0. Các bước thực hiện sẽ duyệt các đỉnh theo trình tự sau:

– Mức 0: Bản thân đỉnh 0.

– Mức 1: Các đỉnh kề với đỉnh mức 0.

– Mức 2: Các đỉnh là kề với đỉnh mức 1. Đỉnh mức 2 là các đỉnh mà tồn tại đường đi từ đỉnh 0 đến đỉnh này theo 2 cạnh, qua đỉnh mức 1.

Quá trình cứ tiếp tục như vậy cho đến khi không thể duyệt thêm được nữa.

Trao đổi, thảo luận nhóm để nhận biết sự khác biệt giữa hai phương pháp duyệt đồ thị theo chiều sâu và chiều rộng khác nhau như thế nào.

Thiết lập bảng sau và điền các thông tin vào các cột. Bảng các đỉnh được duyệt theo chiều rộng, xuất phát từ đỉnh 0.

Bảng 16.1. Duyệt các đỉnh theo mức

Mức	Các đỉnh được duyệt	Ghi chú
0	0	Mức 0 chỉ có đúng phần tử ban đầu 0.
1	1 2 6 9	Các đỉnh kề của 0.
2	3 4 7 8	Các đỉnh kề với các đỉnh mức 1 (nhưng không nằm trong mức 0 và 1), tức là có đường đi qua 2 cạnh xuất phát từ 0.
3	10	Đường đi từ 0: $0 \rightarrow 6 \rightarrow 4 \rightarrow 10$ *Lưu ý:* Có cách đi khác từ 0 đến 10 qua 4 cạnh nhưng chúng ta không chọn (0, 2, 3, 4, 10).
4	5	$0 \rightarrow 6 \rightarrow 4 \rightarrow 10 \rightarrow 5$
5	<không có>	Dừng tìm kiếm

Như vậy nếu duyệt theo cách trên thì thứ tự các đỉnh được duyệt là:

0 1 2 6 9 3 4 7 8 10 5

Theo cách trên, việc duyệt đồ thị theo chiều rộng được bắt đầu từ đỉnh s bất kì. Thực hiện lần lượt theo các đỉnh cùng mức 0, 1, 2,... Đỉnh mức 0 chính là s, mức 1 bao gồm các đỉnh kề với s,..., mức k sẽ bao gồm các đỉnh kề với đỉnh mức k − 1, tức là có tồn tại đường đi k cạnh từ đỉnh s.

Cho trước hai đỉnh s và f. Nếu tồn tại đường đi k cạnh từ s đến f và k là số nhỏ nhất thì ta nói *f có khoảng cách k đến đỉnh s*. Nếu không có đường đi từ s đến f thì ta nói khoảng cách từ s đến f là ∞ (vô cùng).

Lưu ý: Nếu từ đỉnh s có nhiều đường đi đến đỉnh f thì khoảng cách được hiểu là đường đi ngắn nhất (ít cạnh nhất) trong số các đường đi.

Như vậy, ý tưởng của thuật toán duyệt theo chiều rộng có thể như sau:

> Với đỉnh s bất kì, ý tưởng của thuật toán duyệt đồ thị theo chiều rộng xuất phát từ đỉnh s là sẽ lần lượt duyệt các đỉnh theo từng mức hay theo khoảng cách từ đỉnh s đến đỉnh được duyệt.

1. Mệnh đề sau đúng hay sai?

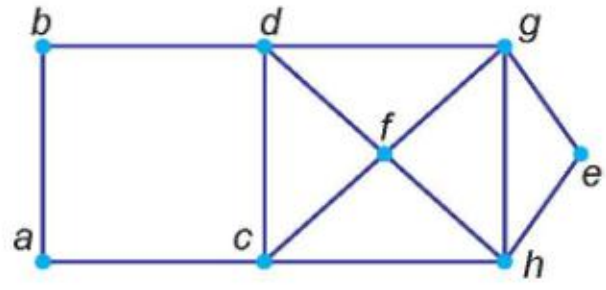
Giả sử gọi `BFS(Adj,s)` là chương trình duyệt đồ thị theo chiều rộng bắt đầu từ đỉnh s. Khi đó với mọi đỉnh v thuộc V, hàm `BFS(Adj,s)` sẽ duyệt qua đỉnh v khi và chỉ khi tồn tại đường đi từ s đến v.

2. Trả lời các câu hỏi dựa trên đồ thị Hình 16.2.

a) Các đỉnh kề với a là đỉnh nào?

b) Khoảng cách từ đỉnh a đến e là bao nhiêu?

c) Nếu thực hiện duyệt đồ thị theo chiều rộng bắt đầu từ đỉnh a thì thứ tự các đỉnh được duyệt có thể như thế nào?



Hình 16.2

2. Thuật toán duyệt đồ thị theo chiều rộng (BFS – Breadth First Search)

Hoạt động 2 **Tìm hiểu thuật toán duyệt đồ thị theo chiều rộng**

Tìm hiểu, thảo luận về cách cài đặt thuật toán duyệt theo chiều rộng.

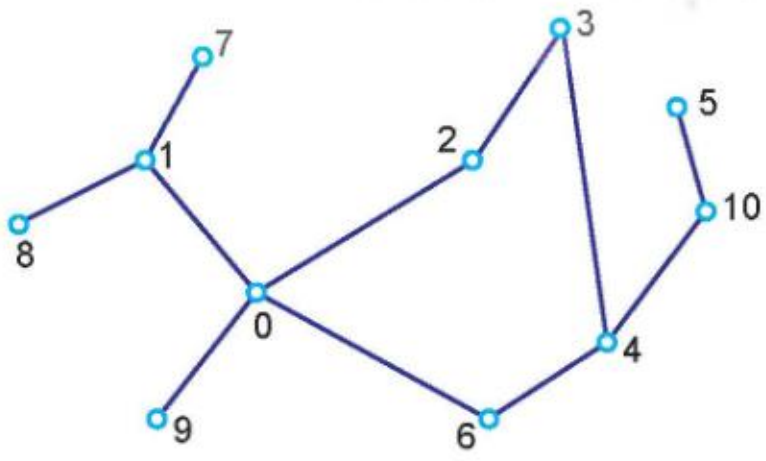
Thuật toán duyệt đồ thị theo chiều rộng được thiết kế gần giống như bản không đệ quy của duyệt đồ thị theo chiều sâu, sự khác biệt chỉ là thay thế công cụ ngăn xếp bằng *hàng đợi*. Thuật toán chính là hàm `BFS(Adj,s)` thiết lập duyệt theo chiều rộng bắt đầu từ đỉnh s. Mảng mark[] dùng để đánh dấu các đỉnh đã duyệt. Ban đầu cần thiết lập mark[v] = False với mọi đỉnh v.

Hàm chính duyệt theo chiều rộng được mô tả như sau:

```
1 def BFS(Adj,s):
2     Q = Queue()
3     enqueue(Q,s)
4     while not isEmptyQueue(Q):
5         v = dequeue(Q)
6         if not mark[v]:
7             mark[v] = True # Đánh dấu đỉnh v
8             for u in Adj[v]:
9                 enqueue(Q,u)
```

Giải thích: Hàm được thiết lập để duyệt bắt đầu từ đỉnh s chưa được đánh dấu. Khi bắt đầu, hàng đợi Q được thiết lập và s được đưa vào Q tại dòng 2 và 3. Thao tác duyệt theo chiều rộng được thực hiện tại vòng lặp 4. Vòng lặp này sẽ thực hiện cho đến khi Q rỗng. Lần lượt lấy v ra khỏi Q, nếu v chưa được đánh dấu thì đánh dấu v và đưa các đỉnh kề của v vào hàng đợi Q. Lập luận tương tự với DFS, ta tính được độ phức tạp thời gian của thuật toán trên là O(V+2E) nếu G là đồ thị vô hướng, là O(V+E) nếu đồ thị G có hướng.

Mô phỏng thủ công thuật toán trên với đồ thị Hình 16.3 có thể như sau.



Hình 16.3. Đồ thị và dữ liệu danh sách kề

graph.inp

```
11
0 1 2 6 9
1 0 7 8
2 0 3
3 2 4
4 3 6 10
5 10
6 0 4
7 1
8 1
9 0
10 4 5
```

Bảng 16.2 sẽ mô tả lại dữ liệu được cập nhật theo từng bước của vòng lặp tại dòng 4. Các thông tin được ghi lại là hàng đợi Q, đỉnh v = `dequeue()` tại dòng 5 và đỉnh được đánh dấu tại dòng 7. Các đỉnh in đậm là mới được bổ sung vào hàng đợi, các đỉnh này là đỉnh kề của đỉnh vừa được đánh dấu ở hàng trên.

Bảng 16.2. Chi tiết các bước duyệt theo BFS

STT	Queue	Dequeue	Mark	Trạng thái
1	0	0	0	Đánh dấu 0
2	**1 2 6 9**	1	1	Đánh dấu 1
3	2, 6, 9, **0, 7, 8**	2	2	Đánh dấu 2
4	6, 9, 0, 7, 8, **0, 3**	6	6	Đánh dấu 6
5	9, 0, 7, 8, 0, 3, **0, 4**	9	9	Đánh dấu 9
6	0, 7, 8, 0, 3, 0, 4, **0**	0	--	
7	7, 8, 0, 3, 0, 4, 0	7	7	Đánh dấu 7
8	8, 0, 3, 0, 4, **0, 1**	8	8	Đánh dấu 8
9	0, 3, 0, 4, 0, 1, **1**	0	--	
10	3, 0, 4, 0, 1, 1	3	3	Đánh dấu 3
11	0, 4, 0, 1, 1, **2**, **4**	0	--	
12	4, 0, 1, 1, **2, 4**	4	4	Đánh dấu 4
13	0, 1, 1, 2, 4, **3, 6, 10**	0	--	
14	1, 1, 2, 4, 3, 6, 10	1	--	
15	1, 2, 4, 3, 6, 10	1	--	
16	2, **4**, 3, 6, 10	2	--	
17	**4**, 3, 6, 10	4	--	
18	3, 6, 10	3	--	
19	6, 10	6	--	
20	10	10	10	Đánh dấu 10
21	**4 5**	4	--	
22	5	5	5	Đánh dấu 5
23	**10**	10	--	
24	<rỗng>			

Thứ tự các đỉnh đã đánh dấu: 0 1 2 6 9 7 8 3 4 10 5.

Chúng ta sẽ thiết lập hàm thực hiện thuật toán duyệt theo chiều rộng trên toàn bộ đồ thị G. Nếu bộ dữ liệu đầu vào của đồ thị là (V, Adj) thì hàm duyệt sẽ có dạng `BFS_Traversal(V,Adj)`.

```
1  def BFS_Traversal(V,Adj):
2      mark = [False]*len(V)
3      for s in V:
4          if not mark[s]:
5              BFS(Adj,s)
```

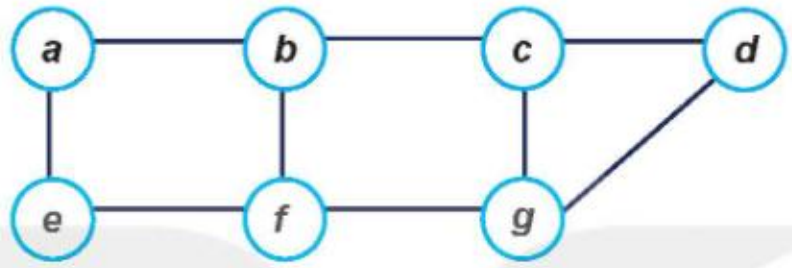
Đoạn chương trình sau mô tả phần thiết lập thông tin ban đầu của đồ thị, sau đó thực hiện thuật toán duyệt theo chiều rộng trên toàn bộ đồ thị G:

```
from Queue import *
fname = "graph.inp"
V,Adj = BuildGraph(fname)
BFS_Traversal(V,Adj)
```

Thuật toán duyệt đồ thị theo chiều rộng bắt đầu tại đỉnh s sẽ lần lượt duyệt các đỉnh có đường đi từ s với số cạnh lần lượt là 0, 1, 2,... Thuật toán BFS có cấu trúc điều khiển tương tự như thuật toán duyệt không đệ quy theo chiều sâu DFS, trong đó sử dụng hàng đợi thay thế cho ngăn xếp.

1. Thứ tự các đỉnh có trong danh sách đỉnh kề Adj có ảnh hưởng đến thứ tự các đỉnh được đánh dấu trong thuật toán duyệt theo chiều rộng hay không?

2. Cho đồ thị Hình 16.4. Nếu thực hiện duyệt theo chiều sâu và chiều rộng bắt đầu từ đỉnh a thì thứ tự các đỉnh được duyệt sẽ như thế nào?



Hình 16.4

LUYỆN TẬP

1. Viết lại hàm `BFS()` in ra các đỉnh đã duyệt. Áp dụng vào đồ thị trong bài để kiểm tra thứ tự các đỉnh đã duyệt có đúng như phần mô phỏng thủ công các hoạt động trên không.

2. Viết lại hàm `BFS()` duyệt theo chiều rộng nhưng sử dụng dữ liệu là ma trận kề A của đồ thị.

VẬN DỤNG

1. Cho đơn đồ thị vô hướng G = (V, E). Sử dụng thuật toán duyệt theo chiều rộng BFS, viết chương trình kiểm tra xem G có chu trình hay không. *Chu trình* (cycle) ở đây được hiểu là một đường đi khép kín, đỉnh xuất phát trùng với đỉnh kết thúc. Cần thiết lập hàm dạng `Acycle(G)`, hàm trả lại True nếu G không có chu trình, ngược lại hàm trả lại False.

2. Cho đơn đồ thị G = (V, E) vô hướng hoặc có hướng. Cho trước hai đỉnh bất kì s và t. Viết chương trình kiểm tra xem có tồn tại đường đi từ s đến t hay không. Nếu có thì chương trình cần chỉ ra dãy các đỉnh tương ứng trên đường đi từ s đến t, nói cách khác chương trình cần chỉ ra một dãy các đỉnh $v_0, v_1,..., v_k$ sao cho:

(v_{j-1}, v_j) là cạnh của đồ thị với j = 1, 2,..., k;

$s = v_0, t = v_k$.

KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

BÀI 17 THỰC HÀNH DUYỆT ĐỒ THỊ TỔNG HỢP

Sau bài học này em sẽ:

- Thực hành duyệt đồ thị theo chiều rộng.
- Thực hành ứng dụng kĩ thuật duyệt đồ thị trong một số bài toán thực tế.

Trong bài thực hành trước chúng ta đã được ôn tập và giải một số bài toán có áp dụng thuật toán duyệt đồ thị theo chiều sâu. Còn về thuật toán duyệt theo chiều rộng em có biết gì về các ứng dụng thực tế của bài toán này không?

Nhiệm vụ: Tìm đường đi xe đạp

Các bạn học sinh lớp em (được đánh số từ 0 đến n – 1) có nhà ở trải rộng khắp thành phố. Trong thành phố có những đường đi chỉ dành cho xe cơ giới, nhưng cũng có đường đi dành cho xe đạp. Các bạn học sinh lớp em chỉ biết đi xe đạp. Dữ liệu đầu vào gồm hai tệp. Tệp Danh-sach.inp sẽ lưu tên các bạn trong lớp, tên mỗi bạn ghi trên một dòng. Tệp thứ hai, xe-dap.inp mô tả các con đường có thể đi xe đạp từ nhà một bạn trong lớp đến nhà bạn khác. Tệp này cũng có nhiều dòng, mỗi dòng là hai số tự nhiên i, j cách nhau bởi dấu cách, chỉ ra từ nhà bạn thứ i có thể đi xe đạp được đến nhà bạn j. Các đường đi xe đạp này là hai chiều.

Danh-sach.inp

```
6
Hòa
Vinh
Lê
Quân
Quang
Vân
```

Xe-dap.inp

```
6
0 1
1 3
2 3
4 5
```

Tệp danh sách lớp học Danh-sach.inp bắt đầu là số tự nhiên n, n dòng tiếp theo mỗi dòng là tên của các bạn học sinh trong lớp.

Tệp lưu thông tin các đường đi xe đạp xe-dap.inp có dòng đầu tiên là số tự nhiên n, các dòng tiếp theo, mỗi dòng chỉ một đường đi bằng xe đạp giữa hai bạn học sinh trong lớp được mô tả bằng hai số tự nhiên cách nhau bởi dấu cách.

Yêu cầu của bài toán sau khi nhập dữ liệu như sau:

– Nhập từ bàn phím hai số tự nhiên i, j ($0 \le i < j \le n - 1$), hai số này sẽ tương ứng với hai học sinh trong lớp.

– Thông báo có cách đi xe đạp từ nhà bạn thứ i đến nhà bạn thứ j hay không, nếu có thì ghi ra dãy các đường đi lần lượt từ nhà bạn thứ i, đi qua một số nhà bạn khác, cuối cùng đến được nhà bạn thứ j.

Trong ví dụ trên, nếu nhập i = 0, j = 2 thì kết quả thể hiện đường đi xe đạp như sau:

```
Hòa -> Vinh -> Quân -> Lê
```

Còn nếu nhập hai chỉ số i = 0, j = 5 thì kết quả sẽ thông báo như sau:

```
Không tồn tại đường đi xe đạp từ nhà bạn Hòa đến nhà bạn Vân
```

Hướng dẫn

Từ dữ liệu đầu vào của bài toán (tệp Danh-sach.inp và Xe-dap.inp) chúng ta dễ dàng thiết lập được đồ thị vô hướng G = (V, E) ứng với danh sách kề Adj. Bài toán cho trước hai đỉnh i, j bất kì của đồ thị, cần tìm một đường đi (nếu có) từ i đến j. Bài toán này có thể giải được dễ dàng bằng cách duyệt đồ thị theo chiều rộng, bắt đầu từ đỉnh i, nếu trong quá trình duyệt gặp đỉnh j thì có thể thiết lập đường đi từ i đến j.

Trong hàm `BFS()` sau đây, chúng ta sử dụng hai mảng mark[] và prev[] như sau:

mark[v] = True khi và chỉ khi đỉnh v đã được đánh dấu khi duyệt đồ thị.

prev[v] = Đỉnh đã được duyệt trước v. Như vậy, nếu u = prev[v] tức là nếu có đường đi từ vị trí s ban đầu đến v thì u sẽ đứng ngay trước v.

Ban đầu toàn bộ mảng mark[] được gán giá trị False, toàn bộ mảng prev[] được gán giá trị None. Vậy nếu prev[v] = None, tức là không tồn tại đường đi từ s đến v.

Hàm `BFS(Adj,s)` theo chiều rộng bắt đầu từ đỉnh s như sau:

```
def BFS(Adj,s):
    Q = Queue()
    mark[s] = True
    enqueue(Q,s)
    while not isEmptyQueue(Q):
        v = dequeue(Q)
        for u in Adj[v]:
            if not mark[u]:
                mark[u] = True
                prev[u] = v
                enqueue(Q,u)
```

Để biểu diễn đường đi từ đỉnh s đến t chúng ta sử dụng hàm đệ quy `printpath(s,t)`. Trong hàm này sử dụng mảng names[] lưu tên các học sinh trong lớp. Mảng này là kết quả của hàm `Getnames(fname)`.

```
def printpath(s,t):
    if t == s:
        print(names[s], end = " ")
    else:
        if prev[t] == None:
            print("Không tồn tại đường đi")
        else:
            printpath(s,prev[t])
            print("->",names[t],end = " ")
```

Các tệp dữ liệu đầu vào Danh-sach.inp và Xe-dap.inp có thể được đặt cùng vị trí với tệp chương trình.

Hàm Getnames(fname) đọc dữ liệu từ tệp Danh-sach.inp và trả về mảng names lưu danh sách tên học sinh của lớp:

```
def Getnames(fname):
    f = open(fname,encoding="UTF-8")
    n = int(f.readline())
    names = []
    for i in range(n):
        names.append(f.readline().strip())
    f.close()
    return names
```

Hàm BuildGraph(fname) đọc dữ liệu từ tệp Xe-dap.inp và trả về dữ liệu chính của đồ thị vô hướng bao gồm V, Adj. Hàm này đã có trong Bài 12.

Sau khi tất cả các hàm trên đã được thiết lập, chúng ta có thể viết đoạn chương trình chính giải bài toán như sau:

```
from Queue import *
fname = "Danh-sach.inp"
fdata = "Xe-dap.inp"
names = Getnames(fname)
n = len(names)
V,Adj = BuildGraph(fdata)
mark = [False]*n
prev = [None]*n
st = input("Nhập số thứ tự hai bạn học sinh: ")
s,t = [int(ch) for ch in st.split()]
BFS(Adj,s)
printpath(s,t)
```

KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

LUYỆN TẬP

1. Sửa lại phần nhập dữ liệu hai học sinh: sẽ nhập trực tiếp tên hai học sinh, kiểm tra các tên này có nhập đúng không và thực hiện yêu cầu như trong chương trình trên.

2. Viết lại hàm BFS() trong chương trình trên nhưng sử dụng ma trận kề A thay thế cho danh sách kề Adj.

VẬN DỤNG

1. Bổ sung thêm yêu cầu của nhiệm vụ trên như sau: Có hay không hai bạn học sinh trong lớp mà không thể đi xe đạp từ nhà bạn này đến nhà bạn kia. Nếu có thì thông báo tên hai bạn học sinh đó.

2. Thiết lập hàm printpath(s,t) không đệ quy có tính năng tương tự hàm cùng tên trong chương trình trên.

BẢNG GIẢI THÍCH THUẬT NGỮ

	Thuật ngữ	Giải thích	Trang
B	BST	BST – Binary Search Tree: Cây tìm kiếm nhị phân.	30
	BFS	BFS – Breadth First Search: Duyệt theo chiều rộng.	77
C	Cây (Tree)	Cấu trúc thông tin bao gồm các nút và quan hệ cha-con giữa các nút. Mỗi cây có một nút gốc không có nút cha. Các nút còn lại đều có nút cha. Mỗi nút có thể có hoặc không có các nút con.	23
	Cây nhị phân	Cấu trúc cây mà mỗi nút có tối đa hai nút con, một nút con trái và một nút con phải.	24
	Cây tìm kiếm nhị phân	Cấu trúc cây nhị phân đặc biệt thoả mãn điều kiện về sắp xếp các khoá: tại mỗi nút, khoá sẽ nhỏ hơn tất cả các khoá thuộc cây con phải và lớn hơn tất cả các khoá thuộc cây con trái.	30
D	Danh sách kề	Dữ liệu biểu diễn thông tin của đồ thị, mỗi đỉnh sẽ gắn với danh sách các đỉnh kề với đỉnh này.	53
	DFS	DFS – Depth First Search: Duyệt theo chiều sâu.	65
Đ	Đồ thị	Đồ thị G = (V, E) bao gồm tập hợp các đỉnh V và tập hợp các cạnh E, mỗi cạnh nối hai đỉnh của V.	49
F	FIFO	FIFO – First In, First Out: Vào trước ra trước, là cơ chế hoạt động của hàng đợi.	7
H	Hàng đợi (Queue)	Cấu trúc dữ liệu tuyến tính được xây dựng theo cơ chế FIFO. Các thao tác chính bao gồm enqueue và dequeue.	7
L	LIFO	LIFO – Last In, First Out: Vào sau ra trước, là cơ chế hoạt động của ngăn xếp.	5
M	Ma trận kề	Ma trận biểu diễn thông tin của đồ thị, A[i][j] = 1 nếu tồn tại cạnh nối đỉnh i đến đỉnh j và A[i][j] = 0 nếu không tồn tại cạnh nối đỉnh i đến đỉnh j.	52
N	Ngăn xếp (Stack)	Cấu trúc dữ liệu tuyến tính được xây dựng theo cơ chế LIFO. Các thao tác chính bao gồm push và pop.	5

Nhà xuất bản Giáo dục Việt Nam xin trân trọng cảm ơn các tác giả có tác phẩm, tư liệu được sử dụng, trích dẫn trong cuốn sách này.

Chịu trách nhiệm xuất bản:
Chủ tịch Hội đồng thành viên – Tổng Giám đốc NGUYỄN TIẾN THANH

Chịu trách nhiệm nội dung:
Tổng biên tập PHẠM VĨNH THÁI

Biên tập lần đầu: NGUYỄN THỊ THANH XUÂN – PHẠM THỊ THANH NAM
Biên tập tái bản: PHẠM THỊ THANH NAM
Biên tập mĩ thuật: NGUYỄN BÍCH LA
Thiết kế sách: PHAN THỊ THANH HOA
Trình bày bìa: NGUYỄN BÍCH LA
Minh hoạ: NGUYỄN THỊ HUẾ
Sửa bản in: PHẠM THỊ TÌNH – TẠ THỊ HƯỜNG
Chế bản: CÔNG TY CỔ PHẦN MĨ THUẬT VÀ TRUYỀN THÔNG

Bản quyền © (2024) thuộc Nhà xuất bản Giáo dục Việt Nam.

Xuất bản phẩm đã đăng kí quyền tác giả. Tất cả các phần của nội dung cuốn sách này đều không được sao chép, lưu trữ, chuyển thể dưới bất kì hình thức nào khi chưa có sự cho phép bằng văn bản của Nhà xuất bản Giáo dục Việt Nam.

CHUYÊN ĐỀ HỌC TẬP TIN HỌC 12 – ĐỊNH HƯỚNG KHOA HỌC MÁY TÍNH

Mã số: G1HHZI007h25
In ..... bản, (QĐ .....) khổ 19 x 26,5 cm.
Đơn vị in:.....
Cơ sở in:.....
Số ĐKXB: 6-2025/CXBIPH/314-11/GD
Số QĐXB: .../..... ngày ... tháng ... năm .....
In xong và nộp lưu chiểu quý ..... năm .....
Mã số ISBN: 978-604-0-39254-1



HUÂN CHƯƠNG HỒ CHÍ MINH

BỘ SÁCH GIÁO KHOA LỚP 12 – KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

1. Ngữ văn 12, tập một
2. Ngữ văn 12, tập hai
3. Chuyên đề học tập Ngữ văn 12
4. Toán 12, tập một
5. Toán 12, tập hai
6. Chuyên đề học tập Toán 12
7. Lịch sử 12
8. Chuyên đề học tập Lịch sử 12
9. Địa lí 12
10. Chuyên đề học tập Địa lí 12
11. Giáo dục kinh tế và pháp luật 12
12. Chuyên đề học tập Giáo dục kinh tế và pháp luật 12
13. Vật lí 12
14. Chuyên đề học tập Vật lí 12
15. Hoá học 12
16. Chuyên đề học tập Hoá học 12
17. Sinh học 12
18. Chuyên đề học tập Sinh học 12
19. Công nghệ 12 – Công nghệ Điện – Điện tử
20. Chuyên đề học tập Công nghệ 12 – Công nghệ Điện – Điện tử
21. Công nghệ 12 – Lâm nghiệp – Thuỷ sản
22. Chuyên đề học tập Công nghệ 12 – Lâm nghiệp – Thuỷ sản
23. Tin học 12 – Định hướng Tin học ứng dụng
24. Chuyên đề học tập Tin học 12 – Định hướng Tin học ứng dụng
25. Tin học 12 – Định hướng Khoa học máy tính
26. Chuyên đề học tập Tin học 12 – Định hướng Khoa học máy tính
27. Mĩ thuật 12 – Thiết kế mĩ thuật đa phương tiện
28. Mĩ thuật 12 – Thiết kế đồ hoạ
29. Mĩ thuật 12 – Thiết kế thời trang
30. Mĩ thuật 12 – Thiết kế mĩ thuật sân khấu, điện ảnh
31. Mĩ thuật 12 – Lí luận và lịch sử mĩ thuật
32. Mĩ thuật 12 – Điêu khắc
33. Mĩ thuật 12 – Kiến trúc
34. Mĩ thuật 12 – Hội hoạ
35. Mĩ thuật 12 – Đồ hoạ (tranh in)
36. Mĩ thuật 12 – Thiết kế công nghiệp
37. Chuyên đề học tập Mĩ thuật 12
38. Âm nhạc 12
39. Chuyên đề học tập Âm nhạc 12
40. Hoạt động trải nghiệm, hướng nghiệp 12
41. Giáo dục thể chất 12 – Bóng chuyền
42. Giáo dục thể chất 12 – Bóng đá
43. Giáo dục thể chất 12 – Cầu lông
44. Giáo dục thể chất 12 – Bóng rổ
45. Giáo dục quốc phòng và an ninh 12
46. Tiếng Anh 12 – Global Success – Sách học sinh

Website: https://www.nxbgd.vn

KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

ISBN 978-604-0-39254-1

9 786040 392541

Giá: 16.000 đ