Chân trời sáng tạo

HOÀNG VĂN KIẾM (Tổng Chủ biên) – NGÔ QUỐC VIỆT (Chủ biên)
TRẦN QUANG VĨNH CHÁNH – TRẦN HÀ SƠN – NGUYỄN ĐẶNG TRÍ TÍN

CHUYÊN ĐỀ HỌC TẬP
TIN HỌC

ĐỊNH HƯỚNG KHOA HỌC MÁY TÍNH

12

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

HOÀNG VĂN KIẾM (Tổng Chủ biên) – NGÔ QUỐC VIỆT (Chủ biên)
TRẦN QUANG VĨNH CHÁNH – TRẦN HÀ SƠN – NGUYỄN ĐẶNG TRÍ TÍN

CHUYÊN ĐỀ HỌC TẬP

TIN HỌC

ĐỊNH HƯỚNG KHOA HỌC MÁY TÍNH

12

Chân trời sáng tạo

NHÀ XUẤT BẢN GIÁO DỤC VIỆT NAM

HƯỚNG DẪN SỬ DỤNG SÁCH

Mỗi bài học đều được thiết kế bao gồm mục tiêu và các hoạt động dạy và học. Các hoạt động trọng tâm được gắn thêm hình ảnh nhận diện là các "biểu tượng" hay "icon"

là những gì em sẽ đạt được sau bài học. Bắt đầu vào bài học, em cần đọc mục tiêu để biết các yêu cầu của bài học. Trước khi kết thúc bài học, em cần so sánh những gì đã học được với mục tiêu của bài.

là hoạt động để gợi mở, tạo hứng thú học tập và định hướng cho các em suy nghĩ, khám phá nội dung bài học. Em sẽ giải quyết được vấn đề đặt ra ở phần này khi tìm hiểu nội dung ở phần **Khám phá**.

là nội dung chính của bài học. Trong đó, **Đọc và quan sát**, **Làm** và **Ghi nhớ** là ba hoạt động cần thực hiện để hoàn thành cơ bản các nhiệm vụ học tập.

Đọc và quan sát - gặp biểu tượng này, em cần đọc, quan sát để tìm hiểu kiến thức, kĩ năng mới của bài học.

Làm - thực hiện các yêu cầu để hoàn thành nhiệm vụ học tập này giúp em khám phá, lĩnh hội kiến thức, kĩ năng mới của bài học.

Ghi nhớ - tóm tắt ngắn gọn kiến thức, kĩ năng trọng tâm của mỗi phần nội dung bài học mà em cần ghi nhớ.

là nội dung gồm các câu hỏi, bài tập để củng cố kiến thức, kĩ năng trong bài học.

là hoạt động rèn luyện thao tác sử dụng thiết bị máy tính, phần mềm tin học.

là nội dung gồm các câu hỏi, bài tập, tình huống, vấn đề thực tiễn mà em cần vận dụng kiến thức, kĩ năng vừa học để giải quyết.

Ngoài ra:

Các hình ảnh trong sách không chỉ là minh hoạ mà còn là một phần quan trọng của nội dung học tập. Các em cần "đọc" được nội dung của hình ảnh (quan sát, tìm hiểu, so sánh,...) để hoàn thành nhiệm vụ học tập. Kĩ năng có được của các em thông qua quá trình làm việc với kênh hình (kênh thông tin về hình ảnh) là yếu tố quan trọng để phát triển năng lực tự tìm hiểu, khám phá phần mềm máy tính trong môn Tin học.

Các chữ số đặt trong vòng tròn (❶, ❷, ❸,...) được dành riêng để đánh số thứ tự các thao tác, công việc cần được thực hiện theo trình tự. Điều này giúp các em dễ dàng nhận biết các bước thực hiện nhiệm vụ và thuận tiện để đối chiếu, tra cứu khi thực hành trên máy tính.

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

LỜI NÓI ĐẦU

Các em học sinh thân mến!

Chuyên đề học tập Tin học 12 - Định hướng Khoa học máy tính giới thiệu hai cấu trúc dữ liệu quan trọng trong lập trình là hàng đợi và ngăn xếp, cùng tìm hiểu những thuật toán hiệu quả để giải quyết bài toán tìm kiếm và sắp xếp, hỗ trợ các em trong việc sử dụng ngôn ngữ lập trình để phát triển tư duy lập trình và năng lực giải quyết vấn đề.

Quyển sách gồm 13 bài học, thời lượng 35 tiết học với ba chuyên đề có sự liên kết chặt chẽ với nhau: tìm hiểu một vài kiểu dữ liệu tuyến tính; tìm hiểu cây tìm kiếm nhị phân trong sắp xếp và tìm kiếm; tìm hiểu kĩ thuật duyệt đồ thị và ứng dụng.

Quyển sách được tích hợp các hoạt động dạy học phát triển năng lực, khuyến khích làm việc theo nhóm, giúp phát triển kĩ năng mềm, kích thích sự tò mò, sáng tạo, tạo động lực, niềm đam mê để các em tiếp tục tìm hiểu, khám phá và mở rộng hiểu biết.

Chúc các em học tốt và vận dụng hiệu quả kiến thức, kĩ năng vào học tập và thực tiễn cuộc sống!

CÁC TÁC GIẢ

MỤC LỤC

Chuyên đề 1

TÌM HIỂU MỘT VÀI KIỂU DỮ LIỆU TUYẾN TÍNH

BÀI 1.1

HÀNG ĐỢI

MỤC TIÊU **Sau bài học này, em sẽ:**

- Mô tả được khái niệm hàng đợi thông qua cơ chế hoạt động của hàng đợi.
- Biểu diễn được hàng đợi bằng mảng một chiều.
- Giải thích và viết được chương trình cho các phép toán cơ bản của hàng đợi: Khởi tạo hàng đợi rỗng, thêm phần tử vào hàng đợi, lấy phần tử ra khỏi hàng đợi.

KHỞI ĐỘNG

Khi làm thủ tục tại các cơ quan hành chính nhà nước, em sẽ gặp các hệ thống xếp hàng tự động (*Hình 1*). Theo em, các hệ thống này hoạt động theo nguyên tắc nào?

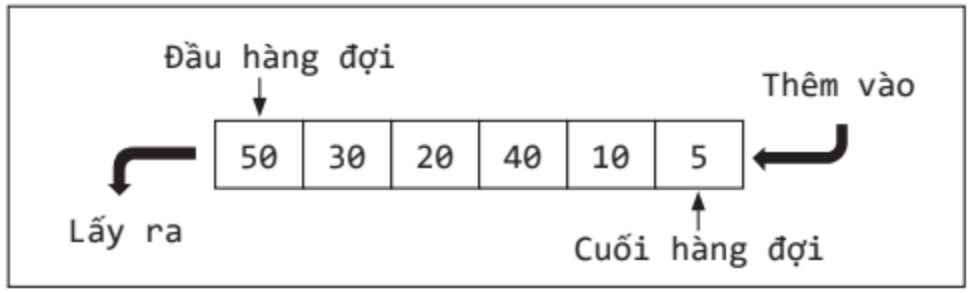
Hình 1. Hệ thống xếp hàng tự động

KHÁM PHÁ

1. Hàng đợi

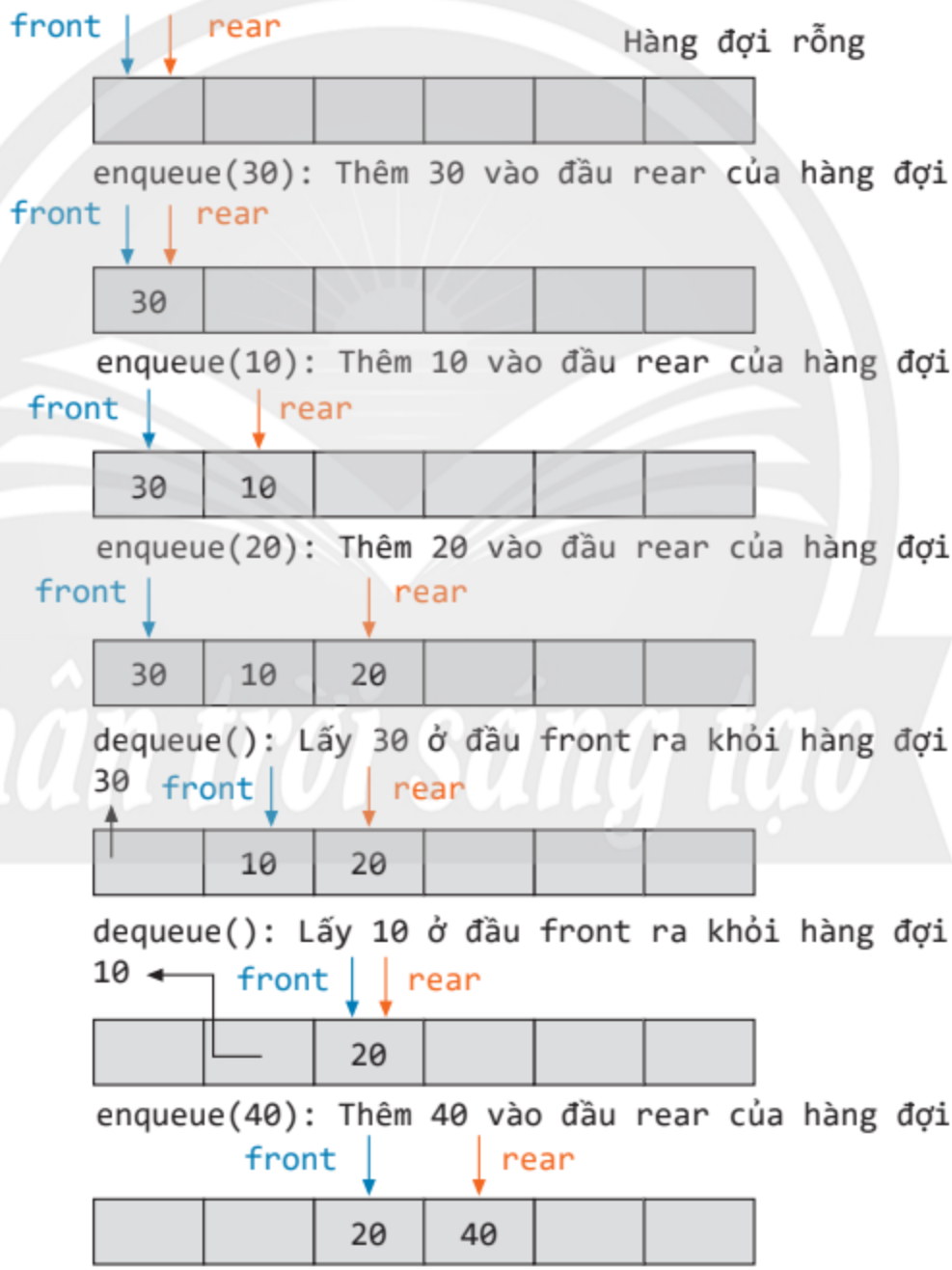
Hiện nay, nhiều ứng dụng như quản lí máy in trong hệ điều hành, quản lí truy xuất dữ liệu trang web trong web server,… đều áp dụng quy tắc "Yêu cầu trước sẽ được phục vụ trước" hay "vào trước ra trước". Trong lập trình, có một cấu trúc dữ liệu cũng áp dụng cơ chế hoạt động "vào trước ra trước" như vậy, được đặt tên là hàng đợi (queue).

Hàng đợi là một cấu trúc dữ liệu tuyến tính dùng để lưu danh sách các phần tử. Thao tác thêm phần tử vào hàng đợi được thực hiện ở một đầu và lấy phần tử ra khỏi hàng đợi được thực hiện ở đầu còn lại. Phần tử được thêm vào hàng đợi trước thì sẽ được lấy ra trước, phần tử được thêm vào sau sẽ được lấy ra sau. Nói cách khác, thứ tự phục vụ các phần tử trong hàng đợi sẽ căn cứ theo thứ tự các phần tử đó được thêm vào hàng đợi, gọi là vào trước ra trước (FIFO – **F**irst **I**n, **F**irst **O**ut). Phần tử được thêm mới luôn ở cuối hàng đợi, phần tử được lấy ra để xử lí luôn ở đầu hàng đợi. Thao tác thêm mới phần tử vào cuối hàng đợi được gọi là thêm vào (enqueue). Thao tác lấy phần tử đầu hàng đợi được gọi là lấy ra (dequeue). *Hình 2* minh hoạ cơ chế FIFO của hàng đợi.



Hình 2. Cơ chế FIFO của hàng đợi

Ngoài thao tác enqueue và dequeue, em có thể sử dụng thêm thao tác front để trả về giá trị của phần tử đầu hàng đợi, thao tác rear (hay còn gọi là back) để trả về giá trị của phần tử cuối hàng đợi và thao tác kiểm tra hàng đợi rỗng. Chẳng hạn, trong *Hình 3* minh hoạ các thao tác enqueue, dequeue trên hàng đợi.



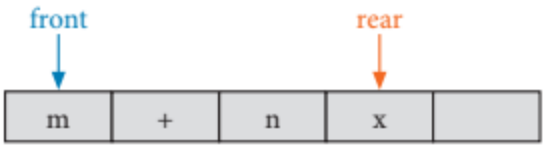
Hình 3. Thao tác enqueue, dequeue, trên hàng đợi và hai đầu front, rear

1. Cho *Hình 4* biểu diễn một hàng đợi, hãy cho biết:

a) Phần tử đầu hàng đợi, phần tử cuối hàng đợi.

b) Sau khi lấy ra một phần tử, thì phần tử đầu hàng đợi là phần tử nào?

c) Sau khi thêm vào phần tử "k" vào thì phần tử cuối hàng đợi là phần tử nào?



Hình 4. Hàng đợi và hai đầu front, rear

2. Cho một hàng đợi rỗng, hãy vẽ hình minh hoạ từng bước thực hiện các thao tác sau: enqueue(1), enqueue(3), enqueue(5), dequeue(), dequeue(), enqueue(7).

Hàng đợi là một cấu trúc dữ liệu tuyến tính dùng để lưu danh sách các phần tử. Hai thao tác cơ bản trên hàng đợi là thao tác thêm vào (enqueue) ở cuối hàng đợi (rear) và thao tác lấy ra (dequeue) ở đầu hàng đợi (front). Hai thao tác này thể hiện cơ chế hoạt động "Vào trước – Ra trước" (FIFO – First In, First Out).

2. Biểu diễn và cài đặt hàng đợi bằng mảng một chiều

a) Biểu diễn hàng đợi bằng mảng một chiều

Hàng đợi là một dãy các phần tử. Do đó, em có thể dùng mảng một chiều để biểu diễn hàng đợi. Phép thêm vào (enqueue) được thực hiện ở đầu rear và phép lấy ra (dequeue) được thực hiện ở đầu front. *Hình 5* minh hoạ hàng đợi có 7 phần tử trong mảng một chiều. Phần tử đầu của hàng đợi được đặt vào phần tử có chỉ số front = 0 của mảng, phần tử cuối của hàng đợi được đặt vào phần tử có chỉ số rear = 6 của mảng.

Hình 5a. Hàng đợi

40	20	30	10	60	50	70
0	1	2	3	4	5	6

Hình 5b. Biểu diễn hàng đợi bằng mảng một chiều

Hình 5. Biểu diễn hàng đợi

1. Các thông tin cần thiết để biểu diễn hàng đợi bằng mảng một chiều là gì?

2. Với hàng đợi ở *Hình 5*, hãy vẽ hình khi thực hiện liên tục các thao tác: thêm vào 0, lấy ra, lấy ra.

b) Cài đặt hàng đợi bằng mảng một chiều trong Python

Hàng đợi được biểu diễn bằng mảng một chiều. Để đơn giản, em cài đặt hàng đợi bằng danh sách kiểu list của Python. Các phép toán enqueue, dequeue, front, rear được thực hiện bằng các hàm của kiểu list. Do đó, đầu lấy ra front luôn là 0 (front = 0) và đầu thêm vào rear luôn ở cuối danh sách. Các phép toán cơ bản cho hàng đợi được cài đặt thông qua các hàm sau:

Khởi tạo hàng đợi rỗng.

```
01. #Khởi tạo hàng đợi rỗng
02. def initQueue():
03.   return []
```

Kiểm tra hàng đợi rỗng.

```
01. #Kiểm tra hàng đợi rỗng
02. def isEmptyQueue(queue):
03.   return len(queue) == 0
```

Phép toán enqueue: sử dụng hàm append() để thêm phần tử vào cuối hàng đợi.

```
01. #Thêm phần tử vào cuối hàng đợi
02. def enqueue(queue, val):
03.   queue.append(val)                    #Thêm phần tử vào cuối hàng đợi
```

Phép toán dequeue: thao tác dequeue bắt đầu bằng việc kiểm tra hàng đợi có rỗng hay không. Nếu không, sử dụng hàm pop() với chỉ số 0 để xoá phần tử đầu hàng đợi.

```
01. #Lấy phần tử ra khỏi đầu hàng đợi
02. def dequeue(queue):
03.     if isEmptyQueue(queue):          #Kiểm tra hàng đợi rỗng
04.       raise ValueError("Hàng đợi rỗng.")
05.     return queue.pop(0)              #Xoá phần tử đầu ra khỏi hàng đợi
```

Phép toán front: kiểm tra nếu hàng đợi không rỗng thì trả về giá trị của queue[0].

```
01. #Lấy giá trị đầu hàng đợi
02. def front(queue):
03.   if isEmptyQueue(queue):            #Kiểm tra hàng đợi rỗng
04.     raise ValueError("Hàng đợi rỗng.")
05.   return queue[0]
```

Phép toán rear: kiểm tra nếu hàng đợi không rỗng thì trả về giá trị của queue[len(queue) – 1].

```
01. #Lấy giá trị cuối hàng đợi
02. def rear(queue):
03.   if isEmptyQueue(queue):            #Kiểm tra hàng đợi rỗng
04.      raise ValueError("Hàng đợi rỗng.")
05.  return queue[len(queue)-1]          #Phần tử cuối mảng, cũng là cuối hàng đợi
```

1. Tại sao không cần sử dụng các chỉ số front, rear khi dùng kiểu list để biểu diễn hàng đợi trong Python?

2. Theo em, có cách nào kiểm tra hàng đợi queue là rỗng mà không dùng hàm len(queue)?

Hàng đợi là một dãy các phần tử. Do đó, hàng đợi có thể được biểu diễn bằng mảng một chiều hoặc danh sách (kiểu list của Python). Khi hàng đợi là danh sách (kiểu list), hàm thêm vào enqueue() dùng hàm append() của kiểu list và hàm lấy ra dequeue() dùng hàm pop() của kiểu list.

LUYỆN TẬP

1. Trong Python, khi sử dụng kiểu list để biểu diễn hàng đợi. Hãy cho biết:

a) Chỉ số của phần tử đầu.

b) Chỉ số của phần tử cuối.

2. Theo em, thứ tự thực hiện phép toán enqueue với các giá trị thích hợp để kết quả là một hàng đợi trong *Hình 5* là những bước nào?

VẬN DỤNG

1. Các phần tử trong hàng đợi biểu diễn bằng kiểu list trong Python có thể thuộc kiểu chuỗi hay không? Nếu có, sử dụng các hàm initQueue(), enqueue() để tạo hàng đợi có các phần tử như sau:

"Một"	"Hai"	"Ba"	"Bốn"

Sau đó, sử dụng các hàm enqueue(), dequeue() để hàng đợi trên có kết quả là:

"Bốn"	"Ba"	"Hai"	"Một"	"Không"

2. Theo em, có thể dùng danh sách liên kết để biểu diễn hàng đợi hay không?

3. Tạo tệp queue.py chứa các hàm enqueue(), dequeue(), front(), rear(), isEmptyQueue() của hàng đợi. Sau đó:

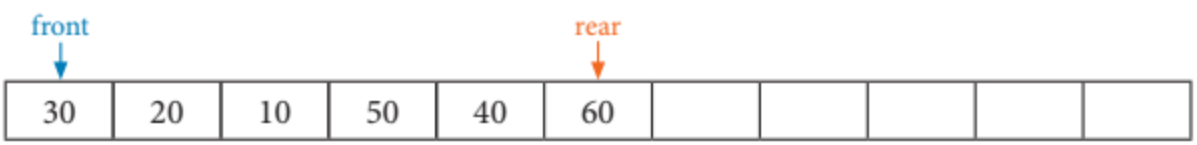
– Khởi tạo hàng đợi rỗng.

– Thực hiện các hàm enqueue() với giá trị thích hợp để hàng đợi có kết quả như *Hình 6a*.

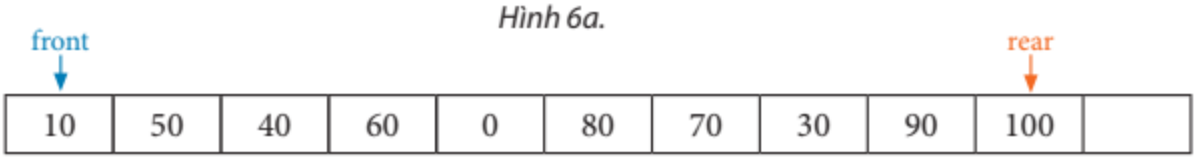
– Thực hiện các hàm enqueue(), dequeue() với các giá trị thích hợp để hàng đợi có kết quả như *Hình 6b*.

Hướng dẫn:

1. Sử dụng các hàm đã trình bày trong mục 2b của phần KHÁM PHÁ.
2. Gọi các hàm enqueue(), dequeue() theo thứ tự cụ thể để thực hiện.



Hình 6a.



Hình 6b.

Hình 6. Hàng đợi sau khi thực hiện các phép toán enqueue, dequeue

BÀI 1.2 NGĂN XẾP

MỤC TIÊU **Sau bài học này, em sẽ:**

- Mô tả được khái niệm ngăn xếp thông qua cơ chế hoạt động của ngăn xếp.
- Biểu diễn được ngăn xếp bằng mảng một chiều.
- Giải thích và viết được chương trình cho các phép toán cơ bản của ngăn xếp: khởi tạo ngăn xếp rỗng, thêm phần tử vào ngăn xếp, lấy phần tử ra khỏi ngăn xếp.

KHỞI ĐỘNG

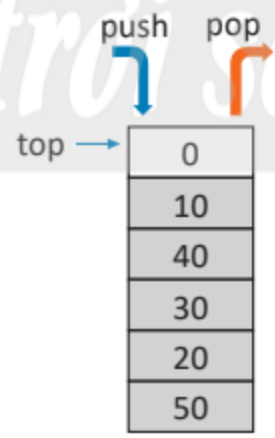
Quan sát *Hình 1* và cho biết cách thêm đĩa mới vào và lấy ra một đĩa từ chồng đĩa.

Hình 1. Ngăn xếp chứa các đĩa

KHÁM PHÁ

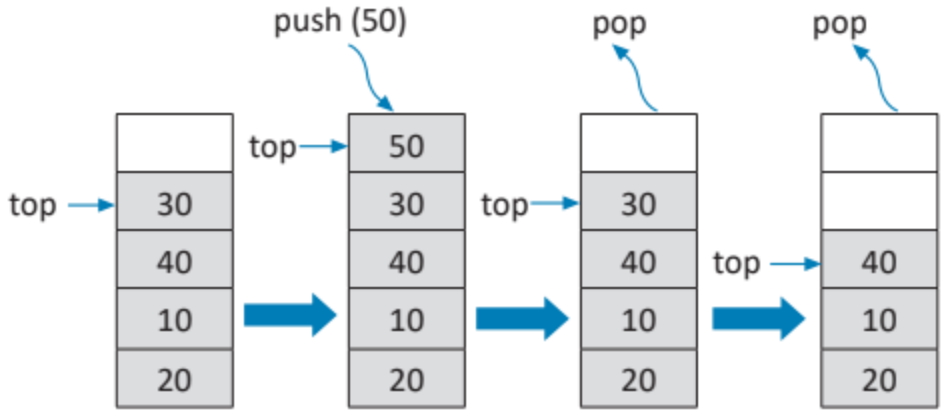
1. Ngăn xếp

Ngăn xếp là một cấu trúc dữ liệu tuyến tính dùng để lưu danh sách các phần tử, trong đó việc thêm một phần tử mới và lấy một phần tử hiện có diễn ra ở cùng một đầu, gọi là đỉnh (top) của ngăn xếp. Phần tử nào được thêm vào ngăn xếp sau cùng thì sẽ được lấy ra trước tiên, phần tử nào được thêm vào ngăn xếp đầu tiên sẽ được lấy ra sau cùng. Cơ chế này gọi là vào sau ra trước (LIFO – Last In, First Out). Thao tác thêm mới phần tử vào đỉnh ngăn xếp được gọi là thêm vào (push). Thao tác lấy phần tử ra khỏi đỉnh ngăn xếp được gọi là lấy ra (pop). *Hình 2* minh hoạ cơ chế LIFO của ngăn xếp.



Hình 2. Cơ chế LIFO của ngăn xếp

Ngoài các thao tác push và pop, em có thể sử dụng thêm thao tác top để trả về giá trị của phần tử ở đỉnh ngăn xếp và thao tác kiểm tra ngăn xếp rỗng. Chẳng hạn, trong *Hình 3* minh hoạ các thao tác push, pop và top trên ngăn xếp.



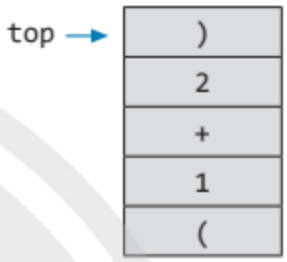
Hình 3. Thao tác push, pop trên ngăn xếp và đỉnh top

Hình 4 biểu diễn một ngăn xếp. Cho biết:

a) Phần tử nào ở đỉnh của ngăn xếp.

b) Sau khi lấy ra một phần tử, thì ngăn xếp gồm các phần tử nào.

c) Sau khi thêm phần tử "x" vào, thì phần tử nào ở đỉnh của ngăn xếp.



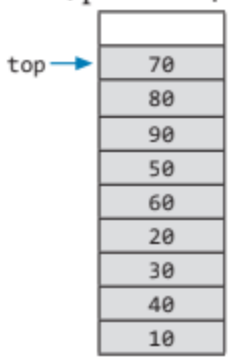
Hình 4. Ngăn xếp với các phần tử là các số, phép toán, dấu ngoặc

Ngăn xếp là một cấu trúc dữ liệu tuyến tính dùng để lưu danh sách các phần tử. Hai thao tác cơ bản trên ngăn xếp là: thao tác thêm vào (push) và thao tác lấy ra (pop) đều ở đỉnh ngăn xếp (top). Hai thao tác này thể hiện cơ chế hoạt động "Vào sau – Ra trước" (LIFO – Last In, First Out).

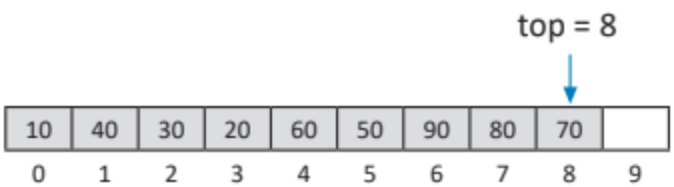
2. Biểu diễn và cài đặt ngăn xếp bằng mảng một chiều

a) Biểu diễn ngăn xếp bằng mảng một chiều

Tương tự như hàng đợi, ngăn xếp dùng để lưu trữ dãy các phần tử tương tự mảng một chiều. Vì vậy, em có thể dùng mảng một chiều để biểu diễn ngăn xếp. Các thao tác thêm vào hoặc lấy ra chỉ diễn ra ở đỉnh của ngăn xếp. *Hình 5* minh hoạ biểu diễn ngăn xếp có 9 phần tử bằng mảng một chiều, phần tử tại đỉnh (top) của ngăn xếp được đặt vào phần tử có chỉ số 8 của mảng.



Hình 5a. Ngăn xếp



Hình 5b. Biểu diễn ngăn xếp bằng mảng một chiều

Hình 5. Biểu diễn ngăn xếp

b) Cài đặt ngăn xếp

Ngăn xếp được biểu diễn bằng mảng một chiều. Để đơn giản, em cài đặt ngăn xếp bằng danh sách kiểu list của Python. Các phép toán push, pop, top được thực hiện bằng các hàm của kiểu list. Do đó, đầu lấy ra và đầu thêm vào phần tử luôn ở cuối danh sách kiểu list của Python. Các phép toán cơ bản cho ngăn xếp được cài đặt thông qua các hàm sau:

Khởi tạo ngăn xếp rỗng.

```
01. #Khởi tạo ngăn xếp rỗng
02. def initStack():
03.   return []
```

Kiểm tra ngăn xếp rỗng.

```
01. #Kiểm tra ngăn xếp rỗng
02. def isEmptyStack(stack):
03.   return len(stack) == 0
```

Phép toán push: thêm phần tử vào đỉnh ngăn xếp. Phép toán push diễn ra ở đỉnh ngăn xếp. Phần tử ở đáy ngăn xếp luôn có chỉ số bằng 0.

```
01. #Thêm phần tử vào ngăn xếp
02. def push(stack, val):
03.   stack.append(val)                    #Thêm phần tử vào cuối danh sách
```

Phép toán pop: thao tác pop bắt đầu bằng việc kiểm tra ngăn xếp có rỗng hay không. Nếu không, trả về phần tử cuối mảng.

```
01. #Trả về giá trị và xoá phần tử ra khỏi đỉnh ngăn xếp
02. def pop(stack):
03.   if isEmptyStack(stack):
04.     raise ValueError("Ngăn xếp rỗng.")
05.   return stack.pop()                   #Trả về phần tử cuối danh sách
```

Phép toán top: trả về giá trị của phần tử ở đỉnh ngăn xếp bằng cách trả về giá trị của phần tử cuối mảng.

```
01. #Trả về giá trị của phần tử ở đỉnh ngăn xếp
02. def top(stack):
03.   if isEmptyStack(stack):
04.     raise ValueError("Ngăn xếp rỗng.")
05.   return stack[-1]                     #Phần tử đỉnh ngăn xếp
```

1. Để biểu diễn ngăn xếp bằng mảng một chiều, em cần sử dụng những thông tin gì?

2. Vì sao có thể dùng danh sách (kiểu list của Python) để biểu diễn ngăn xếp?

Ngăn xếp là một dãy các phần tử. Do đó, ngăn xếp có thể được biểu diễn bằng mảng một chiều hoặc danh sách (kiểu list của Python). Khi ngăn xếp là danh sách (kiểu list), hàm thêm vào push() dùng hàm append() của kiểu list và hàm lấy ra pop() dùng hàm pop() của kiểu list.

LUYỆN TẬP

1. Trong Python, khi sử dụng kiểu list để biểu diễn ngăn xếp. Hãy cho biết:

a) Chỉ số của phần tử đỉnh.

b) Chỉ số của phần tử đáy.

2. Hãy vẽ lại *Hình 5*, cập nhật giá trị top khi thực hiện tuần tự các thao tác sau đây: push(0), pop(), pop(), push(100).

VẬN DỤNG

1. Để tính giá trị một biểu thức số học bằng máy tính, một số nhà khoa học đã sử dụng cách biểu diễn dạng tiền tố (hay còn gọi là kí pháp Ba lan). Ví dụ, biểu thức số học (2–7/3)*(4–1) sẽ được chuyển sang dạng tiền tố có dạng *–2/73–41 (toán tử đặt trước toán hạng) trước khi tính giá trị. Sử dụng các hàm initStack(), push() để tạo ngăn xếp có các phần tử như sau:

"("	2	"–"	7	"/"	3	")"	"*"	"("	4	"–"	1	")"

Sau đó, sử dụng các hàm push(), pop() để ngăn xếp trên có kết quả là:

"*"	"–"	2	"/"	7	3	"–"	4	1

2. Theo em dùng danh sách liên kết để biểu diễn ngăn xếp được hay không?

3. Tạo tệp stack.py chứa các hàm push(), pop(), top(), isEmptyStack() của ngăn xếp. Sau đó:

– Tạo ngăn xếp rỗng.

– Thực hiện các hàm push() với giá trị thích hợp để ngăn xếp có kết quả như *Hình 6a*.

– Thực hiện các hàm push(), pop() với các giá trị thích hợp để ngăn xếp có kết quả như *Hình 6b*.

Hướng dẫn:

1. Sử dụng các hàm đã trình bày trong mục 2b của phần KHÁM PHÁ.
2. Gọi các hàm push(), pop() theo thứ tự cụ thể để thực hiện.

10
20
40
60
70
90
30
50
80

Hình 6a.

100
90
30
70
50
80

Hình 6b.

Hình 6. Ngăn xếp sau khi thực hiện các phép toán push, pop

BÀI 1.3 ỨNG DỤNG CỦA HÀNG ĐỢI

MỤC TIÊU **Sau bài học này, em sẽ:**

- Trình bày và minh hoạ được một số ứng dụng hàng đợi.
- Viết được một chương trình đơn giản để hiểu rõ ứng dụng của hàng đợi.

KHỞI ĐỘNG

Em hãy liệt kê một số hoạt động hàng ngày cần dùng đến hàng đợi.

KHÁM PHÁ

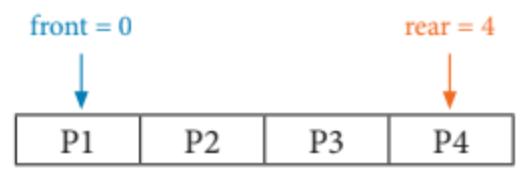
1. Một số ứng dụng của hàng đợi

Hàng đợi với cơ chế vào trước ra trước được ứng dụng để giải quyết một số vấn đề trong các lĩnh vực Công nghệ thông tin, dịch vụ, kinh doanh,… Ví dụ, trong lĩnh vực Công nghệ thông tin, hàng đợi được sử dụng để giải quyết các vấn đề như phân bố tài nguyên của hệ điều hành, thứ tự gửi email, in tài liệu. Trong kinh doanh, hàng đợi được sử dụng để sắp xếp thứ tự xử lí đơn hàng, chăm sóc khách hàng. Một số ứng dụng phổ biến của hàng đợi là:

Sắp xếp lịch trình: Sử dụng hàng đợi để quản lí các nhiệm vụ cần thực hiện. Một trong những cách tiếp cận đơn giản được gọi là "đến trước, phục vụ trước" (FCFS – First Come, First Served). Nghĩa là các nhiệm vụ được thêm vào cuối hàng đợi với thao tác enqueue theo thứ tự yêu cầu. Sau đó, các nhiệm vụ được lấy ra khỏi hàng đợi với thao tác dequeue để thực hiện. *Hình 1* minh hoạ 4 nhiệm vụ P1, P2, P3, P4 được thêm vào hàng đợi theo quy tắc "đến trước, phục vụ trước".

Quá trình	Thời gian thực hiện (Đơn vị: ngày)
P1	14
P2	3
P3	6
P4	2

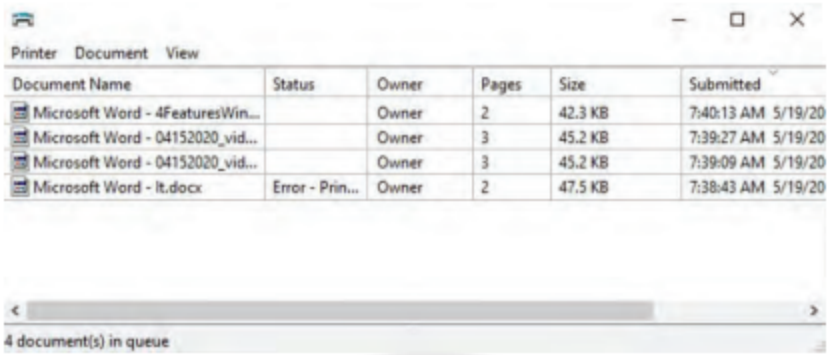
Hình 1a. Thứ tự thực hiện các quá trình



Hình 1b. Hàng đợi công việc chứa các quá trình cần thực hiện

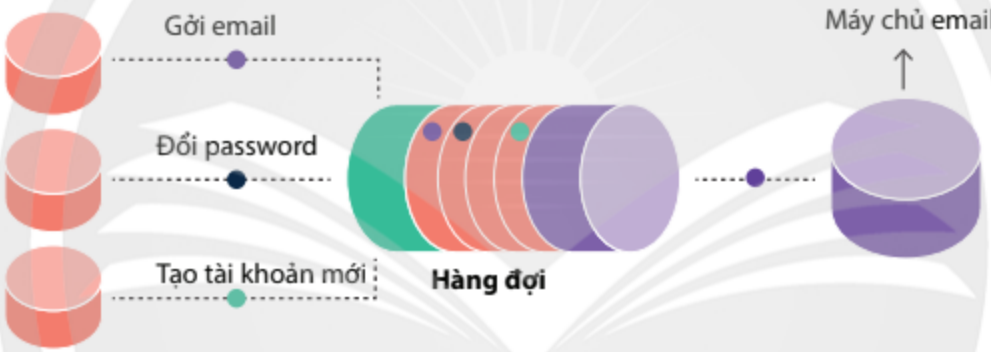
Hình 1. Cơ chế "Đến trước - Phục vụ trước" dùng hàng đợi để thực hiện các quá trình

Hàng đợi máy in: Được sử dụng để quản lí thứ tự xử lí lệnh in. Khi nhiều tài liệu cần được in, thì các tài liệu được thêm vào vùng đệm máy in. Các tài liệu cần in được thêm vào hàng đợi máy in với thao tác enqueue, sau đó máy in sẽ tuần tự lấy tài liệu cần in với thao tác dequeue để thực hiện in. *Hình 2* minh hoạ cửa sổ giao diện các tài liệu cần in được thêm vào hàng đợi.



Hình 2. Minh hoạ hàng đợi máy in

Máy chủ email hay web: Sử dụng hàng đợi để quản lí các yêu cầu đến từ máy khách, nhằm đảm bảo xử lí các yêu cầu theo thứ tự thích hợp (chẳng hạn như yêu cầu nào đến trước sẽ được phục vụ trước). Các yêu cầu được enqueue vào hàng đợi và được dequeue để xử lí theo cơ chế FIFO. *Hình 3* minh hoạ các yêu cầu từ máy khách được thêm vào hàng đợi để chờ máy chủ email xử lí.



Hình 3. Minh hoạ hàng đợi yêu cầu

1. Theo em, hàng đợi có những ứng dụng nào?
2. Hàng đợi máy in được sử dụng như thế nào?

Hàng đợi với cơ chế vào trước ra trước được ứng dụng để giải quyết một số vấn đề trong các lĩnh vực công nghệ thông tin, kinh doanh,… Trong lĩnh vực công nghệ thông tin, hàng đợi được sử dụng để giải quyết các vấn đề như sắp xếp lịch, phân bố tài nguyên của máy tính, thứ tự gửi email, in tài liệu,…

LUYỆN TẬP

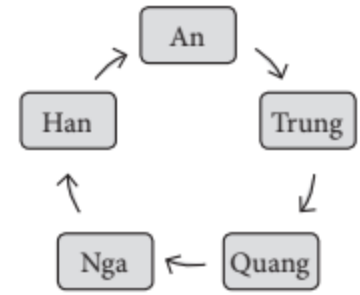
1. Hãy cho biết máy chủ email thường xử lí các yêu cầu từ nhiều người dùng theo thứ tự như thế nào.

2. Trình bày ứng dụng của hàng đợi trong thuật toán First-Come, First-Served thường được dùng trong sắp xếp các tiến trình chờ xử lí của CPU.

THỰC HÀNH

Nhiệm vụ. Trò chơi Hot Potato

Các người chơi đứng thành vòng tròn và lần lượt chuyển củ khoai cho người bên cạnh theo chiều kim đồng hồ trên nền nhạc. Khi nhạc dừng, lượt chơi dừng tạm thời, người nào đang giữ củ khoai sẽ bị loại và giao lại cho người bên cạnh theo chiều kim đồng hồ. Để đơn giản hoá quy định, người chơi có thể quy ước sau m lần chuyển củ khoai thì lượt chơi ngừng tạm thời. Trò chơi tiếp tục cho đến khi chỉ còn lại một người, gọi là người chiến thắng.



Hình 4. Minh hoạ trò chơi Hot Potato với 5 người chơi

Hình 4 minh hoạ trò chơi Hot Potato với 5 người chơi, bắt đầu từ An.

Yêu cầu: Hãy viết chương trình in ra màn mình tên người chiến thắng với danh sách n ($n \geq 1$) người chơi và số nguyên dương m ($m > 0$) được nhập từ bàn phím. Chạy chương trình với số người chơi n = 5.

Dữ liệu vào: danh sách tên các người chơi và số nguyên dương m ($m > 0$).

Dữ liệu ra: tên của người chiến thắng.

Hướng dẫn:

Mã giả trò chơi Hot Potato:

```
hotPotato(players, m):
  Nếu m <= 0 :
    raise ValueError('Số lần chuyển phải lớn hơn zero')
  if len(players) == 0:
    raise ValueError('Không có người chơi')
  Tạo queue rỗng
  for  p in players:            #Thêm danh sách các người chơi vào queue
    Thêm p vào queue
  player = ''                   #Tên người thắng
  while queue khác rỗng:        #Còn người chơi
    for i in range(m)           #Chuyển củ khoai m lần
      Lấy ra player từ queue
      Thêm player vào queue
    Lấy ra player từ queue      #Loại người cầm củ khoai
  return player
```

Mã nguồn tham khảo:

```
def hotPotato(players, m):
  if m <= 0 :
    raise ValueError('Số lần chuyển phải lớn hơn zero')
  if len(players) == 0:
    raise ValueError('Không có người chơi')
  queue = initQueue()           #Tạo hàng đợi rỗng
  for p in players:             #enqueue danh sách các người chơi vào queue
    enqueue(queue, p)
  player = ''
  while not isEmptyQueue(queue):#Còn người chơi
```

```
    for i in range(m):                  #Chuyển củ khoai m lần
      player = dequeue(queue)
      enqueue(queue, player)
    player = dequeue(queue)             #Loại người cầm củ khoai
  return player                         #Trả về tên người thắng

players = list(map(str, input('Nhập danh sách người chơi:').split()))
m = int(input('Nhập số lần chuyển: '))
if len(players) == 0:
  print("Không có người chơi")
elif m <= 0:
  print("Số lần chuyển > 0")
else:
  winner = hotPotato(players, m)
  print("Danh sách người chơi:", players)
  print("Số lần chuyển:", m)
  print("Người chiến thắng:", winner)
```

VẬN DỤNG

Nhiệm vụ. Kiểm tra số Palindrome

Số nguyên không âm Palindrome là số đọc xuôi hay đọc ngược vẫn chỉ cho ra một số. Chẳng hạn, các số sau đây là số Palindrome 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 22, 33, 44, 55, 66, 77, 88, 99, 101,…

Yêu cầu: Sử dụng hàng đợi và ngăn xếp để viết chương trình kiểm tra một số nguyên là Palindrome.

Dữ liệu vào: số nguyên *a*.

Dữ liệu ra: thông báo "a là số Palindrome" nếu *a* là số Palindrome, ngược lại thông báo "a không là số Palindrome".

Hướng dẫn:

Mã giả kiểm tra số Palindrome:

```
isPalindrome(a)
  Nếu a < 0
    raise ValueError('Giá trị tham số nhỏ hơn zero')
  Khởi tạo hàng đợi queue rỗng
  Khởi tạo ngăn xếp stack rỗng
  while a khác 0 :              #Lấy từng giá trị của a đưa vào queue và stack
    r = số dư a chia 10
    Thêm r vào queue
    Thêm r vào stack
    a = a chia nguyên 10
  while queue khác rỗng
    Lấy ra digit1 từ queue
    Lấy ra digit2 từ stack
    if digit1 khác digit2:
      return False
  return True
```

BÀI 1.4 ỨNG DỤNG CỦA NGĂN XẾP

MỤC TIÊU **Sau bài học này, em sẽ:**

- Trình bày và minh hoạ được một số ứng dụng ngăn xếp.
- Viết được một chương trình đơn giản để hiểu rõ ứng dụng của ngăn xếp.

KHỞI ĐỘNG

Cho biết cách trình bày cách để kiểm tra số dấu mở ngoặc và số dấu đóng ngoặc tương ứng trong chuỗi "((){[]}))" có bằng nhau hay không.

KHÁM PHÁ

1. Một số ứng dụng của ngăn xếp

Ngăn xếp với cơ chế vào sau ra trước được ứng dụng trong tác vụ hoàn tác/làm lại, lịch sử duyệt web trong các ứng dụng, giải quyết một số bài toán như tính giá trị của biểu thức số học, kiểm tra dấu ngoặc, theo dõi quá trình thực hiện của thuật toán quay lui, khử đệ quy,… Một số ứng dụng phổ biến của ngăn xếp bao gồm:

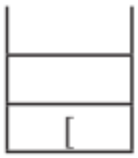
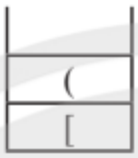
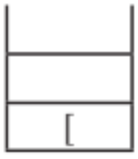
Kiểm tra tính hợp lệ của dấu ngoặc: Một chuỗi có dấu ngoặc cân bằng khi mỗi dấu mở ngoặc được theo sau bởi một dấu đóng ngoặc tương ứng được đặt ở vị trí thích hợp và ngược lại. Ví dụ, các chuỗi sau "((){[]}))", "({})([])" có dấu ngoặc cân bằng và các chuỗi sau "((()]}", "(())" có dấu ngoặc không cân bằng. Ngăn xếp có thể được sử dụng để kiểm tra các dấu ngoặc và đảm bảo tính hợp lệ của một biểu thức toán học hoặc lời gọi hàm trong ngôn ngữ lập trình. Em có thể dùng ngăn xếp để kiểm tra tính hợp lệ của chuỗi bằng cách duyệt qua từng kí tự, ở mỗi kí tự đang xét, em thực hiện như sau:

1. Nếu kí tự là dấu mở ngoặc ("(" hoặc "{" hoặc "[") thì thêm vào ngăn xếp với thao tác push.
2. Nếu kí tự là dấu đóng ngoặc (")" hoặc "}" hoặc "]") thì lấy ra kí tự ở đỉnh ngăn xếp với thao tác pop. Sau đó kiểm tra nếu kí tự được pop là dấu mở ngoặc tương ứng với dấu đóng ngoặc đang xét thì tiếp tục quá trình duyệt chuỗi, ngược lại thì trả về kết quả chuỗi có dấu ngoặc không cân bằng.
3. Bỏ qua nếu kí tự không phải là mở ngoặc hay đóng ngoặc.

Sau khi hoàn thành duyệt chuỗi, kiểm tra nếu ngăn xếp rỗng thì chuỗi có dấu ngoặc cân bằng. Ngược lại, nếu ngăn xếp không rỗng thì chuỗi có dấu ngoặc không cân bằng.

Bảng 1 minh hoạ quá trình sử dụng ngăn xếp để kiểm tra tính hợp lệ các dấu ngoặc của chuỗi [(2+3)*5].

Bảng 1. Minh hoạ quá trình sử dụng ngăn xếp để kiểm tra tính hợp lệ của dấu ngoặc

Chuỗi biểu thức	Ngăn xếp stack	Quá trình thực hiện
[(2 + 3) * 5]		Stack rỗng.
[(2 + 3) * 5]		Thêm "[" vào stack.
[(2 + 3) * 5]		Thêm "(" vào stack.
[(2 + 3) * 5]		Bỏ qua kí tự "2".
[(2 + 3) * 5]		Bỏ qua kí tự "+".
[(2 + 3) * 5]		Bỏ qua kí tự "3".
[(2 + 3) * 5]		Lấy "(" ra khỏi stack. Kiểm tra tương ứng với dấu ")".
[(2 + 3) * 5]		Bỏ qua kí tự "*".
[(2 + 3) * 5]		Bỏ qua kí tự "5".
[(2 + 3) * 5]		Lấy "[" ra khỏi stack. Kiểm tra tương ứng với dấu "]".

Tính giá trị các biểu thức số học: Ngăn xếp được sử dụng để tính giá trị các biểu thức số học bao gồm các toán hạng và toán tử, các dấu mở ngoặc và đóng ngoặc. Tính giá trị biểu thức số học gồm hai bước:

❶ Chuyển biểu thức dạng trung tố sang dạng tiền tố (toán tử nằm trước toán hạng) hoặc hậu tố (toán tử nằm sau toán hạng). Ví dụ, biểu thức dạng trung tố A + B có dạng tiền tố là +AB và dạng hậu tố là AB+. Thông thường, để chuyển từ biểu thức từ trung tố sang dạng tiền tố hay hậu tố, em cần phải phân tích các thành phần của biểu thức thành các phần tử: toán hạng, toán tử, dấu mở ngoặc, dấu đóng ngoặc,… Kết quả biểu thức dạng tiền tố hay hậu tố phải cho thấy sự tách biệt của các phần tử này (chẳng hạn, các phần tử tách nhau bởi khoảng trống). Tuy nhiên, việc phân tích các phần tử trong chuỗi biểu thức đòi hỏi một số kĩ thuật xử lí chuỗi, nên trong bài học này, em có thể giả sử các toán tử và toán hạng trong biểu thức chỉ bao gồm một kí tự. *Bảng 2* minh hoạ quá trình sử dụng ngăn xếp để chuyển biểu thức dạng trung tố 2 * (4 + 3) – 5 sang dạng hậu tố 2 4 3 + * 5 –.

Bảng 2. Minh hoạ quá trình sử dụng ngăn xếp để chuyển từ biểu thức dạng trung tố sang dạng hậu tố

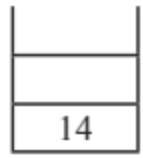
Biểu thức trung tố	Ngăn xếp stack	Biểu thức hậu tố postfix	Quá trình thực hiện
2 * (4 + 3) – 5		[]	Stack và chuỗi postfix rỗng.
2 * (4 + 3) – 5		2	Thêm "2" vào postfix.
2 * (4 + 3) – 5	*	2	Thêm "*" vào stack.
2 * (4 + 3) – 5	(*	2	Thêm "(" vào stack.
2 * (4 + 3) – 5	(*	24	Thêm "4" vào postfix.
2 * (4 + 3) – 5	+ (*	24	Thêm "+" vào stack.

Biểu thức trung tố	Ngăn xếp stack	Biểu thức hậu tố postfix	Quá trình thực hiện
2 * (4 + 3) – 5	+ (*	243	Thêm "3" vào postfix.
2 * (4 + 3) – 5	*	243 +	Gặp dấu ")", lấy các toán tử trong stack thêm vào postfix cho đến khi gặp "(", lấy "+", thêm "+" vào postfix. Lấy "(".
2 * (4 + 3) – 5	–	243 + *	Toán tử "–" có độ ưu tiên thấp hơn "*" ở đỉnh stack nên lấy "*" ra và thêm "*" vào postfix. Thêm "–" vào stack.
2 * (4 + 3) – 5	–	243 + * 5	Thêm "5" vào postfix.
2 * (4 + 3) – 5		243 + * 5 –	Đã quét hết biểu thức trung tố. Lấy tất cả toán tử trong stack thêm vào postfix: lấy "–" thêm vào postfix.

2 Sử dụng ngăn xếp để tính giá trị biểu thức dạng hậu tố. Với mỗi kí tự trong biểu thức dạng hậu tố, nếu là toán hạng thì được thêm vào ngăn xếp. Nếu là toán tử, thì hai toán hạng ở đầu ngăn xếp được lấy ra để thực hiện tính giá trị với toán tử đó, sau đó giá trị tính được sẽ được thêm vào đỉnh ngăn xếp. Quá trình này tiếp tục cho đến khi ngăn xếp chỉ còn một phần tử, cũng chính là giá trị của biểu thức. *Bảng 3* minh hoạ quá trình sử dụng ngăn xếp để tính giá trị của biểu thức dạng hậu tố 2 4 3 + * 5 –.

Bảng 3. Minh hoạ quá trình sử dụng ngăn xếp để tính giá trị của biểu thức dạng hậu tố

Phần tử xử lí	Ngăn xếp đang xét	Quá trình thực hiện
2 4 3 + * 5 –		Stack rỗng.
2 4 3 + * 5 –	3 4 2	Lần lượt thêm 2, 4, 3 vào stack.
2 4 3 + * 5 –	7 2	Lấy 3, 4 ra. Tính 4 + 3 = 7. Thêm 7 vào stack.

Phần tử xử lí	Ngăn xếp đang xét	Quá trình thực hiện
2 4 3 + * 5 –		Lấy 7, 2 ra. Tính 2 * 7 = 14. Thêm 14 vào stack.
2 4 3 + * 5 –		Thêm 5 vào stack.
2 4 3 + * 5 –		Lấy 5, 14 ra. Tính 14 – 5 = 9. Thêm 9 vào stack.
2 4 3 + * 5 –		Lấy 9 ra. Kết quả là: 9.

Em hãy minh hoạ:

a) Kiểm tra tính hợp lệ của dấu ngoặc trong chuỗi [2 * (4 + 3) – 5] bằng cách sử dụng ngăn xếp.

b) Chuyển biểu thức (1 – 4) * 2 + 7 sang dạng hậu tố bằng cách sử dụng ngăn xếp.

Một số ứng dụng phổ biến của ngăn xếp là: kiểm tra tính hợp lệ của dấu ngoặc trong biểu thức; tính giá trị biểu thức số học,… Ngăn xếp cũng được ứng dụng trong các vấn đề thực tiễn như cơ chế hoàn tác hay làm lại, lịch sử duyệt web,…

LUYỆN TẬP

Viết chương trình kiểm tra tính hợp lệ của dấu ngoặc trong chuỗi biểu thức số học.

Dữ liệu vào: chuỗi biểu thức số học.

Dữ liệu ra: nếu các dấu ngoặc cân bằng thì thông báo lên màn hình "Chuỗi có dấu ngoặc hợp lệ", ngược lại thông báo "Chuỗi có dấu ngoặc không hợp lệ".

Mã giả hàm kiểm tra dấu ngoặc hợp lệ trong chuỗi:

```
#Mã giả kiểm tra dấu ngoặc hợp lệ trong chuỗi biểu thức
checkParentheses(s)
  openParentheses = ["[","{","("]   #Danh sách các ngoặc mở
  closeParentheses = ["]","}",")"]  #Danh sách các ngoặc đóng
  khởi tạo ngăn xếp stack rỗng
  for  c in s:
    if c là là dấu mở ngoặc:
      Thêm c vào stack
    elif c là dấu đóng ngoặc:
      if stack rỗng:
        return False
```

```
      else:
        pos = vị trí của c trong closeParentheses
        c1 là dấu mở ngoặc ở vị trí pos (tương ứng với c)
        c2 là dấu mở ngoặc lấy ra từ stack
        if c1 khác c2:
          return False
  if stack rỗng:
    return True
  else:
    return False
```

Hàm kiểm tra dấu ngoặc hợp lệ trong chuỗi:

```
#Hàm kiểm tra dấu ngoặc hợp lệ trong chuỗi biểu thức
def checkParentheses(s):
  openParentheses = ["[","{","("]   #Danh sách các kí tự ngoặc mở
  closeParentheses = ["]","}",")"]  #Danh sách các kí tự ngoặc đóng
  stack = initStack()               #Khởi tạo ngăn xếp rỗng
  for c in s:                       #Duyệt mọi kí tự trong s
    if c in openParentheses:        #Kiểm tra nếu c là dấu mở ngoặc
      push(stack, c)                #push c vào stack
    elif c in closeParentheses:     #Ngược lại, nếu c là dấu đóng ngoặc
      if isEmptyStack(stack):
        return False
      else:
        pos = closeParentheses.index(c) #Lấy vị trí của kí tự ngoặc đóng
        c1 = openParentheses[pos]
        c2 = pop(stack)
        if c1 != c2:
          return False
  if isEmptyStack(stack):       #Nếu stack rỗng, dấu ngoặc hợp lệ
    return True
  else:
    return False                #Ngược lại, chuỗi có dấu ngoặc không hợp lệ
```

THỰC HÀNH

Nhiệm vụ 1. Chuyển biểu thức dạng trung tố sang hậu tố

Yêu cầu: Cho biểu thức 2 * (4 + 3) – 5. Hãy viết chương trình để chuyển một biểu thức dạng trung tố sang biểu thức dạng hậu tố với giả sử toán hạng, toán tử trong chuỗi chỉ gồm 1 kí tự. Chạy chương trình trên.

Dữ liệu vào: chuỗi biểu thức dạng trung tố.

Dữ liệu ra: chuỗi biểu thức dạng hậu tố.

Hướng dẫn:

Sử dụng hàm isOperator() để kiểm tra kí tự là toán tử, hàm getPriority() để xác định độ ưu tiên của toán tử, hàm inFixtoPostfix() để chuyển biểu thức dạng trung tố sang dạng hậu tố.

Hàm kiểm tra kí tự có phải là toán tử hay không:

```
def isOperator(c):
    return c in '+-*/^'
```

Hàm kiểm tra độ ưu tiên của toán tử:

```
def getPriority(op):
    if op == '+' or op == '-':
```

```
        return 1
    elif op == '*' or op == '/':
        return 2
    elif op == '^':
        return 3
    else:
        return 0
```

Hàm chuyển biểu thức dạng trung tố sang dạng hậu tố:

```
#Hàm chuyển biểu thức trung tố sang hậu tố
def inFixtoPostfix(infix):
    infix = infix.replace('[', '(').replace(']', ')').replace('{', '(').replace('}', ')')
    stack = initStack()
    postfix = ""
    for c in infix:
        if c == ' ':          #Bỏ qua khoảng trắng
            continue
        #Nếu kí tự là toán hạng, thêm vào chuỗi hậu tố
        if c.isalpha() or c.isdigit():
            postfix += c
        #Nếu kí tự là '(', thêm vào stack
        elif c == '(':
            push(stack, c)
        #Nếu kí tự là ')'
        elif c == ')':  #pop khỏi stack và thêm vào postfix cho đến khi gặp (
            while not isEmptyStack(stack) and top(stack) != '(':
                postfix += pop(stack)
            pop(stack)
        elif isOperator(c):    #Toán tử
            if c != '^':
                while not isEmptyStack(stack) and getPriority(c) <= getPriority(top(stack)):
                    postfix += pop(stack)
            push(stack, c)
    while not isEmptyStack(stack):
        postfix += pop(stack)
    return postfix
infix = '2 * (4 + 3) - 5'
postfix = inFixtoPostfix(infix)
print("Biểu thức hậu tố: ", postfix)
infix = '2^2^3'
postfix = inFixtoPostfix(infix)
print("Biểu thức hậu tố: ", postfix)
```

Nhiệm vụ 2. Viết chương trình tính giá trị của biểu thức số học với biểu thức đầu vào là biểu thức hậu tố

Yêu cầu: Viết chương trình tính giá trị biểu thức số học dùng hàm chuyển biểu thức inFixtoPostfix(). Chạy chương trình trên.

Hướng dẫn:

1. Sử dụng inFixtoPostfix() để chuyển biểu thức dạng trung tố sang dạng hậu tố.
2. Sử dụng hàm calPostfix() để tính giá trị biểu thức dạng hậu tố.

Hàm tính b op a, với a, b là toán hạng, op là toán tử:

```
def calculate(a, b, op):
  if op == '+':
    return b+a
  elif op == '-':
    return b-a
  elif op == '*':
    return b*a
  elif op == '/':
    return b/a
  else:
    return b**a
```

Hàm tính giá trị biểu thức dạng hậu tố:

```
def calPostfix(s):
    stack = initStack()
    for op in s:
      if not isOperator(op):
        push(stack, op)           #Thêm toán hạng vào stack
      else:                       #Kiểm tra nếu op là toán tử
        a = int(pop(stack))       #Lấy toán hạng a ra khỏi stack
        b = int(pop(stack))       #Lấy toán hạng b ra khỏi stack
        d = calculate(a, b, op)   #Gọi calculate để thực hiện phép tính b op a
        push(stack, d)            #Thêm d vào stack
    k = pop(stack)                #Lấy phần tử còn lại trong stack
    return k                      #Trả về kết quả

k = calPostfix('245-+6*')
print(k)
k = calPostfix('243+*5-')
print(k)
```

VẬN DỤNG

Nhiệm vụ. Chuyển biểu thức dạng trung tố sang tiền tố

Yêu cầu: Viết chương trình chuyển một biểu thức dạng trung tố inFix sang dạng tiền tố preFix.

Dữ liệu vào: chuỗi biểu thức dạng trung tố.

Dữ liệu ra: chuỗi biểu thức dạng tiền tố.

Ví dụ:

Biểu thức dạng trung tố	Biểu thức dạng tiền tố
2 * 5 + 7/9	+ * 25/79
(2 – 7/3) * (2/8 – 4)	* – 2/73 – /284

Hướng dẫn:

1. Sử dụng lệnh inFix = inFix[::–1] để đảo ngược chuỗi biểu thức trung tố. Ví dụ, "(2 – 7/3) * (2/8 – 4)" được chuyển thành ")4–8/2(*)3/7–2(".
2. Thay thế các dấu đóng ngoặc thành dấu mở ngoặc và dấu mở ngoặc thành dấu đóng ngoặc trong chuỗi biểu thức đã đảo ngược trong bước 1.
   Ví dụ, chuỗi ")4–8/2(*)3/7–2(" được chuyển thành "(4 – 8/2) * (3/7 – 2)".
3. Gọi hàm inFixtoPostfix() chuyển từ biểu thức dạng trung tố sang dạng hậu tố.
4. Đảo ngược chuỗi biểu thức kết quả của 3.
5. Kết thúc.

Chuyên đề 2

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

BÀI 2.1 CÂY VÀ CÂY NHỊ PHÂN

MỤC TIÊU **Sau bài học này, em sẽ:**

- Nêu được khái niệm cây, cây nhị phân.
- Biểu diễn được cây nhị phân bằng mảng một chiều.

KHỞI ĐỘNG

Quan sát sơ đồ tổ chức của một công ty ở *Hình 1* và cho biết các phòng ban chức năng trong công ty được tổ chức như thế nào.

Hình 1. Sơ đồ tổ chức của một công ty theo chức năng

KHÁM PHÁ

1. Giới thiệu cây, cây nhị phân

a) Cây

Cấu trúc dữ liệu tuyến tính bao gồm các phần tử tạo thành một dãy nối tiếp nhau và thông thường thao tác duyệt được thực hiện tuần tự theo trình tự của các phần tử. Mỗi phần tử có nhiều nhất một phần tử kế đi sau nó. Trong khi đó, cấu trúc dữ liệu phi tuyến tính bao gồm các phần tử không tạo thành một dãy nối tiếp nhau. Mỗi phần tử có thể có nhiều phần tử kế đi sau nó. Chẳng hạn, cấu trúc dữ liệu cây là cấu trúc dữ liệu phi tuyến.

Cây là một tập hợp rỗng hoặc bao gồm nhiều phần tử được gọi là **nút**. **Cây rỗng** là cây không có nút nào. Ngược lại, các nút có liên kết "cha – con" với nhau: mỗi **nút cha** có thể có nhiều **nút con** và mỗi nút con chỉ có một nút cha; một cây chỉ có một **nút gốc** là nút không có nút cha.

Biểu diễn nút cha nằm ở trên và nút con nằm ở dưới, chúng kết nối với nhau bằng một cạnh. Một nút và các nút nằm ở phía dưới nút này liên kết "cha – con" với nhau tạo thành **cây con** có nút gốc là nút này. Điều này cho thấy cây có cấu trúc đệ quy: một cây có thể chứa các cây con.

Một số khái niệm cơ bản về cây:

Nút gốc là nút đầu tiên của cây, thường là điểm xuất phát cho việc truy cập và duyệt cây. Tất cả các nút khác trong cây sẽ nằm dưới nút gốc theo một cấu trúc phân nhánh, gọi là quan hệ "cha – con". Mỗi nút trong cây có thể không có hoặc có một số **nút con**. Một nút có nút con được gọi là **nút cha**. Nút gốc (của cây) không có nút cha. Các nút còn lại chỉ có một nút cha. Nút cha và nút con được kết nối với nhau bằng một cạnh trong hình vẽ của cây.

Nút lá là nút không có nút con. Nếu cây chỉ có một nút thì nút gốc cũng là nút lá.

Nút nhánh (nút trong) là nút có ít nhất một nút con. Nếu cây có nhiều hơn một nút thì nút gốc cũng là nút trong.

Mức của một nút: Mức của nút gốc bằng 0 (quy ước). Mức của một nút (khác nút gốc) bằng mức của nút cha cộng 1.

Chiều cao của cây là mức lớn nhất của các nút lá. Cây rỗng có chiều cao là –1.

Đường đi từ nút gốc đến nút x là dãy các cạnh từ nút gốc đến nút x.

Hình 2 minh hoạ các khái niệm cơ bản về cây.

Hình 2. Minh hoạ các khái niệm trên cây

Trong *Hình 2*,

Cây gốc 14 có **nút gốc** 14, nút này có ba **cây con** là cây gốc 7, cây gốc 23 và cây gốc 8. Cây gốc 8 có nút gốc 8, nút này có hai cây con là cây gốc 2 và cây gốc 27.

Nút 14 là **nút cha** của nút 7 và nút 7 là **nút con** của nút 14.

Nút 24 là **nút lá** vì nó không có các nút con.

Nút 8 là **nút trong** vì nó có các nút con (là nút 2 và nút 27).

Mức của nút gốc là 0 (theo quy ước), **mức** của nút 14 là 0, **mức** của nút 7 là 1, **mức** của nút 24 là 2.

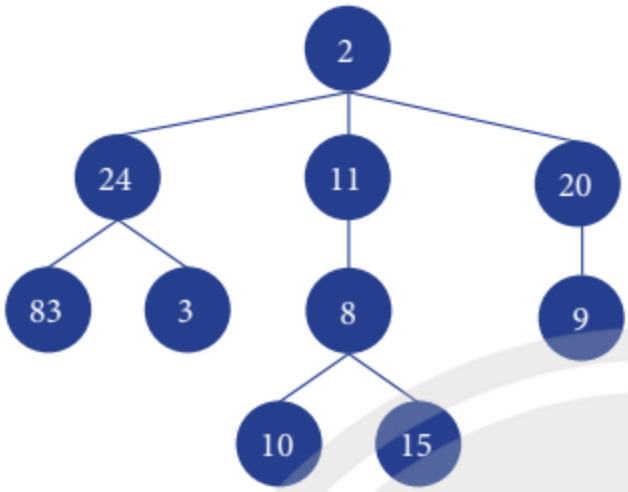
Cây gốc 14 có **chiều cao** là 2.

Đường đi từ nút 14 đến nút 26 là cạnh (14-7) và cạnh (7-26), kí hiệu là [14, 7, 26].

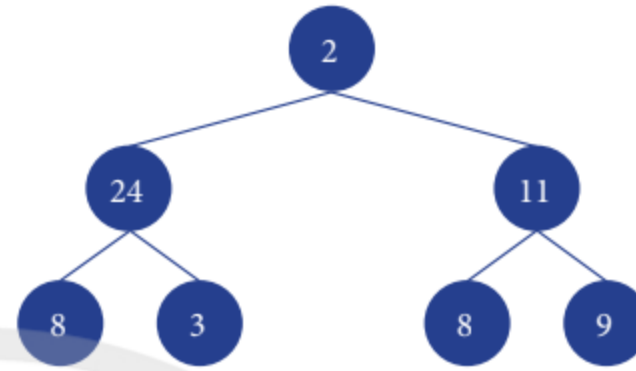
Cho các cây như ở *Hình 3*. Cho biết:

a) Nút gốc, nút nhánh, nút lá ở *Hình 3a*.

b) Mức và chiều cao của cây trong *Hình 3b*.



Hình 3a.



Hình 3b.

Hình 3. Biểu diễn cây

Cây là một tập hợp rỗng hoặc bao gồm nhiều phần tử được gọi là nút. Cây rỗng là cây không có nút nào. Ngược lại, các nút có liên kết "cha – con" với nhau: mỗi nút cha có thể có nhiều nút con và mỗi nút con chỉ có một nút cha; một cây chỉ có một nút gốc là nút không có nút cha.

b) Cây nhị phân

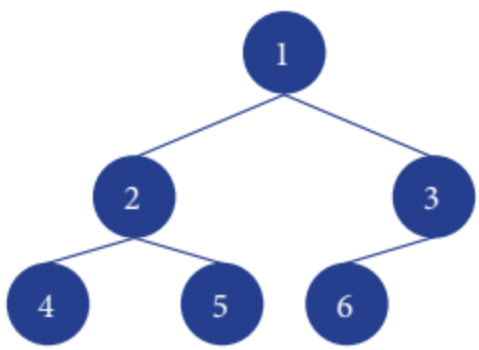
Cây nhị phân là cây mà mỗi nút của cây có tối đa hai cây con (cây con trái và cây con phải).

Một số dạng đặc biệt của cây nhị phân:

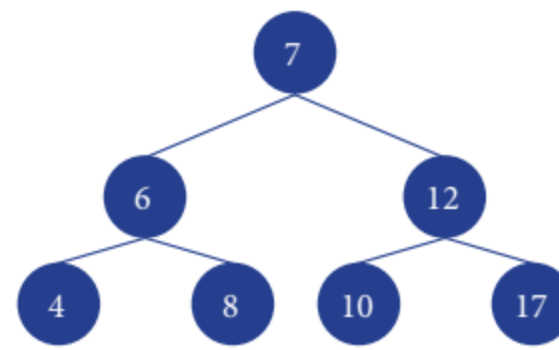
Gọi h là chiều cao của cây nhị phân.

Cây nhị phân hoàn chỉnh (*Hình 4a*) là cây nhị phân mà tại mức i ($0 \le i < h$) có 2^i nút và tại mức h thì các nút được điền từ trái qua phải.

Cây nhị phân hoàn hảo (*Hình 4b*) là cây nhị phân mà tại mức i ($0 \le i \le h$) có 2^i nút.



Hình 4a. Cây nhị phân hoàn chỉnh



Hình 4b. Cây nhị phân hoàn hảo

Hình 4. Minh hoạ các dạng đặc biệt của cây nhị phân

Một số tính chất của cây nhị phân:

Tính chất 1: Số nút của mức i là $n_i \le 2^i$ $(0 \le i \le h)$.

Tính chất 2: Số nút lá là $n_{lá} \le 2^h$.

Tính chất 3: Số nút trong là $n_{trong} \le 2^h - 1$.

Tính chất 4: Số nút của cây là $n \le 2^{h+1} - 1$.

Tính chất 5: $h \ge \log_2(n + 1) - 1$ (suy ra từ Tính chất 4).

Cho biết cây nào ở *Hình 3* là cây nhị phân.

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

2. Biểu diễn cây nhị phân

a) Biểu diễn cây nhị phân bằng mảng một chiều

Cây nhị phân có thể được biểu diễn bằng mảng một chiều bằng cách đánh chỉ số cho các nút của cây bắt đầu từ nút gốc.

Theo Tính chất 4 ở mục 1b, cây nhị phân có thể được lưu trữ trong mảng một chiều có $(2^{h+1} - 1)$ phần tử như sau:

– Nếu cây rỗng thì phần tử có chỉ số 0 là None.

– Nút gốc có chỉ số 0.

– Nếu nút có chỉ số i thì nút con trái có chỉ số là 2i + 1 và nút con phải có chỉ số là 2i + 2.

– Nút có chỉ số i được lưu vào phần tử có chỉ số i.

– Nếu nút có chỉ số i không có nút con trái thì phần tử có chỉ số 2i + 1 là None.

– Nếu nút có chỉ số i không có nút con phải thì phần tử có chỉ số 2i + 2 là None.

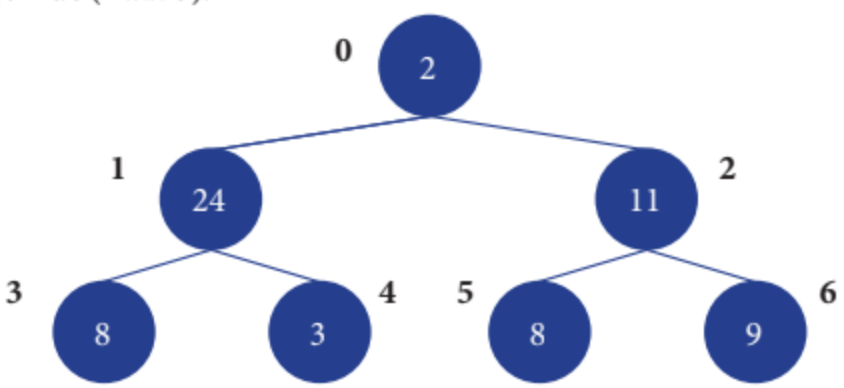
– Các phần tử còn lại là None.

– Không lưu các giá trị None ở cuối mảng.

Điều này có nghĩa là thêm các nút None vào cây nhị phân tổng quát để cây này trở thành cây nhị phân hoàn chỉnh.

Ví dụ 1: Để biểu diễn cây nhị phân ở *Hình 3b* (cây nhị phân hoàn hảo) bằng mảng một chiều, em thực hiện như sau:

1 Đánh chỉ số bắt đầu từ 0 cho các nút theo thứ tự từ mức 0 đến mức cuối cùng, từ trái sang phải trên cùng một mức (*Hình 5*).



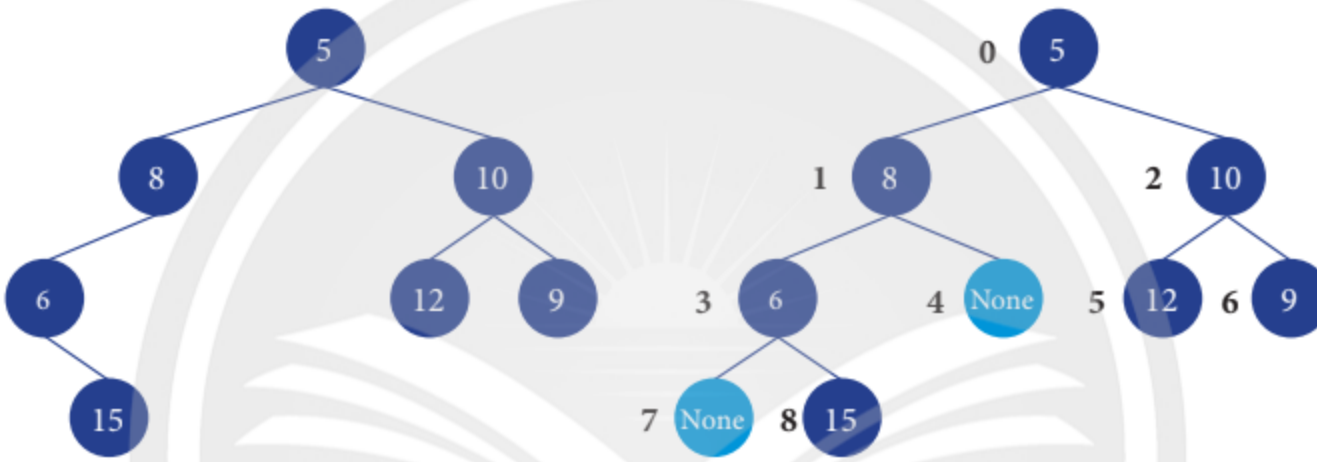
Hình 5. Đánh chỉ số cho các nút

❷ Lưu các nút vào mảng một chiều: nút có chỉ số i được lưu vào phần tử có chỉ số i.

Chỉ số của phần tử	0	1	2	3	4	5	6
Giá trị nút	2	24	11	8	3	8	9

Ví dụ 2: Với cây nhị phân tổng quát, em sẽ thêm vào một số nút giả và gán giá trị None để được cây nhị phân hoàn chỉnh (*Hình 6*), đưa các nút cây vào mảng một chiều vào đúng vị trí theo thứ tự vừa được đánh số.

Để tiết kiệm vùng nhớ của mảng một chiều, em không lưu trữ các nút giả None vào cuối mảng. Chẳng hạn, trong *Hình 6b*, nút 12 và nút 9 là các nút lá; các nút con của các nút này là các nút giả None và không được lưu trữ vào cuối mảng.



Hình 6a. Cây chưa có nút giả *Hình 6b. Thêm nút giả vào cây*

Hình 6. Thêm nút giả mang giá trị None cho cây

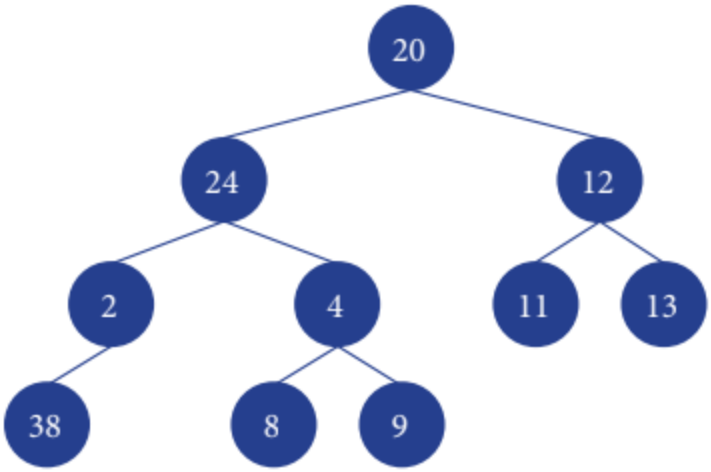
b) Cài đặt cây nhị phân bằng mảng một chiều

Giả sử nút thật có giá trị là số dương và nút giả có giá trị là 0. Dữ liệu nhập để tạo cây nhị phân tổng quát trong *Hình 6b* là dãy các số 5, 8, 10, 6, 0, 12, 9, 0, 15. Chương trình sau đây sẽ tạo cây nhị phân hoàn chỉnh từ cây nhị phân tổng quát sau khi thêm vào các nút giả.

Chương trình tham khảo:

```
01. def createTree(tree, a):
02.   for i in range(len(a)):
03.     if a[i] != 0:
04.       tree.append(a[i]) #Thêm a[i] vào tree
05.     else:
06.       tree.append(None) #Thêm nút giả None vào tree
07. print(" Dãy các số để tạo cây: ")
08. a = list(map(int,input().split()))
09. tree = []                #Tạo cây tree là rỗng
10. createTree(tree, a)      #Thêm mảng a vào cây
11. print(tree)
```

Xây dựng mảng một chiều biểu diễn cây ở *Hình 7*.



Hình 7. Cây nhị phân

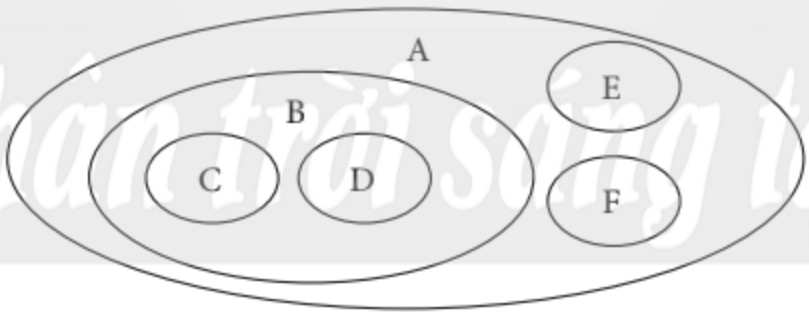
Cách lưu trữ các nút của cây vào các phần tử của mảng một chiều như sau:

Nút gốc là nút có chỉ số 0. Nếu nút có chỉ số i thì nút con trái có chỉ số là 2*i + 1 và nút con phải có chỉ số là 2*i + 2.

Cây rỗng thì phần tử có chỉ số 0 là None. Nút có chỉ số i được lưu vào phần tử có chỉ số i. Nếu nút có chỉ số i không có nút con trái thì phần tử có chỉ số 2*i + 1 là None. Nếu nút có chỉ số i không có nút con phải thì phần tử có chỉ số 2*i + 2 là None. Các phần tử còn lại là None. Không lưu các giá trị None ở cuối mảng. Điều này có nghĩa là thêm các nút None vào cây nhị phân tổng quát để cây này trở thành cây nhị phân hoàn chỉnh.

LUYỆN TẬP

1. Biểu diễn cấu trúc các tập bao lồng nhau trong *Hình 8* dưới dạng cây. Cho biết cây vừa nêu có mấy nút lá.



Hình 8. Mô hình biểu diễn các tập bao lồng nhau A, B, C, D, E, F

2. Hãy vẽ một cây nhị phân có chiều cao là 4; sau đó, xây dựng mảng một chiều để biểu diễn cây này.

VẬN DỤNG

Cho mảng một chiều A biểu diễn cho một cây nhị phân hoàn chỉnh như sau:

2	24	11	8	3	8	9	1	2	3	4

Chỉ ra các nút con của nút có giá trị là 11.

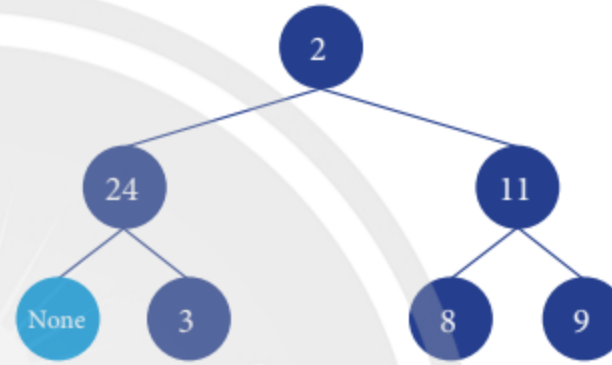
BÀI 2.2 CÁC PHÉP TOÁN DUYỆT CÂY NHỊ PHÂN

MỤC TIÊU **Sau bài học này, em sẽ:**

- Trình bày và mô phỏng các phép toán duyệt trước, duyệt giữa và duyệt sau cây nhị phân.
- Viết được chương trình duyệt cây nhị phân.

KHỞI ĐỘNG

Cho cây nhị phân như *Hình 1*. Hãy dùng mảng một chiều để biểu diễn các giá trị trong cây nhị phân.



Hình 1. Cây nhị phân

KHÁM PHÁ

1. Các phép toán duyệt cây nhị phân

Duyệt cây nhị phân là quá trình thăm tất cả nút và mọi nút chỉ được thăm đúng một lần. Tuỳ theo quá trình thăm tất cả nút mà thứ tự xử lí của các nút là khác nhau. *Hình 2* trình bày cây nhị phân gốc i: nút gốc i, cây con trái và cây con phải.

Hình 2. Cây nhị phân gốc i

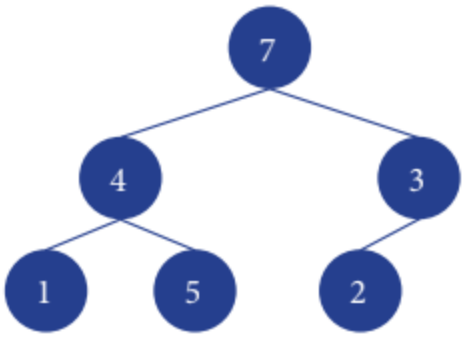
Tuỳ theo thứ tự giải quyết: xử lí nút i (gốc), duyệt cây con trái (trái), duyệt cây con phải (phải), có ba phép toán duyệt cây nhị phân như sau:

Duyệt trước (gốc – trái – phải): xử lí nút gốc i (gốc), duyệt cây con trái (trái), duyệt cây con phải (phải).

Duyệt giữa (trái – gốc – phải): duyệt cây con trái (trái), xử lí nút gốc i (gốc), duyệt cây con phải (phải).

Duyệt sau (trái – phải – gốc): duyệt cây con trái (trái), duyệt cây con phải (phải), xử lí nút gốc i (gốc).

Cho cây nhị phân ở *Hình 3*.



Hình 3. Cây nhị phân

a) Duyệt trước (gốc – trái – phải)

Thứ tự của phép toán duyệt trước cây gốc i là:

1. Xử lí nút gốc i (gốc).
2. Duyệt cây con trái (trái).
3. Duyệt cây con phải (phải).

Các bước thực hiện phép toán duyệt giữa cây nhị phân ở *Hình 3* được minh hoạ như sau:

1. i = 0, xử lí nút 7.
2. Duyệt cây con trái của nút 7: là duyệt cây gốc 4.
   - 2a Xử lí nút 4.
   - 2b Duyệt cây con trái của nút 4: là duyệt cây gốc 1.
     - Xử lí nút 1.
     - Duyệt cây con trái của nút 1: không tồn tại trong cây nên bỏ qua.
     - Duyệt cây con phải của nút 1: không tồn tại trong cây nên bỏ qua.
   - 2c Duyệt cây con phải của nút 4: là duyệt cây gốc 5.
     - Xử lí nút 5.
     - Duyệt cây con trái của nút 5: không tồn tại trong cây nên bỏ qua.
     - Duyệt cây con phải của nút 5: không tồn tại trong cây nên bỏ qua.
3. Duyệt cây con phải của nút 7: là duyệt cây gốc 3.
   - 3a Xử lí nút 3.
   - 3b Duyệt cây con trái của nút 3: là duyệt cây gốc 2.
     - Xử lí nút 2.
     - Duyệt cây con trái của nút 2: không tồn tại trong cây nên bỏ qua.
     - Duyệt cây con phải của nút 2: không tồn tại trong cây nên bỏ qua.
   - 3c Duyệt cây con phải của nút 3: không tồn tại trong cây nên bỏ qua.

Kết quả thực hiện phép toán duyệt trước cây nhị phân ở *Hình 3* là:

7	4	1	5	3	2

b) Duyệt giữa (trái – gốc – phải)

Thứ tự của phép toán duyệt giữa cây gốc i là:

1. Duyệt cây con trái (trái).
2. Xử lí nút gốc i (gốc).
3. Duyệt cây con phải (phải).

Các bước thực hiện phép toán duyệt giữa cây nhị phân của *Hình 3* được minh hoạ như sau.

1. Duyệt cây con trái của nút 7: là duyệt cây gốc 4.
   - 1a Duyệt cây con trái của nút 4: là duyệt cây gốc 1.
   - – Duyệt cây con trái của nút 1: không tồn tại trong cây nên bỏ qua.
   - – Xử lí nút 1.
   - – Duyệt cây con phải của nút 1: không tồn tại trong cây nên bỏ qua.
   - 1b Xử lí nút 4.
   - 1c Duyệt cây con phải của nút 4: là duyệt cây gốc 5.
   - – Duyệt cây con trái của nút 5: không tồn tại trong cây nên bỏ qua.
   - – Xử lí nút 5.
   - – Duyệt cây con phải của nút 5: không tồn tại trong cây nên bỏ qua.
2. Xử lí nút 7.
3. Duyệt cây con phải của nút 7: là duyệt cây gốc 3.
   - 3a Duyệt cây con trái của nút 3: là duyệt cây gốc 2.
   - – Duyệt cây con trái của nút 2: không tồn tại trong cây nên bỏ qua.
   - – Xử lí nút 2.
   - – Duyệt cây con phải của nút 2: không tồn tại trong cây nên bỏ qua.
   - 3b Xử lí nút 3.
   - 3c Duyệt cây con phải của nút 3: không tồn tại trong cây nên bỏ qua.

Kết quả thực hiện phép toán duyệt giữa cây nhị phân ở *Hình 3* là:

1	4	5	7	2	3

c) Duyệt sau (trái – phải – gốc)

Thứ tự của phép toán duyệt sau cây gốc i là:

1. Duyệt cây con trái (trái).
2. Duyệt cây con phải (phải).
3. Xử lí nút gốc i (gốc).

Các bước thực hiện phép toán duyệt sau cây nhị phân của *Hình 3* được minh hoạ như sau.

1. Duyệt cây con trái của nút 7: là duyệt cây gốc 4.
   - 1a Duyệt cây con trái của nút 4: là duyệt cây gốc 1.
   - – Duyệt cây con trái của nút 1: không tồn tại trong cây nên bỏ qua.
   - – Duyệt cây con phải của nút 1: không tồn tại trong cây nên bỏ qua.
   - – Xử lí nút 1.

1b Duyệt cây con phải của nút 4: là duyệt cây gốc 5.

– Duyệt cây con trái của nút 5: không tồn tại trong cây nên bỏ qua.

– Duyệt cây con phải của nút 5: không tồn tại trong cây nên bỏ qua.

– Xử lí nút 5.

1c Xử lí nút 4.

2 Duyệt cây con phải của nút 7: là duyệt cây gốc 3.

2a Duyệt cây con trái của nút 3: là duyệt cây gốc 2.

– Duyệt cây con trái của nút 2: không tồn tại trong cây nên bỏ qua.

– Duyệt cây con phải của nút 2: không tồn tại trong cây nên bỏ qua.

– Xử lí nút 2.

2b Duyệt cây con phải của nút 3: không tồn tại trong cây nên bỏ qua.

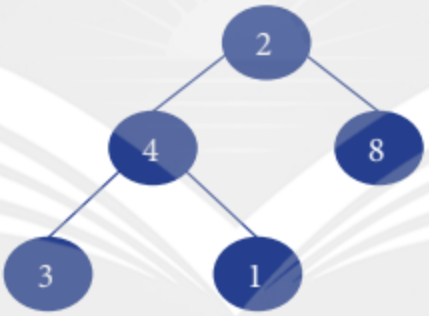
2c Xử lí nút 3.

3 Xử lí nút 7.

Kết quả thực hiện phép toán duyệt sau cây nhị phân ở *Hình 3* là:

1	5	4	2	3	7

Cho cây nhị phân như *Hình 4*.



Hình 4. Cây nhị phân

Kết quả thực hiện phép toán duyệt cây nhị phân như sau:

3	4	1	2	8

Phép toán duyệt cây nhị phân cho kết quả như bảng ở trên là phép toán nào?

2. Cài đặt các phép toán duyệt cây nhị phân

a) Duyệt trước (gốc – trái – phải)

Với cây nhị phân ở *Hình 1*, dữ liệu nhập là dãy số 2 24 11 0 3 8 9 và kết quả duyệt trước là 2, 24, 3, 11, 8, 9.

2	24	3	11	8	9

Thuật toán:

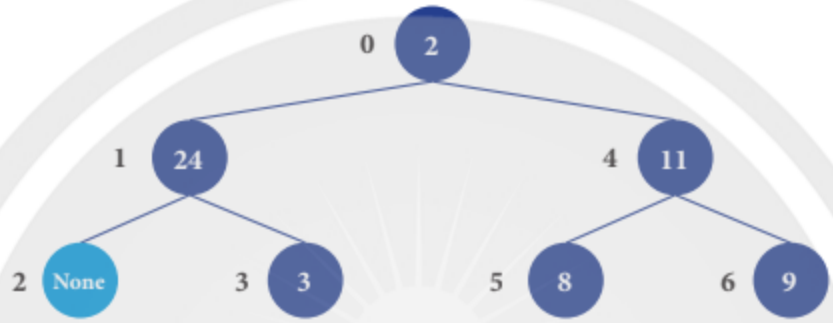
```
01. preOrderTraversal(cây gốc i):
02.   if cây gốc i khác rỗng:
03.     xử lí nút i
```

```
    preOrderTraversal(cây con trái của nút i)
    preOrderTraversal(cây con phải của nút i)
#Phép toán duyệt trước cây ban đầu:
preOrderTraversal(cây gốc 0)
```

Cài đặt thuật toán duyệt trước bằng hàm đệ quy:

```
def preOrderTraversal(tree, i):
  if i < len(tree) and tree[i] != None: #Cây gốc i khác rỗng
    print(tree[i],end = ' ')            #Xử lí nút gốc
    preOrderTraversal(tree, 2*i + 1)    #Duyệt cây con trái
    preOrderTraversal(tree, 2*i + 2)    #Duyệt cây con phải
#Lệnh gọi hàm:
preOrderTraversal(tree, 0)
```

Hình 5 biểu diễn quá trình duyệt trước cây nhị phân theo thứ tự duyệt được đánh số từ 0 đến 6.



Hình 5. Quá trình duyệt trước cây nhị phân

Chương trình tham khảo:

```
def preOrderTraversal(tree, i):
  if i < len(tree) and tree[i] != None: #Cây gốc i khác rỗng
    print(tree[i],end = ' ')            #Xử lí nút gốc
    preOrderTraversal(tree, 2*i + 1)    #Duyệt cây con trái
    preOrderTraversal(tree, 2*i + 2)    #Duyệt cây con phải
def createTree(tree, a):
    for i in range(len(a)):
      if a[i] != 0:
        tree.append(a[i])
      else:
        tree.append(None)
print("Nhập dãy các số để tạo cây:")
a = list(map(int,input().split()))
tree = []
createTree(tree, a)
#Lệnh gọi hàm:
preOrderTraversal(tree, 0)              #Duyệt trước cây tree
```

b) Duyệt giữa (trái – gốc – phải)

Với cây nhị phân ở *Hình 1*, dữ liệu nhập là dãy số 2 24 11 0 3 8 9 và kết quả duyệt giữa là 24, 3, 2, 8, 11, 9.

24	3	2	8	11	9

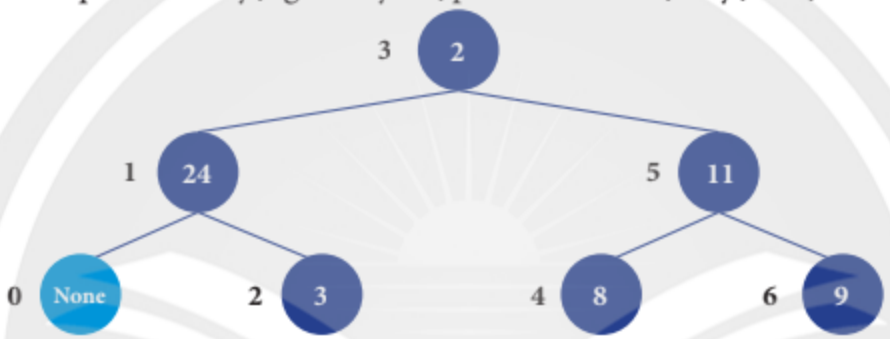
Thuật toán:

```
01. inOrderTraversal(cây gốc i):
02.   if cây gốc i khác rỗng:
03.     inOrderTraversal(cây con trái của nút i)
04.     xử lí nút i
05.     inOrderTraversal(cây con phải của nút i)
06. #Phép toán duyệt giữa cây ban đầu:
07. inOrderTraversal(cây gốc 0)
```

Cài đặt thuật toán duyệt giữa bằng hàm đệ quy:

```
01. def inOrderTraversal(tree, i):
02.   if i < len(tree) and tree[i] != None:  #Cây gốc i khác rỗng
03.     inOrderTraversal(tree, 2*i + 1)      #Duyệt cây con trái
04.     print(tree[i],end = ' ')             #Xử lí nút gốc
05.     inOrderTraversal(tree, 2*i + 2)      #Duyệt cây con phải
06. #Lệnh gọi hàm:
07. inOrderTraversal(tree, 0)
```

Hình 6 biểu diễn quá trình duyệt giữa cây nhị phân theo thứ tự duyệt được đánh số từ 0 đến 6.



Hình 6. Quá trình duyệt giữa cây nhị phân

Chương trình tham khảo:

```
01. def inOrderTraversal(tree, i):
02.   if i < len(tree) and tree[i] != None:  #Cây gốc i khác rỗng
03.     inOrderTraversal(tree, 2*i + 1)      #Duyệt cây con trái
04.     print(tree[i],end = ' ')             #Xử lí nút gốc
05.     inOrderTraversal(tree, 2*i + 2)      #Duyệt cây con phải
06. def createTree(tree, a):
07.   for i in range(len(a)):
08.     if a[i] != 0:
09.       tree.append(a[i])
10.     else:
11.       tree.append(None)
12. print("Nhập dãy các số để tạo cây:")
13. a = list(map(int,input().split()))
14. tree = []
15. createTree(tree, a)
16. #Lệnh gọi hàm:
17. inOrderTraversal(tree, 0)                #Duyệt giữa cây tree
```

c) Duyệt sau (trái – phải – gốc)

Với cây nhị phân ở *Hình 1*, dữ liệu nhập là dãy số 2 24 11 0 3 8 9 và kết quả duyệt giữa là 3, 24, 8, 9, 11, 2.

3	24	8	9	11	2

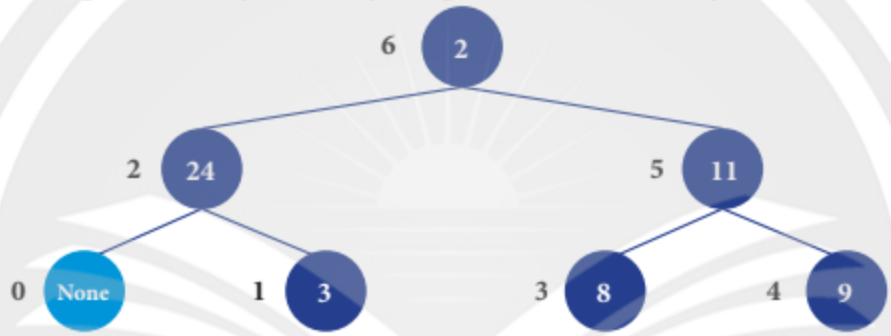
Thuật toán:

```
01. postOrderTraversal(cây gốc i):
02.   if cây gốc i khác rỗng:
03.     postOrderTraversal (cây con trái của nút i)
04.     postOrderTraversal (cây con phải của nút i)
05.     xử lí nút i
06. #Phép toán duyệt sau cây ban đầu:
07. postOrderTraversal (cây gốc 0)
```

Cài đặt thuật toán duyệt sau bằng hàm đệ quy:

```
01. def postOrderTraversal(tree, i):
02.   if i < len(tree) and tree[i] != None:  #Cây gốc i khác rỗng
03.     postOrderTraversal(tree, 2*i + 1)    #Duyệt cây con trái
04.     postOrderTraversal(tree, 2*i + 2)    #Duyệt cây con phải
05.     print(tree[i],end = ' ')             #Xử lí nút gốc
06. #Lệnh gọi hàm:
07. postOrderTraversal(tree, 0)
```

Hình 7 biểu diễn quá trình duyệt sau cây nhị phân theo thứ tự duyệt được đánh số từ 0 đến 6.



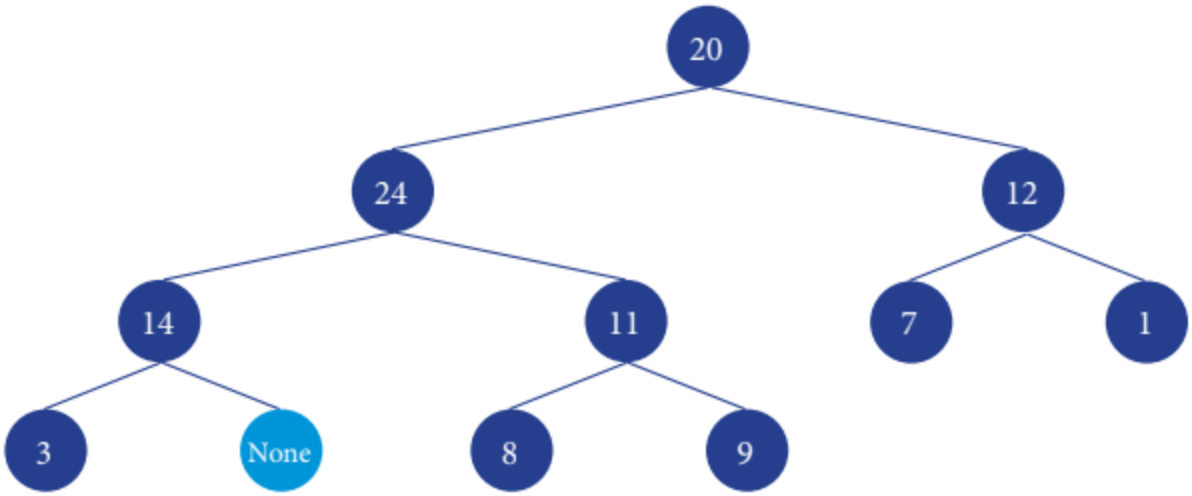
Hình 7. Quá trình duyệt sau cây nhị phân

Chương trình tham khảo:

```
01. def postOrderTraversal(tree, i):
02.   if i < len(tree) and tree[i] != None:  #Cây gốc i khác rỗng
03.     postOrderTraversal(tree, 2*i + 1)    #Duyệt cây con trái
04.     postOrderTraversal(tree, 2*i + 2)    #Duyệt cây con phải
05.     print(tree[i],end = ' ')             #Xử lí nút gốc
06. def createTree(tree, a):
07.   for i in range(len(a)):
08.     if a[i] != 0:
09.       tree.append(a[i])
10.     else:
11.       tree.append(None)
12. print("Nhập dãy các số để tạo cây:")
13. a = list(map(int,input().split()))
14. tree = []
15. createTree(tree, a)
16. #Lệnh gọi hàm:
17. postOrderTraversal(tree, 0)              #Duyệt sau cây tree
```

Cho cây nhị phân như *Hình 8*. Biểu diễn các giá trị trong cây nhị phân bằng mảng một chiều theo:

a) Duyệt trước; b) Duyệt giữa; c) Duyệt sau.



Hình 8. Cây nhị phân

- Ba phép toán duyệt cây nhị phân được sử dụng phổ biến: duyệt trước, duyệt giữa và duyệt sau. Các phép toán này khác nhau chủ yếu ở thứ tự duyệt các nút gốc của các cây con.
- Trong phép toán duyệt trước, nút gốc được duyệt đầu tiên, sau đó duyệt cây con trái và cuối cùng duyệt cây con phải.
- Trong phép toán duyệt giữa, cây con trái được duyệt đầu tiên, sau đó duyệt nút gốc và cuối cùng duyệt cây con phải.
- Trong phép toán duyệt sau, cây con trái được duyệt đầu tiên, sau đó duyệt cây con phải và cuối cùng duyệt nút gốc.

LUYỆN TẬP

Cho các thao tác: (1) Duyệt nút gốc; (2) Duyệt cây con trái; (3) Duyệt cây con phải. Sắp xếp thứ tự các thao tác tương ứng với các phép toán duyệt cây nhị phân:

a) Duyệt trước; b) Duyệt giữa; c) Duyệt sau.

THỰC HÀNH

Cho mảng số nguyên dương A = [5, 8, 7, 4, 9, 2].

a) Xây dựng cây nhị phân với mảng số nguyên dương trên.

b) Sử dụng phép toán duyệt trước, duyệt giữa, duyệt sau để xuất thứ tự các giá trị trên cây nhị phân được xây dựng ở câu a).

VẬN DỤNG

Cho mảng các số nguyên dương A = [9, 6, 5, 17, 10, 3, 8, 12].

a) Xây dựng cây nhị phân với mảng số nguyên dương trên.

b) Viết chương trình có sử dụng phép toán duyệt trước, duyệt giữa, duyệt sau để:

– Kiểm tra giá trị 10 có trong cây hay không?

– Kiểm tra giá trị 7 có trong cây hay không?

BÀI 2.3 CÂY TÌM KIẾM NHỊ PHÂN

MỤC TIÊU **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 biểu diễn một tập số nguyên dương và thuật toán xác định một giá trị đã cho có thuộc tập hợp đó hay không.

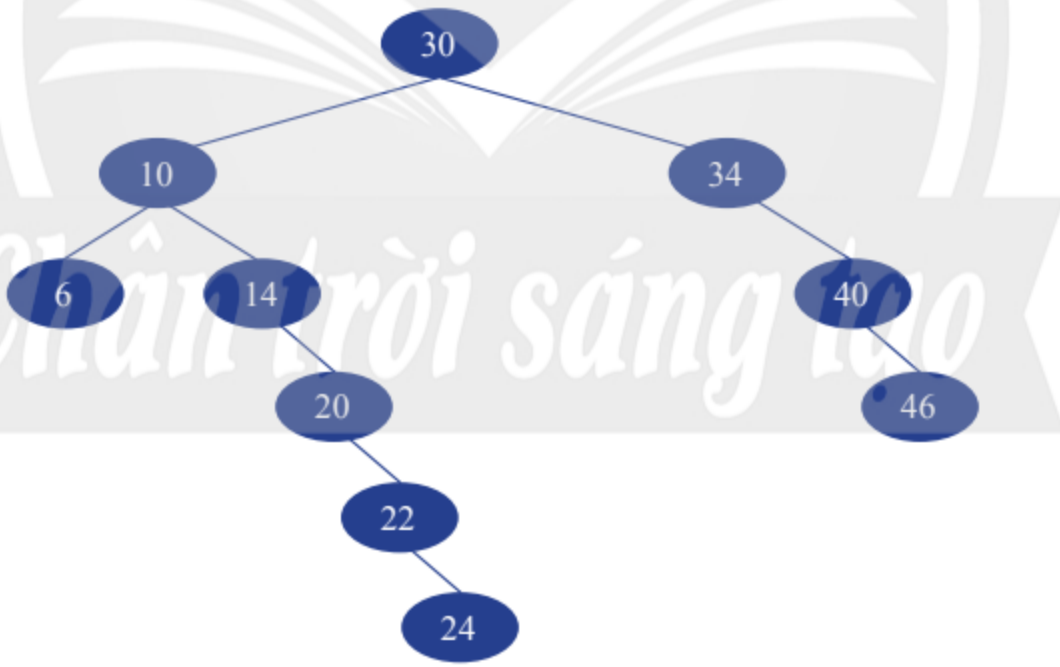
KHỞI ĐỘNG

Cho tập hợp A gồm các số nguyên dương A = {6, 14, 10, 34, 40, 30, 46, 20, 24, 22} được lưu trữ bằng hai cách sau:

Cách 1: Lưu vào mảng một chiều.

6	14	10	34	40	30	46	20	24	22

Cách 2: Lưu vào các nút của cây nhị phân.



Hình 1. Cây nhị phân

Cho giá trị x = 20. Em hãy trình bày:

a) Cách tìm kiếm x trong mảng A.

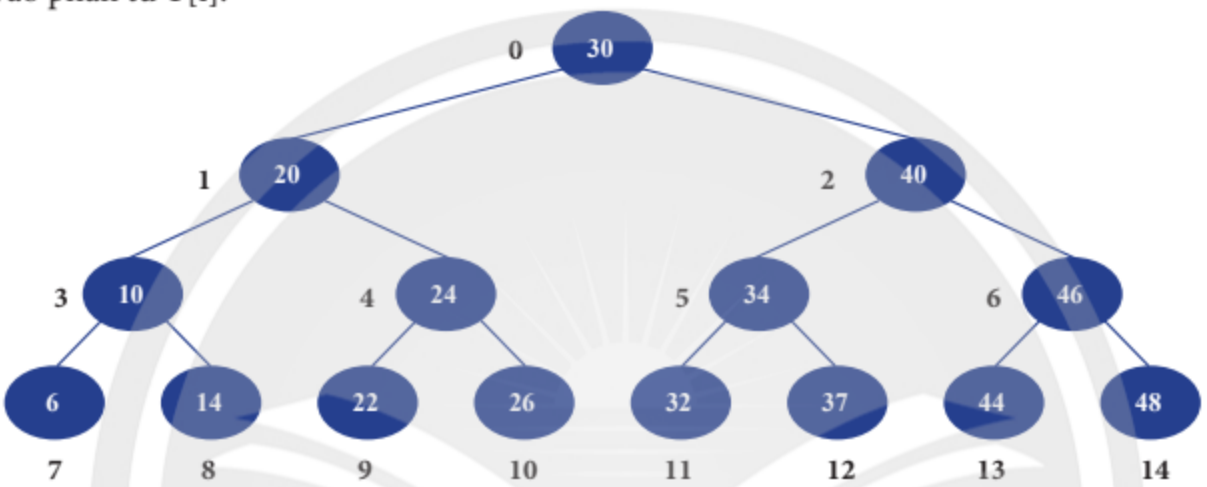
b) Cách tìm kiếm x trong cây nhị phân.

KHÁM PHÁ

1. Khái niệm cây tìm kiếm nhị phân

Trong cây nhị phân ở *Hình 1*, giá trị của nút bất kì luôn lớn hơn giá trị của tất cả nút thuộc cây con trái và nhỏ hơn giá trị của tất cả nút thuộc cây con phải. Cây nhị phân thoả tính chất này được gọi là cây tìm kiếm nhị phân.

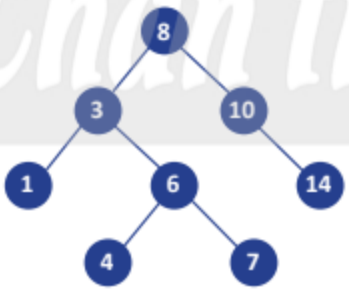
Cho cây tìm kiếm nhị phân như *Hình 2*. Em đánh chỉ số cho các nút như sau: nút gốc có chỉ số 0; nút có chỉ số i có nút con trái là nút có chỉ số 2*i + 1 và nút con phải có chỉ số 2*i + 2. Lưu trữ cây tìm kiếm nhị phân này vào mảng một chiều T: giá trị của nút có chỉ số i được lưu trữ vào phần tử T[i].



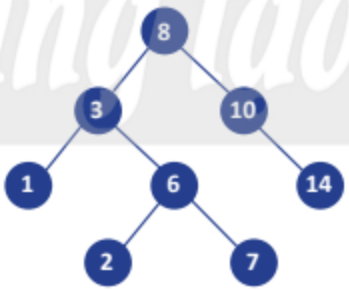
Hình 2. Cây tìm kiếm nhị phân sau khi được đánh chỉ số cho các nút

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T[i]	30	20	40	10	24	34	46	6	14	22	26	32	37	44	48

Hình nào trong *Hình 3* biểu diễn cây tìm kiếm nhị phân?



Hình 3a.



Hình 3b.

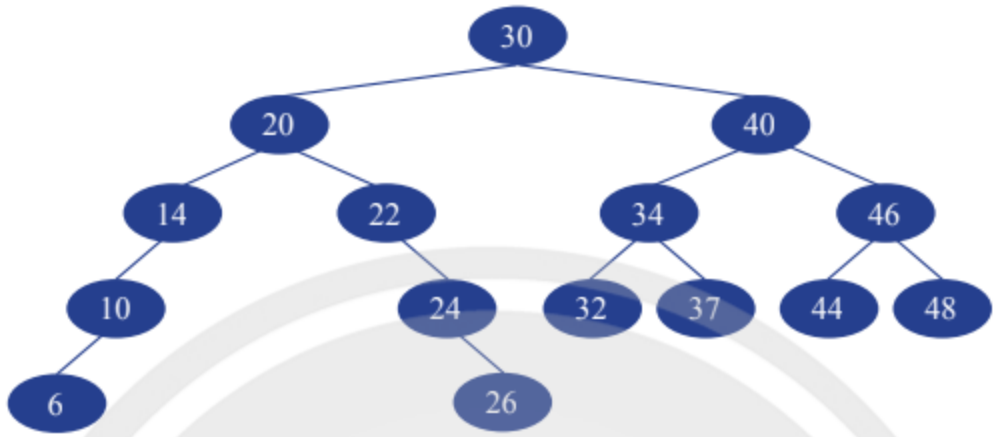
Hình 3. Các cây nhị phân

Cây tìm kiếm nhị phân là cây nhị phân, trong đó, giá trị khoá của nút bất kì luôn lớn hơn giá trị khoá của tất cả nút thuộc cây con trái và nhỏ hơn giá trị khoá của tất cả nút thuộc cây con phải.

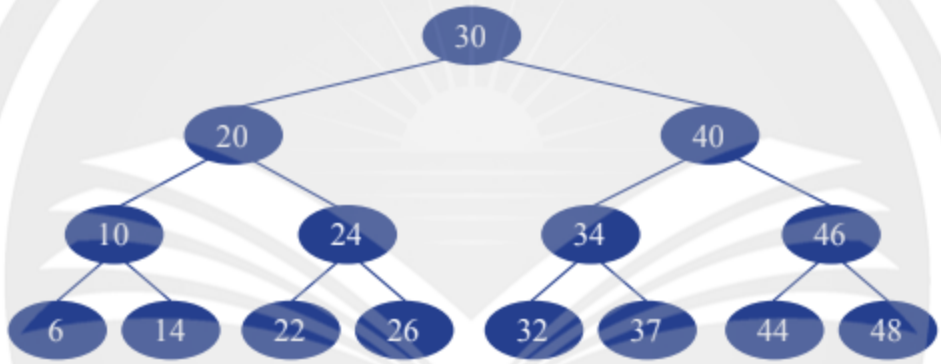
2. Mô phỏng thuật toán tạo cây tìm kiếm nhị phân

a) Cây tìm kiếm nhị phân biểu diễn một tập số nguyên dương

Cây tìm kiếm nhị phân ở *Hình 4a* và *Hình 4b* biểu diễn cùng một tập hợp A ở KHỞI ĐỘNG, nhưng có chiều cao khác nhau. Ngoài ra, phép toán duyệt giữa cây tìm kiếm nhị phân cho kết quả là một dãy các giá trị tăng dần.



Hình 4a.



Hình 4b.

Hình 4. Hai cây tìm kiếm nhị phân biểu diễn cùng một tập hợp số nguyên dương

b) Thuật toán tạo cây tìm kiếm nhị phân biểu diễn tập hợp số nguyên dương

Ví dụ: Cho mảng A gồm các số nguyên dương: A = [15, 3, 16, 7]. Cây tìm kiếm nhị phân được xây dựng từ mảng A theo các bước như sau:

Ban đầu, cây tìm kiếm nhị phân T là rỗng.

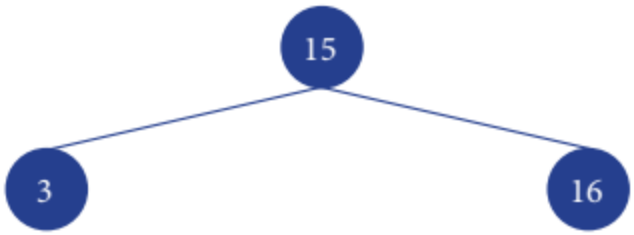
Thêm vào khoá 15: Vì cây T rỗng, nên tạo nút mới có khoá 15 và cây T có nút 15.

Hình 5. Cây T có nút 15

Thêm vào khoá 3: Vì 3 < 15, nên thêm 3 vào cây con trái của nút 15. Nhưng nút 15 không có cây con trái, nên tạo nút mới có khoá 3 và nút này là nút con trái của nút 15.

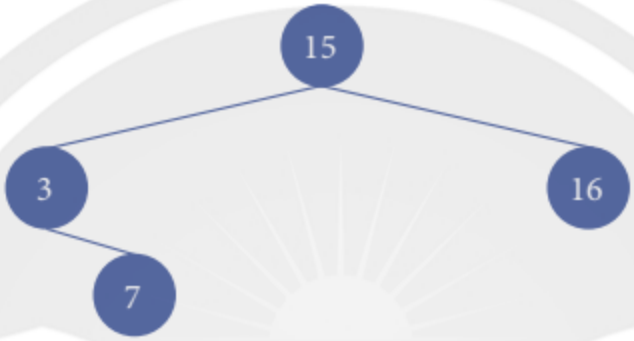
Hình 6. Thêm nút mới có khoá 3 vào cây T

Thêm vào khoá 16: Vì 16 > 15, nên thêm 16 vào cây con phải của nút 15. Nhưng nút 15 không có cây con phải, nên tạo nút mới có khoá 16 và nút này là nút con phải của nút 15.



Hình 7. Thêm nút mới có khoá 16 vào cây T

Thêm vào khoá 7: Vì 7 < 15, nên thêm 7 vào cây con trái của nút 15 (có gốc là nút 3). Vì 7 > 3, nên thêm 7 vào cây con phải của nút 3. Nhưng nút 3 không có cây con phải, nên tạo nút mới có khoá 7 và nút này là nút con phải của nút 3.



Hình 8. Thêm nút mới có khoá 7 vào cây T

Nhận xét: Nút mới thêm vào cây tìm kiếm nhị phân luôn luôn là nút lá.

Thuật toán sau đây tạo cây tìm kiếm nhị phân T từ một dãy số nguyên dương a. Cây T là cây nhị phân hoàn chỉnh (có thể có các nút None) được cài đặt bằng danh sách (kiểu list của Python).

```
#Thêm khoá v vào cây tìm kiếm nhị phân T gốc i
insertTree(T, i, v):
    if i>=len(T):
        Thêm (i - len(T) + 1) giá trị None vào cuối T
    if T[i] == None:
        T[i] = v
    elif v == T[i]:
        print("Đã tồn tại nút có giá trị bằng", v)
    elif v < T[i]:
        insertTree(T, 2*i + 1, v)
    else:
        insertTree(T, 2*i + 2, v)
createBSTTree(T, a):
    for v in a:
        insertTree(T, 0, v)
printTree(T):
    for v in T:
        if v != None:
            print(v)
inOrderTraversal(T, i):
    if i < len(T) and T[i] != None:
        inOrderTraversal(T, 2*i + 1)
        xử lí nút i
        inOrderTraversal(T, 2*i + 2)
```

```
Input danh sách a
T = []                                  #Khởi tạo cây T rỗng
createBSTTree(T, a)                     #Tạo cây T từ danh sách a
printTree(T)                            #In cây T
inOrderTraversal(T, 0)                  #Duyệt giữa cây T
```

Chú thích: T là mảng lưu cây tìm kiếm nhị phân, v là giá trị cần chèn, i là vị trí đang duyệt.

Cho mảng A = [5, 7, 30, 23, 34, 15]. Hãy vẽ cây tìm kiếm nhị phân biểu diễn mảng A.

Có nhiều cây tìm kiếm nhị phân biểu diễn cùng một tập hợp số nguyên dương.

LUYỆN TẬP

1. Em hãy vẽ cây tìm kiếm nhị phân bằng cách đưa vào cây rỗng lần lượt các phần tử của mảng A = [3, 6, 13, 7, 5, 2, 8, 9].

2. Trình bày thuật toán xác định giá trị x = 34 có thuộc cây tìm kiếm nhị phân được biểu diễn ở *Hình 4b* hay không.

Hướng dẫn:

Cách 1: Sử dụng các phép toán duyệt trước, duyệt giữa, duyệt sau để xác định giá trị x = 34 có thuộc cây tìm kiếm nhị phân ở *Hình 4b* hay không.

Ví dụ: Sử dụng phép duyệt trước để tìm giá trị x.

```
def insertTree(T, i, v):
    if i >= len(T):
        T.extend([None]*(i-len(T)+1))
    if T[i] == None:
        T[i] = v
    elif v == T[i]:
        print("Đã tồn tại nút có giá trị bằng", v)
    elif v < T[i]:
        insertTree(T, 2*i + 1, v)
    else:
        insertTree(T, 2*i + 2, v)
def createBSTTree(T, a):
    for v in a:
        insertTree(T, 0, v)
def preOrderSearch(T, i, x):
    global found
    if i < len(T) and T[i] != None:
        if T[i] == x:
            found = True
            return
        else:
            preOrderSearch(T, 2*i + 1, x)
            preOrderSearch(T, 2*i + 2, x)
def Search(T, x):
    global found
    found = False
    preOrderSearch(T, 0, x)
    return found
```

```
29. a = list(map(int,input().split()))
30. x = int(input())
31. T = []
32. createBSTTree(T, a)
33. found = Search(T, x)
34. print(found)
```

Cách 2: Sử dụng thuật toán đệ quy search(T, i, x) để tìm kiếm x trên cây tìm kiếm nhị phân T gốc i.

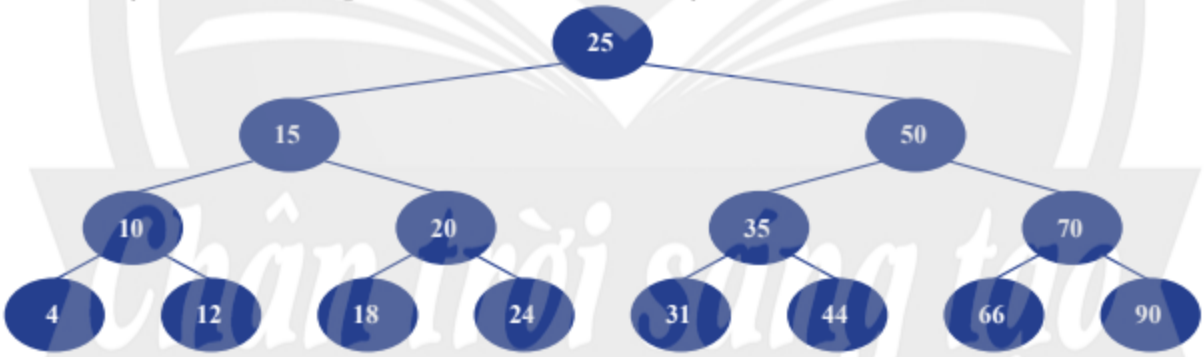
Mã nguồn hàm tìm kiếm giá trị trên cây tìm kiếm nhị phân sử dụng đệ quy:

Em có thể sử dụng đệ quy hoặc vòng lặp để tìm một nút trên cây tìm kiếm nhị phân. Hàm đệ quy search(T, i, x) dùng để tìm kiếm giá trị x trên cây tìm kiếm nhị phân T gốc i.

```
01. #Tìm x trên cây tìm kiếm nhị phân T gốc i
02. def search(T, i, x):
03.     if i >= len(T) or T[i] == None:     #Cây T gốc i là rỗng
04.         return False                    #Không tìm thấy x
05.     elif T[i] == x:
06.         return True                     #Tìm thấy x
07.     elif x < T[i]:
08.         return search(T, 2*i+1, x)      #Tìm x trên cây con trái
09.     else:
10.         return search(T, 2*i+2, x)      #Tìm x trên cây con phải
```

VẬN DỤNG

1. Cho cây tìm kiếm nhị phân như *Hình 9*. Em hãy thực hiện:



Hình 9. Cây tìm kiếm nhị phân

a) Mô tả các bước để tìm giá trị x = 22 có trong cây theo các thuật toán: duyệt trước, duyệt giữa, duyệt sau và tìm kiếm trên cây tìm kiếm nhị phân.

b) Với các thuật toán ở câu a), trong trường hợp tổng quát của cây tìm kiếm nhị phân, thuật toán nào có số lần so sánh khoá cần tìm với khoá của các nút là ít nhất.

c) Viết chương trình tạo cây tìm kiếm nhị phân ở *Hình 9*. Sau đó, in ra màn hình các khoá có trong cây này theo thứ tự tăng dần.

2. Em hãy nêu các bước để tìm kiếm giá trị x = 5 có trong cây tìm kiếm nhị phân vừa xây dựng được ở

LUYỆN TẬP

.

BÀI 2.4 THỰC HÀNH CÂY TÌM KIẾM NHỊ PHÂN

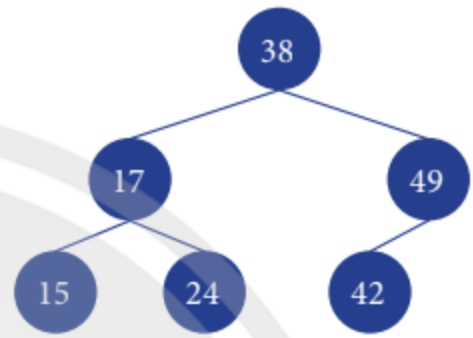
MỤC TIÊU **Sau bài học này, em sẽ:**

Ứng dụng được cây tìm kiếm nhị phân giải bài toán sắp xếp và tìm kiếm.

KHỞI ĐỘNG

Cho cây tìm kiếm nhị phân ở *Hình 1*.

Ở bài học trước, em đã được giới thiệu về việc giá trị các nút trong cây tìm kiếm nhị phân được sắp xếp theo một trình tự nhất định. Em hãy mô phỏng thuật toán để xuất giá trị các nút của cây tìm kiếm nhị phân trong *Hình 1* theo thứ tự tăng dần.



Hình 1. Cây tìm kiếm nhị phân

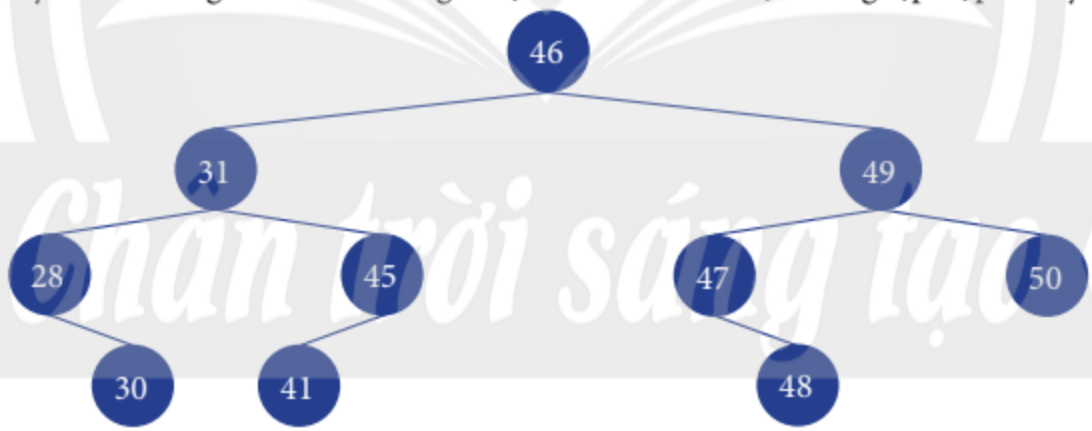
THỰC HÀNH

Nhiệm vụ 1. Ứng dụng cây tìm kiếm nhị phân để giải bài toán tìm kiếm

Yêu cầu: Cho cây tìm kiếm nhị phân (*Hình 2*) biểu diễn tập hợp số nguyên dương

$$A = \{46, 49, 31, 45, 41, 50, 47, 28, 30, 48\}.$$

Em hãy viết chương trình kiểm tra giá trị x = 41 có xuất hiện trong tập hợp A hay không.



Hình 2. Cây tìm kiếm nhị phân

Hướng dẫn: Sử dụng chương trình tạo cây tìm kiếm nhị phân và chương trình con tìm kiếm ở Bài 2.3 để thực hiện yêu cầu trên.

Mã nguồn tham khảo:

```
#Tìm x trên cây tìm kiếm nhị phân T gốc i
def search(T, i, x):
    if i >= len(T) or T[i] == None:          #Cây T gốc i là rỗng
        return False                         #Không tìm thấy x
    elif T[i] == x:
        return True                          #Tìm thấy x
    elif x < T[i]:
```

```
        return search(T, 2*i+1, x)          #Tìm x trên cây con trái
    else:
        return search(T, 2*i+2, x)          #Tìm x trên cây con phải
#Thêm giá trị v vào cây tìm kiếm nhị phân T gốc i
def insertTree(T, i, v):
    if i>=len(T):
        T.extend([None]*(i-len(T)+1))
    if T[i] == None:
        T[i] = v
    elif v == T[i]:
        print("Đã tồn tại nút có giá trị bằng", v)
    elif v < T[i]:
        insertTree(T, 2*i + 1, v)
    else:
        insertTree(T, 2*i + 2, v)
#Tạo cây tìm kiếm nhị phân T từ mảng a
def createBSTTree(T, a):
   for i in range(len(a)):
        insertTree(T, 0, a[i])              #Thêm a[i] vào cây T
def printTree(T):
    for i in T:
        if i!=None:
            print(i, end=" ")
def inOrderTraversal(tree, i):
    if i < len(tree) and tree[i] != None:    #Cây gốc i khác rỗng
        inOrderTraversal(tree, 2*i + 1)      #Duyệt cây con trái
        print(tree[i],end = ' ')             #Xử lí nút gốc
        inOrderTraversal(tree, 2*i + 2)      #Duyệt cây con phải
print("nhập danh sách các số để tạo cây: ")
a = list(map(int, input().split()))
print("nhập giá trị cần tìm: ")
x = int(input())
T = []
createBSTTree(T,a)
printTree(T)
print()
print (search (T, 0, x))
inOrderTraversal(T, 0):
```

Nhiệm vụ 2. Sắp xếp mảng dùng cây tìm kiếm nhị phân

Yêu cầu: Sắp xếp một mảng số nguyên a tăng dần.

Hướng dẫn: Thực hiện các bước sau:

1. Tạo cây tìm kiếm nhị phân T từ mảng a.
2. Duyệt giữa cây tìm kiếm nhị phân T.

Mã nguồn tham khảo:

```
def insertTreeT(T, i, v):
    if i >= len(T):
        T.extend([None]*(i-len(T)+1))
    if T[i] == None:
        T[i] = v
    elif v < T[i]:
        insertTreeT(T, 2*i + 1, v)
    else:
        insertTreeT(T, 2*i + 2, v)
def createTreeT(T, a):
    for i in range(len(a)):
```

```
            insertTreeT(T, 0, a[i])
def inOrderSort(T, i, a):
    if i < len(T) and T[i] != None:
        inOrderSort(T, 2*i + 1, a)
        a.append(T[i])
        inOrderSort(T, 2*i + 2, a)
def SortBST(a):
    T = []
    createTreeT(T, a)
    a.clear()
    inOrderSort(T, 0, a)
print("Nhập mảng cần sắp xếp tăng dần: ")
a = list(map(int, input().split()))
SortBST(a)
print("Mảng có thứ tự tăng dần:", a)
```

LUYỆN TẬP

Nhiệm vụ. Tìm thanh gỗ theo yêu cầu

Tại xưởng gỗ, bác thợ mộc cần tìm một thanh gỗ có kích thước là k (cm) trong n thanh gỗ có các kích thước khác nhau để đóng tủ.

Yêu cầu: Để giúp bác thợ mộc tìm thanh gỗ đúng với kích thước đã cho. Hãy viết chương trình.

Hướng dẫn: Sử dụng chương trình tạo cây tìm kiếm nhị phân và chương trình con tìm kiếm để giải quyết bài toán tìm kiếm trên.

Dữ liệu vào:

– Hai số nguyên n, k (giá trị n cho biết có n thanh gỗ trong xưởng; giá trị k cho biết kích thước thanh gỗ mà bác thợ mộc cần tìm).

– n giá trị tiếp theo là kích thước của n thanh gỗ có trong xưởng.

Dữ liệu ra: In ra màn hình "Có" nếu tìm thấy thanh gỗ đúng kích thước. Ngược lại, in "Không".

Ví dụ:

Dữ liệu vào	Dữ liệu ra
8 200 120 180 200 160 165 198 123 167	Có
10 170 210 120 100 180 167 169 143 171 190 200	Không

VẬN DỤNG

Cho tập hợp số nguyên dương A = {28, 21, 43, 13, 23, 35, 50, 10, 15, 22, 27, 30, 40, 47, 52}.

a) Viết chương trình tạo cây tìm kiếm nhị phân T biểu diễn tập hợp A.

b) Vẽ minh hoạ cây T.

c) Viết chương trình kiểm tra giá trị x = 10 có xuất hiện trong cây tìm kiếm nhị phân T hay không.

d) Viết chương trình xuất ra màn hình các giá trị của tập hợp A được sắp xếp giảm dần.

Chuyên đề 3

TÌM HIỂU KĨ THUẬT DUYỆT ĐỒ THỊ VÀ ỨNG DỤNG

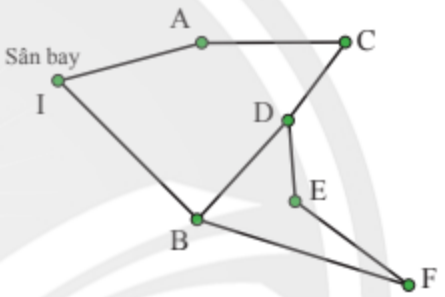
BÀI 3.1 CÁC KHÁI NIỆM CƠ BẢN CỦA ĐỒ THỊ

MỤC TIÊU **Sau bài học này, em sẽ:**

Trình bày được một số khái niệm cơ bản của đồ thị.

KHỞI ĐỘNG

Một khách du lịch xuất phát từ sân bay của thành phố, muốn ghé thăm tất cả các địa điểm nổi tiếng A, B, C, D, E, F, mỗi địa điểm đúng một lần sau đó quay trở lại sân bay. Theo em, có tồn tại một hành trình như vậy không?



Hình 1. Bản đồ đường đi giữa các địa điểm tham quan

KHÁM PHÁ

1. Khái niệm đồ thị

Trong nhiều tình huống, người ta thường vẽ những sơ đồ, gồm những điểm biểu thị các đối tượng được xem xét (địa điểm, người, đội bóng,…) và nối một số điểm với nhau bằng đoạn đường cong (hoặc thẳng) hay mũi tên tượng trưng cho mối quan hệ nào đó giữa các đối tượng. Các sơ đồ như vậy xuất hiện ở nhiều nơi trong thực tế như: sơ đồ mạng điện, sơ đồ giao thông,… Đó là những hình ảnh về đồ thị.

Đồ thị G = (V, E) là một cấu trúc gồm hai tập hợp, trong đó tập V chứa các đỉnh và tập E chứa các cạnh, mỗi cạnh kết nối hai đỉnh của đồ thị với nhau. Nếu đồ thị G = (V, E) với V = ∅ và E = ∅ thì G được gọi là đồ thị rỗng. Nếu đồ thị G = (V, E) với V ≠ ∅ và E = ∅ thì G được gọi là đồ thị trống.

Các đỉnh của đồ thị thường được kí hiệu bằng các chữ cái A, B, C,… hoặc u, v,… hoặc bằng cách đánh số 1, 2, 3,… Cạnh kết nối đỉnh A với đỉnh B được kí hiệu là {A, B}. Ví dụ, đồ thị ở *Hình 1* có 7 đỉnh A, B, C, D, E, F, I và có 8 cạnh là {I, A}, {I, B}, {A, C}, {C, D}, {D, B}, {D, E}, {E, F}, {B, F}.

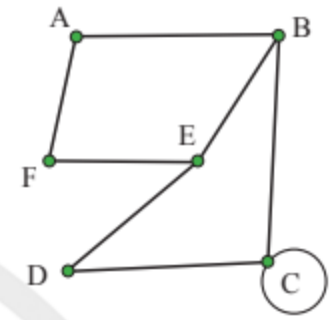
Hai đỉnh kề nhau: Hai đỉnh của đồ thị được gọi là kề nhau nếu chúng được kết nối bởi một cạnh. Một đỉnh không kề với đỉnh nào gọi là đỉnh cô lập (đỉnh treo). Ví dụ, hai đỉnh A và C ở *Hình 1* được gọi là hai đỉnh kề nhau.

***Hai cạnh kề nhau*:** Hai cạnh của một đồ thị được gọi là kề nhau nếu chúng có chung một đỉnh. Ví dụ, hai cạnh {A, C} và {C, D} ở *Hình 1* có chung đỉnh C là hai cạnh kề nhau.

***Khuyên*:** Cạnh của đồ thị có hai đầu mút trùng nhau gọi là khuyên. Ví dụ, cạnh {C, C} của đồ thị trong *Hình 2* là khuyên.

***Đường đi*:** Đường đi p từ đỉnh u_1 đến đỉnh u_n là dãy các cạnh kề nhau đi từ đỉnh u_1 đến đỉnh u_n và không có chu trình. Được kí hiệu là $p = (u_1, u_2, \ldots, u_n)$. Ví dụ, dãy các cạnh {A, C}, {C, D}, {D, E} ở *Hình 1* là một đường đi từ đỉnh A đến đỉnh E, được kí hiệu là (A, C, D, E).

Ví dụ: *Hình 2* biểu diễn một đồ thị có đỉnh A và B kề nhau, đỉnh A và đỉnh C không kề nhau vì không có cạnh nào của đồ thị kết nối chúng, cạnh {A, B} và cạnh {B, C} kề nhau do có chung đỉnh B, tại đỉnh C có một khuyên. Dãy các cạnh {A, F}, {F, E}, {E, D} là một đường đi từ đỉnh A đến đỉnh D. Dãy các cạnh {A, F}, {F, E}, {E, B}, {B, A} không là đường đi.



Hình 2. Minh hoạ các khái niệm của đồ thị

Một mạng máy tính gồm có 7 máy tính được kết nối với một máy chủ thông qua một switch. Máy chủ được kết nối với mạng Internet thông qua modem. Ngoài ra, máy chủ còn được kết nối với 2 máy in. Em hãy:

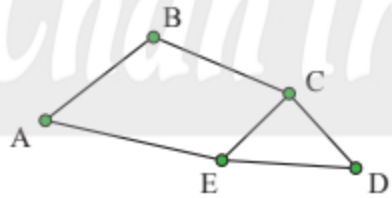
a) Vẽ đồ thị biểu diễn mạng máy tính.

b) Cho biết đồ thị đó có bao nhiêu đỉnh, bao nhiêu cạnh?

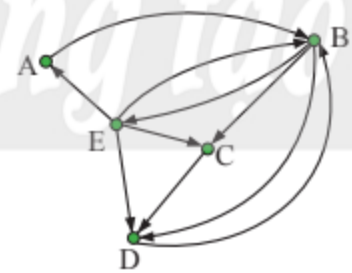
Đồ thị G = (V, E) là một cấu trúc gồm hai tập hợp: tập hợp đỉnh V bao gồm các đỉnh và tập hợp cạnh E bao gồm các cạnh nối các đỉnh.

2. Một số dạng đồ thị

Đơn đồ thị là đồ thị không chứa khuyên và giữa hai đỉnh bất kì có nhiều nhất một cạnh. Ví dụ, *Hình 3* là một đơn đồ thị có 5 đỉnh và 6 cạnh.



Hình 3. Minh hoạ đơn đồ thị

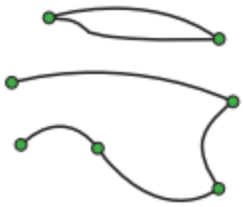


Hình 4. Minh hoạ đồ thị có hướng

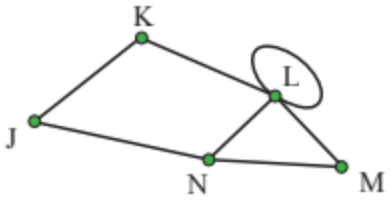
Đồ thị có hướng là đồ thị mà mỗi cạnh có chiều, cạnh còn được gọi là cung. Cạnh kết nối từ đỉnh u đến đỉnh v được kí hiệu là (u, v) và được biểu diễn bằng mũi tên đi từ đỉnh u đến đỉnh v; u là đỉnh gốc (đỉnh đầu) và v là đỉnh ngọn (đỉnh cuối); v là đỉnh kề của u; u không là đỉnh kề của v. Cạnh (u, v) và cạnh (v, u) là hai cạnh khác nhau. Ví dụ, *Hình 4* là một đồ thị có hướng.

Đồ thị vô hướng là đồ thị mà tất cả các cạnh đều không có chiều. Cạnh {u, v} và cạnh {v, u} là giống nhau. Ví dụ, các đồ thị ở *Hình 1, 2, 3* là các đồ thị vô hướng.

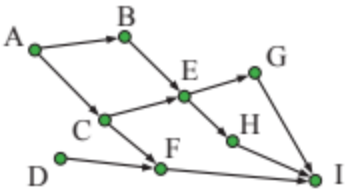
Em hãy cho biết trong các đồ thị ở *Hình 5*, đồ thị nào là đơn đồ thị, đồ thị vô hướng, đồ thị có hướng.



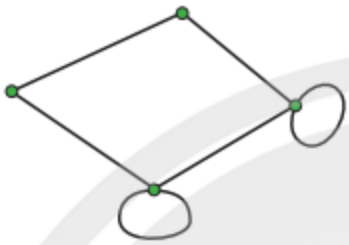
Hình 5a.



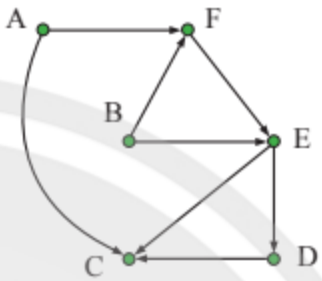
Hình 5b.



Hình 5c.



Hình 5d.



Hình 5e.

Hình 5. Một số dạng đồ thị

Có nhiều loại đồ thị để biểu diễn các đối tượng với các mục đích khác nhau như đơn đồ thị, đồ thị có hướng, đồ thị vô hướng,…

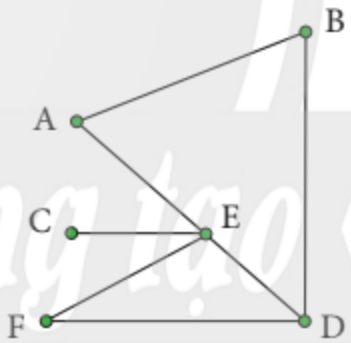
LUYỆN TẬP

Cho đồ thị như ở *Hình 6*. Em hãy cho biết:

a) Tập các đỉnh và tập các cạnh của đồ thị.

b) Các đỉnh kề với đỉnh A, D.

c) Các cạnh kề với cạnh {C, E}, {D, F}.



Hình 6. Đồ thị G

THỰC HÀNH

1. Một nhóm 5 người A, B, C, D, E trong đó:

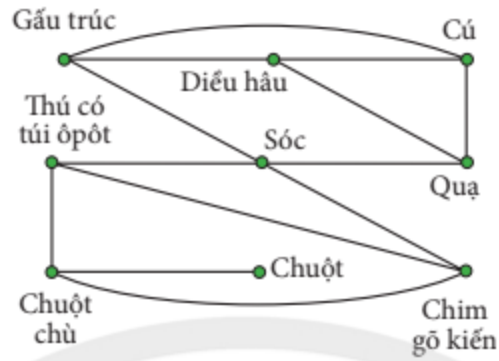
– A cùng tuổi với B;

– C cùng tuổi với D nhưng không cùng tuổi với A;

– Riêng E không cùng tuổi với ai.

Hãy sử dụng đồ thị để biểu diễn quan hệ cùng tuổi của nhóm người này.

2. *Hình 7* là đồ thị cho biết sự tương tác giữa các loài động vật. Mỗi loài được biểu diễn bằng một đỉnh. Mỗi cạnh nối hai đỉnh thể hiện các loài này cạnh tranh với nhau (các loài có chung nguồn thức ăn). Quan sát hình và cho biết những loài nào không cạnh tranh với nhau?



Hình 7. Đồ thị tương tác giữa các loài vật

VẬN DỤNG

1. Trong một cuộc họp có hai người A và B chỉ nói được tiếng Anh, ba người C, D, E chỉ nói được tiếng Việt và một thông dịch viên I.

a) Để biểu diễn ai có thể giao tiếp trực tiếp được với nhau trong cuộc họp. Hãy vẽ đồ thị.

b) Cho biết có bao nhiêu người có thể giao tiếp trực tiếp được với B, E, I.

2. *Bảng 1* cho biết cân nặng của một số thành viên trong một câu lạc bộ Judo. Hai người có thể thi đấu với nhau nếu cân nặng của họ hơn kém nhau không quá 10 kg. Theo em, những cặp nào có thể thi đấu được với nhau?

Bảng 1. Cân nặng của một số thành viên trong câu lạc bộ Judo

Thành viên	Khối lượng (kg)
An	59
Nam	75
Trường	94
Đông	68
Hiển	71

BÀI 3.2 BIỂU DIỄN ĐỒ THỊ

MỤC TIÊU **Sau bài học này, em sẽ:**

Biểu diễn được đồ thị bằng ma trận kề (mảng hai chiều) và danh sách kề.

KHỞI ĐỘNG

Em hãy nhắc lại định nghĩa mảng hai chiều và cách khai báo mảng hai chiều trong ngôn ngữ Python. Theo em, có thể sử dụng mảng hai chiều để biểu diễn một đồ thị được không?

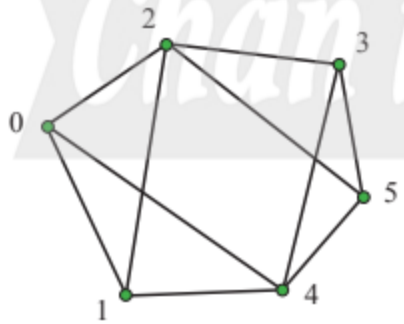
KHÁM PHÁ

1. Biểu diễn đồ thị bằng ma trận kề (mảng hai chiều)

Cho đồ thị G gồm có tập đỉnh $V = \{v_0, v_1, v_2, \ldots, v_{n-1}\}$ và tập cạnh $E = \{e_1, e_2, \ldots, e_m\}$. Đồ thị G có thể được biểu diễn bằng một ma trận kề A (mảng hai chiều) có n hàng và n cột, với n là số đỉnh của đồ thị. Phần tử ở hàng i cột j của ma trận kề A, kí hiệu là A[i, j], được xác định như sau:

$$A[i, j] = \begin{cases} 1, & \text{nếu đỉnh } v_j \text{ là đỉnh kề của đỉnh } v_i \\ 0, & \text{nếu đỉnh } v_j \text{ không là đỉnh kề của đỉnh } v_i. \end{cases}$$

Ví dụ 1: Đồ thị vô hướng G_1 ở *Hình 1* gồm có 6 đỉnh và 10 cạnh. Ma trận kề biểu diễn đồ thị G_1 là một mảng hai chiều có 6 hàng, 6 cột như ở *Bảng 1*.

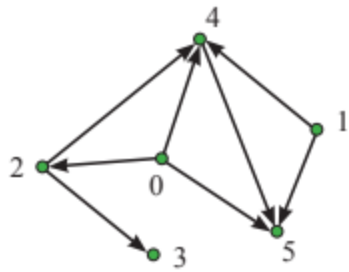


Hình 1. Đồ thị vô hướng G_1

Bảng 1. Ma trận kề biểu diễn đồ thị vô hướng G_1

i\j	0	1	2	3	4	5
0	0	1	1	0	1	0
1	1	0	1	0	1	0
2	1	1	0	1	0	1
3	0	0	1	0	1	1
4	1	1	0	1	0	1
5	0	0	1	1	1	0

Ví dụ 2: Đồ thị có hướng G_2 ở *Hình 2* gồm có 6 đỉnh và 8 cạnh. Ma trận kề biểu diễn đồ thị G_2 là một mảng hai chiều có 6 hàng, 6 cột như ở *Bảng 2*.



Hình 2. Đồ thị có hướng G_2

Bảng 2. Ma trận kề biểu diễn đồ thị có hướng G_2

i\j	0	1	2	3	4	5
0	0	0	1	0	1	1
1	0	0	0	0	1	1
2	0	0	0	1	1	0
3	0	0	0	0	0	0
4	0	0	0	0	0	1
5	0	0	0	0	0	0

Ma trận kề được cài đặt bằng danh sách (kiểu list của Python). Sau đây là các hàm cho phép thêm các đỉnh, các cạnh vào đồ thị và hàm tạo đồ thị (vô hướng hoặc có hướng) từ tệp.

```
#Thêm đỉnh v vào đồ thị biểu diễn bằng ma trận kề
#graph:    đồ thị biểu diễn bằng ma trận kề
#vertices: danh sách các đỉnh đã có trong đồ thị
#v:        đỉnh cần thêm vào
def addVertex2AdjMatrixGraph(graph, vertices, v):
  if v in vertices:                          #Kiểm tra  đỉnh v đã tồn tại
    print("Đỉnh ", v, " đã tồn tại.")
  else:
    vertices.append(v)                       #Thêm đỉnh mới vào vertices
    numVertices = len(vertices)
    if numVertices > 1:
        for vertex in graph:                 #Thêm cột mới vào ma trận kề
            vertex.append(0)
    graph.append([0]*numVertices)
#Thêm cạnh có đỉnh đầu  v1 và đỉnh cuối v2 vào đồ thị biểu diễn bằng ma trận kề.
#graph:    đồ thị biểu diễn bằng ma trận kề
#vertices: danh sách các đỉnh đã có trong đồ thị
#v1, v2:   hai đỉnh của cạnh cần thêm vào
#flag = 0  đồ thị vô hướng (mặc định flag = 0)
#flag = 1  đồ thị có hướng
def addEdge2AdjMatrixGraph(graph, vertices, v1, v2, flag = 0):
  #Kiểm tra nếu một trong hai đỉnh không có trong vertices thì báo lỗi
  if v1 not in vertices:
    print("Đỉnh ", v1, " không tồn tại.")
  elif v2 not in vertices:
    print("Đỉnh ", v2, " không tồn tại.")
  else:
    r = vertices.index(v1)                   #Lấy r là chỉ số của đỉnh v1
    c = vertices.index(v2)                   #Lấy c là chỉ số của đỉnh v2
    graph[r][c] = 1
    if flag == 0:                            #Đồ thị vô hướng
      graph[c][r] = 1
```

Hàm kiểm tra tệp đồ thị

```
#Hàm kiểm tra tệp đồ thị hợp lệ
#flag = 0  Tệp đồ thị vô hướng
#flag = 1  Tệp đồ thị có hướng
def isValidGraphFile(filename, flag = 0):
  f = open(filename)                         #Mở tệp
  line = f.readline()                        #Đọc hàng đầu chứa các đỉnh từ tệp
  vs = [v for v in line.split()]
  if len(vs) != len(set(vs)):                #Kiểm tra có đỉnh trùng
    f.close()
    raise ValueError("Đồ thị có đỉnh trùng!")
  graph = [[0 for c in range(len(vs))] for c in range(len(vs))]
  for line in f:                             #Thêm các cạnh vào graph
    va = [v for v in line.split()]
```

```
    if len(va) <= 1:
      raise ValueError("Cạnh chỉ có một đỉnh!")
    if va[0] not in vs:
      raise ValueError(f"Đỉnh {va[0]} không có trong đồ thị")
    if va[1] not in vs:
      raise ValueError(f"Đỉnh {va[1]} không có trong đồ thị")
    r = vs.index(va[0])                #Lấy r là chỉ số của đỉnh v[0]
    c = vs.index(va[1])                #Lấy c là chỉ số của đỉnh v[1]
    graph[r][c] += 1
    if flag == 0:
      graph[c][r] += 1
  f.close()
  for r in range (len(vs)):
    for c in range(len(vs)):
      if r == c and graph[r][c] > 0:  #Có khuyên
        raise ValueError(f"Đồ thị có khuyên tại đỉnh {r}")
      if graph[r][c] > 1:
        raise ValueError(f"Đồ thị có cạnh {r, c} trùng!")
  return True
isValidGraphFile('dothi.txt', 0)
```

Lưu ý: Để tạo đồ thị vô hướng/có hướng từ tệp dữ liệu vô hướng/có hướng, em có thể viết chương trình có tham số điều khiển (control parameter) là một cờ (flag) với quy ước:

Hàm createAdjMatrixGraph(filename, flag):

flag = 0 là tệp dữ liệu của đồ thị vô hướng,

flag = 1 là tệp dữ liệu của đồ thị có hướng.

Như vậy, tệp dữ liệu của đồ thị vô hướng chỉ chứa các cạnh (không phải chứa các cung, số lượng cung gấp đôi số lượng cạnh).

```
#Hàm tạo đồ thị biểu diễn bằng ma trận kề từ tệp
def createAdjMatrixGraph(filename, flag = 0):
  isValidGraphFile(filename, flag)       #Kiểm tra tệp đồ thị
  f = open(filename)                     #Mở tệp
  line = f.readline()                    #Đọc hàng đầu chứa các đỉnh từ tệp
  vs = [v for v in line.split()]         #Lấy danh sách đỉnh
  graph = []
  vertices = []
  for v in vs:                           #Thêm các đỉnh vào graph
    addVertex2AdjMatrixGraph(graph, vertices, v)
  for line in f:                         #Thêm các cạnh vào graph
    v1, v2 = [v for v in line.split()]
    addEdge2AdjMatrixGraph(graph, vertices, v1, v2, flag)
  f.close()
  return graph, vertices
graph, vertices = createAdjMatrixGraph('dothi.txt', 0)
print(graph)
```

Lưu ý: Tệp dothi.txt có dạng:

– Hàng đầu tiên là danh sách các đỉnh của đồ thị, mỗi đỉnh cách nhau bởi khoảng trắng.

– Các hàng kế tiếp: mỗi hàng chứa một cung gồm đỉnh gốc và đỉnh ngọn.

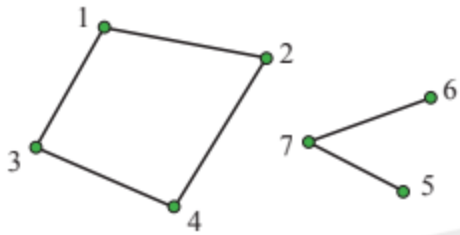
Ví dụ 3: Tệp dothi.txt tương ứng với đồ thị G_1 ở *Hình 1* có dạng:

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

Kết quả của chương trình trên là:

```
[[0, 1, 1, 0, 1, 0], [1, 0, 1, 0, 1, 0], [1, 1, 0, 1, 0, 1], [0, 0, 1, 0, 1, 1],
[1, 1, 0, 1, 0, 1], [0, 0, 1, 1, 1, 0]]
```

1. Cho đồ thị G_3 như *Hình 3*. Hãy biểu diễn đồ thị bằng ma trận kề.



Hình 3. Đồ thị G_3

Bảng 3. Ma trận kề biểu diễn đồ thị

i\j	0	1	2	3	4
0	0	1	0	1	0
1	1	0	1	1	0
2	0	1	0	0	1
3	1	1	0	0	1
4	0	0	1	1	0

2. Từ ma trận kề như *Hình 3*. Hãy vẽ đồ thị vô hướng tương ứng.

Đồ thị (vô hướng hoặc có hướng), với tập đỉnh $V = \{v_0, v_1, v_2, \ldots, v_{n-1}\}$, có thể được biểu diễn bằng một ma trận kề. Ma trận kề có n hàng và n cột với n là số đỉnh của đồ thị. Phần tử ở hàng i, cột j bằng 1 nếu đỉnh v_j là đỉnh kề của đỉnh v_i và bằng 0 nếu đỉnh v_j không là đỉnh kề của đỉnh v_i.

2. Biểu diễn đồ thị bằng danh sách kề

Cho đồ thị G gồm có tập đỉnh $V = \{v_0, v_1, v_2, \ldots, v_{n-1}\}$ và tập cạnh $E = \{e_1, e_2, \ldots, e_m\}$. Đồ thị G có thể được biểu diễn bằng n danh sách kề của đỉnh với n là số đỉnh của đồ thị. Danh sách kề của đỉnh v_i, kí hiệu là adjList(v_i) là tập

$$\text{adjList}(v_i) = \{u \mid \text{đỉnh } u \text{ kề với đỉnh } v_i\} \text{ với } 0 \le i \le n-1.$$

Cách biểu diễn như vậy gọi là biểu diễn đồ thị bằng danh sách kề.

Ví dụ 4: *Bảng 4* là danh sách kề biểu diễn đồ thị G_1 ở *Hình 1* và *Bảng 5* là danh sách kề biểu diễn đồ thị G_2 ở *Hình 2*.

Bảng 4. Danh sách kề biểu diễn đồ thị G_1

Đỉnh	adjList(i)
0	1, 2, 4
1	0, 2, 4
2	0, 1, 3, 5
3	2, 4, 5
4	0, 1, 3, 5
5	2, 3, 4

Bảng 5. Danh sách kề biểu diễn đồ thị G_2

Đỉnh	adjList(i)
0	2, 4, 5
1	4, 5
2	3, 4
3	
4	5
5	

Danh sách kề có thể được cài đặt bằng kiểu từ điển (dictionary) của Python. Phần tử thứ i trong từ điển sẽ chứa danh sách các đỉnh kề của đỉnh i. Sau đây là các hàm cho phép thêm các đỉnh, các cạnh vào đồ thị và hàm tạo danh sách kề của đồ thị với dữ liệu được lấy từ tệp.

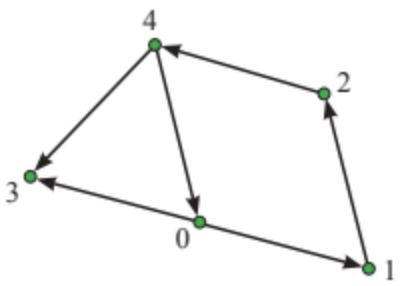
```
#Hàm thêm đỉnh u vào danh sách kề graph
def addVertex2AdjListGraph(graph, v):
  if v in graph:   #Nếu đỉnh v đã có, in thông báo
    print("Đỉnh", v, " đã tồn tại.")
  else:
    graph[v] = []  #Thêm đỉnh v vào graph, khởi tạo danh sách rỗng

#Hàm thêm cạnh (v1, v2) vào danh sách kề graph
def addEdge2AdjListGraph(graph, v1, v2, flag = 0):
  #Nếu đỉnh v1 hoặc v2 không có trong graph thì in thông báo
  if v1 not in graph:
    print("Đỉnh ", v1, " không tồn tại.")
  #Kiểm tra tính hợp lệ của đỉnh v2
  elif v2 not in graph:
    print("Đỉnh ", v2, " không tồn tại.")
  else:
    graph[v1].append(v2)                  #Thêm giá trị v2 vào graph[v1]
    if flag == 0 and v1 not in graph[v2]: #Đồ thị vô hướng
      graph[v2].append(v1)
#Hàm tạo đồ thị biểu diễn bằng danh sách kề từ tệp
def createAdjListGraph(filename, flag = 0):
  isValidGraphFile(filename, flag)       #Kiểm tra tệp đồ thị
  f = open(filename)                     #Mở tệp
  line = f.readline()                    #Đọc hàng đầu chứa các đỉnh từ tệp
  vertices = [v for v in line.split()] #Lấy danh sách đỉnh
  graph = {}
  for v in vertices:
    addVertex2AdjListGraph(graph, v)   #Thêm các đỉnh vào graph
  for line in f:                         #Thêm các cạnh vào graph
    v1, v2 = [v for v in line.split()]
    addEdge2AdjListGraph(graph, v1, v2, flag)
  f.close()
  return graph, vertices
graph, vertices = createAdjListGraph('dothi.txt', 0)
print(graph)
```

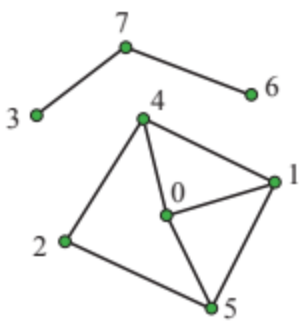
Kết quả của chương trình trên với tệp dothi.txt tương ứng với đồ thị G_1 ở *Hình 1* là:

```
{'0': ['1', '2', '4'], '1': ['0', '2', '4'], '2': ['0', '1', '3', '5'], '3':
['2', '4', '5'], '4': ['0', '1', '3', '5'], '5': ['2', '3', '4']}
```

1. Em hãy dùng danh sách kề biểu diễn các đồ thị ở *Hình 4* và *Hình 5*.



Hình 4. Đồ thị G_4



Hình 5. Đồ thị G_5

Bảng 6. Danh sách kề biểu diễn đồ thị

Đỉnh	adjList(u)
A	B, D
B	A, C, D
C	B, E
D	A, E
E	C, D

2. Từ danh sách kề ở *Bảng 6*. Hãy vẽ đồ thị có hướng tương ứng.

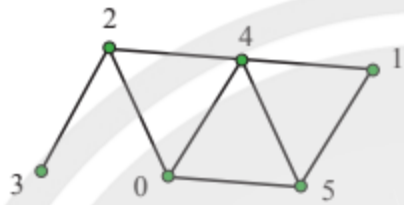
Đồ thị (vô hướng hoặc có hướng), với tập đỉnh $V = \{v_0, v_1, v_2, \ldots, v_{n-1}\}$, có thể được biểu diễn bằng n danh sách kề của đỉnh với n là số đỉnh của đồ thị. Danh sách kề của đỉnh v bao gồm các đỉnh u là đỉnh kề của đỉnh v.

LUYỆN TẬP

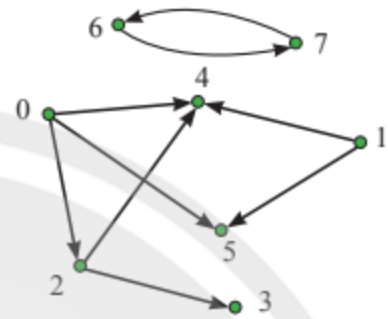
Cho hai đồ thị G_6 (*Hình 6*) và G_7 (*Hình 7*). Em hãy thực hiện biểu diễn bằng hai cách:

– Ma trận kề.

– Danh sách kề.



Hình 6. Đồ thị G_6



Hình 7. Đồ thị G_7

THỰC HÀNH

1. Sử dụng chương trình trong bài học, hãy viết chương trình xuất ra màn hình ma trận kề biểu diễn đồ thị G_2 (*Hình 2*) và G_3 (*Hình 3*).

2. Sử dụng chương trình trong bài học, hãy viết chương trình xuất ra màn hình danh sách kề biểu diễn đồ thị G_4 (*Hình 4*) và G_5 (*Hình 5*).

VẬN DỤNG

Một hãng hàng không đưa ra lịch bay trong ngày như sau:

– Từ TP.HCM: có một chuyến đến Hà Nội, Đà Nẵng, Phú Quốc, Nghệ An và Hải Phòng;

– Từ Hà Nội: có hai chuyến đến TP.HCM và một chuyến đến Đà Nẵng, Nghệ An, Hải Phòng;

– Từ Đà Nẵng: có một chuyến đến Hải Phòng, hai chuyến bay đến TP.HCM, một chuyến đến Hà Nội;

– Từ Nghệ An: có một chuyến đến Hà Nội, một chuyến đến TP.HCM;

– Từ Hải Phòng: có một chuyến đến Hà Nội, một chuyến đến TP.HCM, và một chuyến đến Đà Nẵng;

– Từ Phú Quốc: có một chuyến đến TP.HCM.

a) Vẽ đồ thị biểu diễn các thành phố có chuyến bay giữa chúng (không quan tâm đến số lượng các chuyến bay).

b) Từ đồ thị đã vẽ được trong câu a). Hãy biểu diễn đồ thị bằng hai cách:

– Ma trận kề.

– Danh sách kề.

BÀI 3.3 DUYỆT ĐỒ THỊ THEO CHIỀU RỘNG

MỤC TIÊU **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 một đồ thị cụ thể cho bằng biểu diễn trực quan.

KHỞI ĐỘNG

Bản đồ giao thông kết nối 8 địa điểm nổi tiếng được mô tả như đồ thị G_1 (*Hình 1*). Theo em, có tồn tại một hành trình đi từ địa điểm D đến địa điểm G sao cho phải đi qua ít địa điểm trung gian nhất không? Chỉ ra hành trình đó.

Hình 1. Đồ thị G_1

KHÁM PHÁ

1. Duyệt đồ thị theo chiều rộng

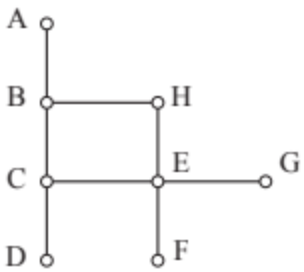
Duyệt đồ thị là quá trình đi đến tất cả các đỉnh của đồ thị và mọi đỉnh chỉ được duyệt đúng một lần. Tuỳ theo quá trình đi đến tất cả các đỉnh mà thứ tự duyệt (xử lí) của các đỉnh là khác nhau. Có hai phép duyệt đồ thị là duyệt đồ thị theo chiều rộng (BFS - Breadth First Search) và duyệt đồ thị theo chiều sâu (DFS - Depth First Search).

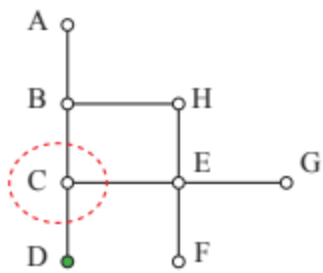
Duyệt đồ thị theo chiều rộng bắt đầu từ đỉnh u chưa được duyệt. Duyệt đỉnh u, sau đó duyệt đến các đỉnh v mà chiều dài đường đi từ đỉnh u đến đỉnh v (số cạnh của đường đi) lần lượt là 1, 2, 3,… cho đến khi không tồn tại đường đi từ đỉnh u đến đỉnh v mà đỉnh v chưa được duyệt. Lặp lại cách duyệt này cho đến khi tất cả các đỉnh của đồ thị đã được duyệt.

Duyệt đồ thị theo chiều rộng có thể được thực hiện bằng cách sử dụng hàng đợi. Giả sử quá trình duyệt bắt đầu từ đỉnh u chưa được duyệt. Ban đầu, hàng đợi chỉ có đỉnh u. Sau đó, lấy đỉnh u ra khỏi hàng đợi, duyệt đỉnh u và thêm các đỉnh kề chưa duyệt của đỉnh u vào hàng đợi. Lặp lại quá trình lấy ra và thêm vào hàng đợi cho đến khi hàng đợi rỗng.

Ví dụ: Xét đồ thị G_1 ở *Hình 1*, nếu duyệt đồ thị này theo chiều rộng bắt đầu từ đỉnh D, em sẽ thực hiện từng bước như *Bảng 1*:

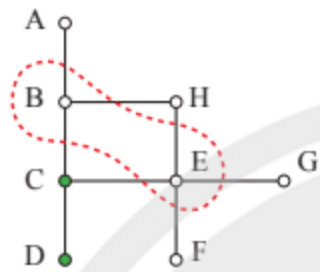
Bảng 1. Minh hoạ duyệt đồ thị theo chiều rộng

Đồ thị G_1	Hàng đợi Queue
	❶ Duyệt đỉnh D và thêm đỉnh D vào hàng đợi 



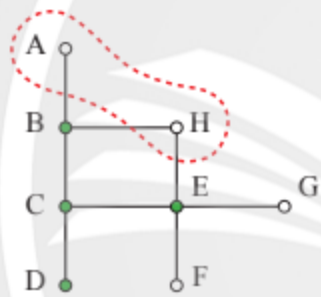
2 Lấy đỉnh D ra khỏi hàng đợi, đỉnh C kề với D chưa được duyệt. Duyệt đỉnh C và thêm C vào hàng đợi.

C								



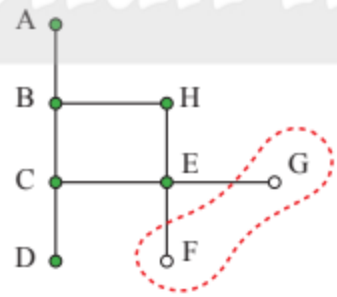
3 Lấy đỉnh C ra khỏi hàng đợi, các đỉnh kề với C chưa được duyệt là B và E. Duyệt B và E, thêm B và E vào hàng đợi.

B	E							



4 Lấy đỉnh B ra khỏi hàng đợi, các đỉnh kề với B chưa được duyệt là A và H. Duyệt A và H, thêm A và H vào hàng đợi.

E	A	H						



5a Lấy đỉnh E ra khỏi hàng đợi, có đỉnh F và G kề với E chưa được duyệt. Duyệt F và G, thêm F và G vào hàng đợi.

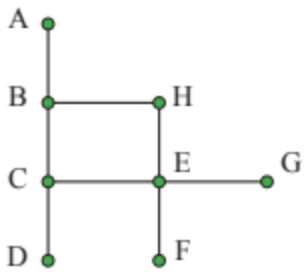
A	H	F	G					

5b Lấy đỉnh A ra khỏi hàng đợi, không có đỉnh nào kề với A chưa được duyệt.

H	F	G						

5c Lấy đỉnh H ra khỏi hàng đợi, không có đỉnh nào kề với H chưa được duyệt.

F	G							



6a Lấy đỉnh F ra khỏi hàng đợi, không có đỉnh nào kề với F chưa được duyệt.

G								

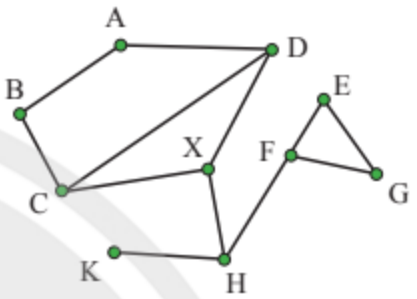
6b Lấy đỉnh G ra khỏi hàng đợi, không có đỉnh nào kề với G chưa được duyệt.

6c Hàng đợi rỗng. Tất cả các đỉnh của đồ thị đã được duyệt xong. Thứ tự duyệt là D, C, B, E, A, H, F, G.

Nhận xét: Cách duyệt đồ thị theo chiều rộng như trên cũng cho em cách đi từ đỉnh D tới đỉnh G mà phải qua ít đỉnh trung gian nhất là:

(D, C, E, G).

Hãy cho biết thứ tự duyệt các đỉnh với phương pháp duyệt đồ thị theo chiều rộng xuất phát từ đỉnh X của đồ thị G_2 ở *Hình 2*.



Hình 2. Đồ thị G_2

2. Thuật toán duyệt đồ thị theo chiều rộng

Thuật toán duyệt đồ thị theo chiều rộng sử dụng hàng đợi để lưu các đỉnh trong quá trình duyệt. Giả sử quá trình duyệt bắt đầu từ một đỉnh u chưa được duyệt của đồ thị. Ban đầu, duyệt đỉnh u và thêm đỉnh u vào hàng đợi. Sau đó, lấy đỉnh u ra khỏi hàng đợi, duyệt các đỉnh kề của đỉnh u mà chúng chưa được duyệt và thêm các đỉnh này vào hàng đợi. Lặp lại quá trình lấy ra và thêm vào hàng đợi cho đến khi hàng đợi rỗng. Thuật toán được mô tả như sau:

Thuật toán duyệt đồ thị theo chiều rộng:

```
#Duyệt đồ thị G bắt đầu từ đỉnh u
def bft (G, u):
    Tạo hàng đợi Q rỗng
    Đánh dấu đỉnh u đã duyệt
    Thêm đỉnh u vào hàng đợi Q
    while hàng đợi Q khác rỗng:
        Lấy đỉnh u từ hàng đợi Q
        Xử lí đỉnh u
        #Thêm các đỉnh kề v của đỉnh u vào hàng đợi Q
        for đỉnh v thuộc tập đỉnh kề của đỉnh u:
            if đỉnh v chưa duyệt:
                Đánh dấu đỉnh v đã duyệt
                Thêm đỉnh v vào hàng đợi Q
```

Khi đó, thủ tục thực hiện duyệt đồ thị G = (V, E) theo chiều rộng như sau:

```
#Duyệt đồ thị G theo chiều rộng
def bfs(G):
  #Đánh dấu các đỉnh của đồ thị G chưa duyệt
  for đỉnh u thuộc đồ thị G:
       Đánh dấu đỉnh u chưa duyệt
  #Duyệt các đỉnh của đồ thị G
  for đỉnh u thuộc của đồ thị G:
    if đỉnh u chưa duyệt:
       bft(G, u)
```

Từ đồ thị G_1 trong *Hình 1*. Hãy thực hiện yêu cầu sau:

a. Dùng thuật toán duyệt đồ thị theo chiều rộng để tìm đường đi ngắn nhất từ đỉnh C đến tất cả các đỉnh của đồ thị.

b. Từ câu a, mô tả cách duyệt cây theo chiều rộng bắt đầu từ đỉnh C.

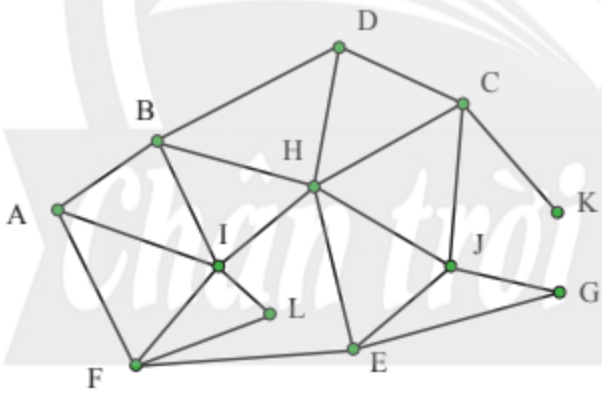
Thuật toán duyệt đồ thị theo chiều rộng sử dụng hàng đợi để lưu các đỉnh trong quá trình duyệt. Giả sử quá trình duyệt bắt đầu từ một đỉnh u chưa được duyệt của đồ thị. Ban đầu, duyệt đỉnh u và thêm đỉnh u vào hàng đợi. Sau đó, lấy đỉnh u ra khỏi hàng đợi, duyệt các đỉnh kề của đỉnh u mà chúng chưa được duyệt và thêm các đỉnh này vào hàng đợi.

LUYỆN TẬP

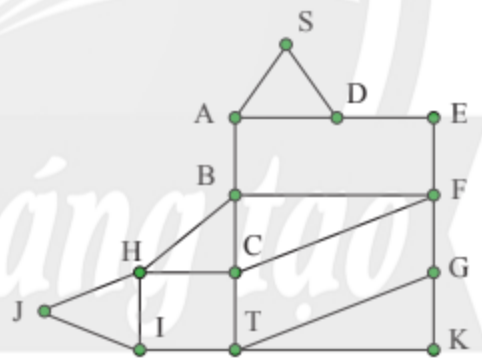
Cho hai đồ thị G_3 (*Hình 3*) và G_4 (*Hình 4*). Dùng thuật toán duyệt đồ thị theo chiều rộng để thực hiện hai yêu cầu sau:

– Liệt kê thứ tự duyệt các đỉnh của đồ thị G_3 xuất phát từ đỉnh A.

– Cho biết đường đi từ đỉnh F đến đỉnh J trong đồ thị G_4?



Hình 3. Đồ thị G_3



Hình 4. Đồ thị G_4

THỰC HÀNH

Chương trình dưới đây duyệt đồ thị G_3 (*Hình 3*) bắt đầu từ đỉnh B và đỉnh F. Em có nhận xét gì về thứ tự duyệt bắt đầu với đỉnh B và đỉnh F.

Chương trình sau được viết bằng Python duyệt đồ thị theo chiều rộng với đồ thị graph được biểu diễn bằng danh sách kề:

```
01. def bft(graph, u):
02.   queue = initQueue()                          #Khởi tạo queue rỗng
03.   visited[vertices.index(u)] = True            #Đánh dấu đỉnh u đã duyệt
04.   enqueue(queue,u)                             #Thêm đỉnh u vào queue
05.   while not isEmptyQueue (queue):              #queue khác rỗng
06.     u = dequeue(queue)                         #Lấy đỉnh u ra khỏi queue
07.     print(u, end = " ")                        #Xử lí đỉnh u
08.     for v in graph[u]:                         #Xét đỉnh kề v của đỉnh u
09.       if not visited[vertices.index(v)]:       #Đỉnh v chưa duyệt
10.         visited[vertices.index(v)] = True      #Đánh dấu đỉnh v đã duyệt
11.         enqueue(queue,v)                       #Thêm đỉnh v vào queue
12. #Hàm duyệt graph dạng danh sách kề theo chiều rộng
13. def bfs(graph):
14.   global visited
15.   visited = [False] * len(graph)               #Đánh dấu các đỉnh chưa duyệt
16.   for u in graph:                              #Xét đỉnh u
17.     if not visited[vertices.index(u)]:         #Đỉnh u chưa duyệt
18.       bft(graph, u)                            #Duyệt đồ thị từ đỉnh u
19. graph, vertices = createAdjListGraph('dothi.txt')   #Tạo đồ thị từ tệp
20. bfs(graph)
```

VẬN DỤNG

Nhiệm vụ. Viết chương trình đếm số nước liên minh với nước đã cho

Yêu cầu: Có N nước, các nước được chia thành các liên minh. Quan hệ liên minh như sau: nếu nước A liên minh với nước B, nước B liên minh với nước C thì nước A liên minh với nước C. Cho biết nước X, sử dụng thuật toán duyệt đồ thị theo chiều rộng, hãy cho biết có bao nhiêu nước liên minh với nước X.

Dữ liệu vào: Tệp lienminh.txt chứa dữ liệu của các nước. Hàng đầu tiên là danh sách các nước. Các hàng kế tiếp: mỗi hàng chứa một cạnh gồm hai nước liên minh. Hàng cuối cùng là nước X.

Dữ liệu ra: Số nước liên minh với nước X.

Ví dụ:

Dữ liệu vào	Dữ liệu ra
0 1 2 3 4 0 1 0 2 3 4 2	2

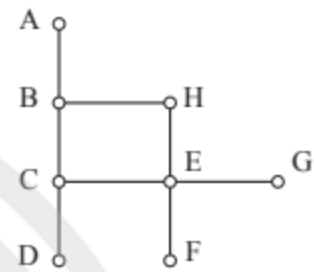
BÀI 3.4 DUYỆT ĐỒ THỊ THEO CHIỀU SÂU

MỤC TIÊU **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 một đồ thị cụ thể bằng biểu diễn trực quan.

KHỞI ĐỘNG

Cho đồ thị G_1 như ở *Hình 1*. Hãy tìm đường đi ngắn nhất từ đỉnh H đến đỉnh D bằng thuật toán duyệt đồ thị theo chiều rộng.



Hình 1. Đồ thị G_1

KHÁM PHÁ

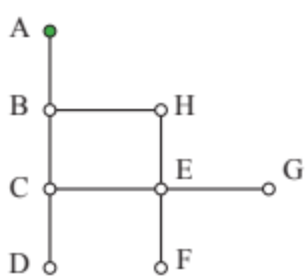
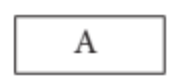
1. Duyệt đồ thị theo chiều sâu

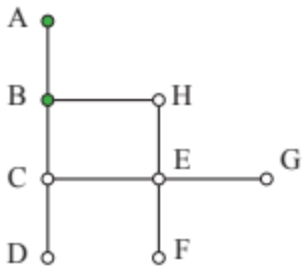
Duyệt đồ thị theo chiều sâu (DFS - Depth First Search) bắt đầu từ việc "***duyệt theo chiều sâu từ đỉnh u***" chưa được duyệt: duyệt đỉnh u, sau đó lần lượt xét các đỉnh kề v của đỉnh u, nếu có đỉnh v chưa được duyệt thì "***duyệt theo chiều sâu từ đỉnh v***", ngược lại quay lui về bước duyệt trước đó.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

Duyệt đồ thị theo chiều sâu có thể được thực hiện bằng cách sử dụng ngăn xếp. Giả sử quá trình duyệt bắt đầu từ đỉnh u chưa được duyệt. Khởi tạo ngăn xếp rỗng, sau đó duyệt đỉnh u và thêm đỉnh u vào ngăn xếp. Xem đỉnh p ở đỉnh ngăn xếp, nếu có đỉnh kề v chưa duyệt của đỉnh p thì duyệt đỉnh v và thêm đỉnh v vào ngăn xếp, ngược lại, lấy đỉnh p ra khỏi ngăn xếp. Lặp lại quá trình lấy ra và thêm vào ngăn xếp cho đến khi ngăn xếp rỗng.

Ví dụ: Xét đồ thị G_1 ở *Hình 1*, duyệt đồ thị này theo chiều sâu bắt đầu từ đỉnh A, em sẽ thực hiện từng bước như *Bảng 1*. Các đỉnh được ghi trong ngoặc đơn là các đỉnh đã duyệt.

Bảng 1. Minh hoạ duyệt đồ thị theo chiều sâu

Đồ thị G_1	Ngăn xếp Stack
	❶ Duyệt đỉnh A, thêm đỉnh A vào ngăn xếp.   Stack



2 Xem đỉnh A ở đỉnh ngăn xếp. Đỉnh kề B của đỉnh A chưa duyệt. Duyệt đỉnh B và thêm đỉnh này vào ngăn xếp.

A	B							đã duyệt

B
A

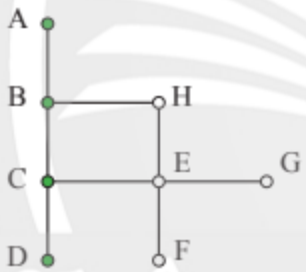
Stack

3 Xem đỉnh B ở đỉnh ngăn xếp. Đỉnh kề C của đỉnh B chưa duyệt. Duyệt đỉnh C và thêm đỉnh này vào ngăn xếp.

A	B	C						đã duyệt

C
B
A

Stack



4a Xem đỉnh C ở đỉnh ngăn xếp. Đỉnh kề D của đỉnh C chưa duyệt. Duyệt đỉnh D và thêm đỉnh này vào ngăn xếp.

A	B	C	D					đã duyệt

D
C
B
A

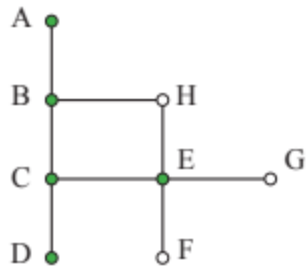
Stack

4b Xem đỉnh D ở đỉnh ngăn xếp. Đỉnh D không có đỉnh kề chưa duyệt. Lấy đỉnh D ra khỏi ngăn xếp.

A	B	C	D					đã duyệt

C
B
A

Stack

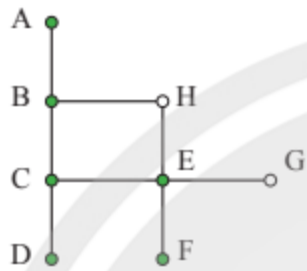


5 Xem đỉnh C ở đỉnh ngăn xếp. Đỉnh kề E của đỉnh C chưa duyệt. Duyệt đỉnh E và thêm đỉnh này vào ngăn xếp.

A	B	C	D	E				đã duyệt

E
C
B
A

Stack



6a Xem đỉnh E ở đỉnh ngăn xếp. Đỉnh kề F của đỉnh E chưa duyệt. Duyệt đỉnh F và thêm đỉnh này vào ngăn xếp.

A	B	C	D	E	F			đã duyệt

F
E
C
B
A

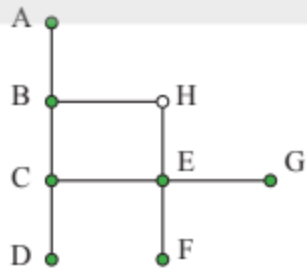
Stack

6b Xem đỉnh F ở đỉnh ngăn xếp. Đỉnh F không có đỉnh kề chưa duyệt. Lấy đỉnh F ra khỏi ngăn xếp.

A	B	C	D	E	F			đã duyệt

E
C
B
A

Stack

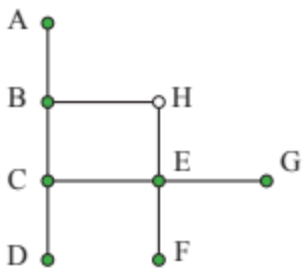


7a Xem đỉnh E ở đỉnh ngăn xếp. Đỉnh kề G của đỉnh E chưa duyệt. Duyệt đỉnh G và thêm đỉnh này vào ngăn xếp.

A	B	C	D	E	F	G		đã duyệt

G
E
C
B
A

Stack

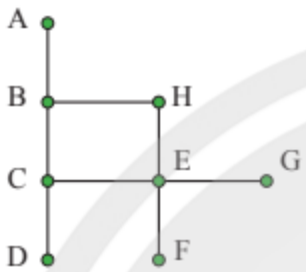


7b Xem đỉnh G ở đỉnh ngăn xếp. Đỉnh G không có đỉnh kề chưa duyệt. Lấy đỉnh G ra khỏi ngăn xếp.

A	B	C	D	E	F	G		đã duyệt

E
C
B
A

Stack

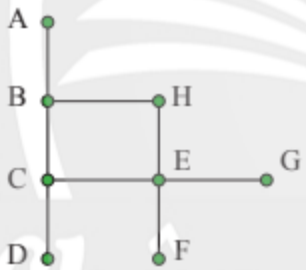


8 Xem đỉnh E ở đỉnh ngăn xếp. Đỉnh kề H của đỉnh E chưa duyệt. Duyệt đỉnh H và thêm đỉnh này vào ngăn xếp..

A	B	C	D	E	F	G	H	đã duyệt

H
E
C
B
A

Stack



9a Xem đỉnh H ở đỉnh ngăn xếp. Đỉnh H không có đỉnh kề chưa duyệt. Lấy đỉnh H ra khỏi ngăn xếp.

A	B	C	D	E	F	G	H	đã duyệt

E
C
B
A

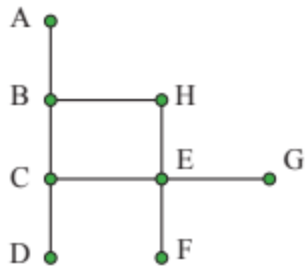
Stack

9b Xem đỉnh E ở đỉnh ngăn xếp. Đỉnh E không có đỉnh kề chưa duyệt. Lấy đỉnh E ra khỏi ngăn xếp.

A	B	C	D	E	F	G	H	đã duyệt

C
B
A

Stack



9c Xem đỉnh C ở đỉnh ngăn xếp. Đỉnh C không có đỉnh kề chưa duyệt. Lấy đỉnh C ra khỏi ngăn xếp.

A	B	C	D	E	F	G	H	đã duyệt

B
A

Stack

9d Xem đỉnh B ở đỉnh ngăn xếp. Đỉnh B không có đỉnh kề chưa duyệt. Lấy đỉnh B ra khỏi ngăn xếp.

A	B	C	D	E	F	G	H	đã duyệt

A

Stack

9e Xem đỉnh A ở đỉnh ngăn xếp. Đỉnh A không có đỉnh kề chưa duyệt. Lấy đỉnh A ra khỏi ngăn xếp.

9g Ngăn xếp rỗng. Kết thúc. Thứ tự duyệt đồ thị theo chiều sâu là A, B, C, D, E, F, G, H.

A	B	C	D	E	F	G	H	đã duyệt

Em hãy minh hoạ duyệt theo chiều sâu của đồ thị G_2 ở *Hình 2* (tương tự như *Bảng 1*) bắt đầu từ đỉnh 2.

Hình 2. Đồ thị G_2

Duyệt đồ thị theo chiều sâu bắt đầu từ việc duyệt theo chiều sâu đỉnh u chưa được duyệt. Duyệt đỉnh u, sau đó lần lượt xét các đỉnh kề v của đỉnh u, nếu đỉnh v chưa được duyệt thì duyệt theo chiều sâu bắt đầu từ đỉnh v. Lặp lại cách duyệt này cho đến khi tất cả các đỉnh của đồ thị đã được duyệt.

2. Thuật toán duyệt đồ thị theo chiều sâu

Thuật toán duyệt đồ thị theo chiều sâu sử dụng kĩ thuật đệ quy bắt đầu từ đỉnh u chưa được duyệt được minh họa như sau.

```
01. def dfs(G, u):
02.    Xử lí đỉnh u
03.    Đánh dấu duyệt đỉnh u
04.    for đỉnh v là đỉnh kề của đỉnh u:
```

```
05.       if đỉnh v chưa được đánh dấu duyệt:
06.             dfs(G, v)
```

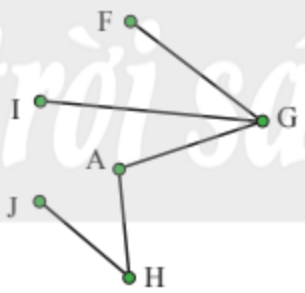
Thuật toán duyệt đồ thị theo chiều sâu bắt đầu từ đỉnh u:

```
01. def dft(G, u):
02.   Khởi tạo ngăn xếp stack rỗng
03.   Xử lí đỉnh u
04.   Đánh dấu duyệt đỉnh u
05.   Thêm đỉnh u vào ngăn xếp stack
06.   while ngăn xếp stack khác rỗng:
07.     Xem đỉnh p ở đầu ngăn xếp stack
08.     #Xét các đỉnh kề v chưa được duyệt của đỉnh p
09.     found = False
10.     for đỉnh v thuộc tập đỉnh kề của đỉnh p:
11.       if đỉnh v chưa duyệt:
12.         found=True
13.         break
14.     if not found:
15.       Lấy đỉnh p ra khỏi ngăn xếp stack
16.     else:
17.       Xử lí đỉnh v
18.       Đánh dấu duyệt đỉnh v
19.       Thêm đỉnh v vào ngăn xếp stack
```

Thuật toán duyệt theo chiều sâu các đỉnh của đồ thị G được minh hoạ như sau:

```
01. def dfs(G):
02.  for đỉnh u thuộc G:
03.     Đánh dấu đỉnh u chưa duyệt
04.
05.  for đỉnh u thuộc G:
06.      if đỉnh u chưa duyệt:
07.        dft(G, u)
```

Hãy liệt kê thứ tự duyệt các đỉnh với thuật toán duyệt đồ thị theo chiều sâu xuất phát từ đỉnh A của đồ thị G_3 ở *Hình 3*.

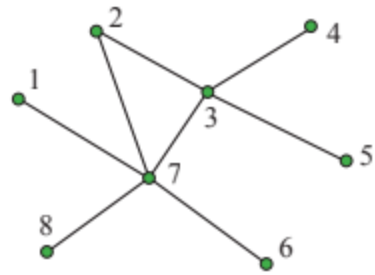


Hình 3. Đồ thị G_3

Thuật toán duyệt đồ thị theo chiều sâu (DFS) sử dụng ngăn xếp để lưu các đỉnh trong quá trình duyệt. Bắt đầu từ đỉnh u chưa được duyệt. Khởi tạo ngăn xếp rỗng. Duyệt đỉnh u và thêm đỉnh u vào ngăn xếp. Sau đó, xem đỉnh p ở đỉnh ngăn xếp, nếu có đỉnh kề v của đỉnh p chưa duyệt thì duyệt đỉnh v và thêm đỉnh v vào ngăn xếp, ngược lại thì lấy đỉnh p ra khỏi ngăn xếp. Lặp lại quá trình lấy ra và thêm vào ngăn xếp cho đến khi ngăn xếp rỗng.

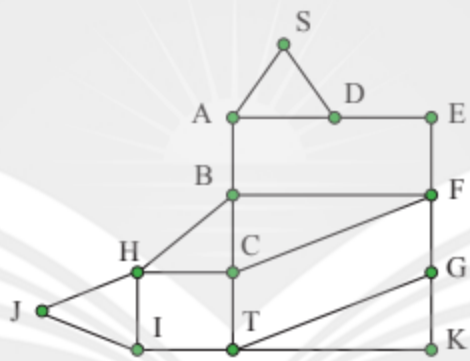
LUYỆN TẬP

1. Dùng thuật toán duyệt đồ thị theo chiều sâu xuất phát từ đỉnh 1. Hãy cho biết thứ tự duyệt các đỉnh của đồ thị ở *Hình 4*.



Hình 4. Đồ thị G_4

2. Cho đồ thị G_5 (*Hình 5*). Chỉ ra đường đi từ đỉnh F đến đỉnh J bằng thuật toán duyệt đồ thị theo chiều sâu trong đồ thị G_5.



Hình 5. Đồ thị G_5

THỰC HÀNH

Nhiệm vụ. Duyệt đồ thị theo chiều sâu

Yêu cầu: Chương trình sau được viết bằng Python duyệt đồ thị theo chiều sâu, với đồ thị graph được biểu diễn bằng danh sách kề.

```
01. def dft(graph, u):
02.   stack = initStack()              #Khởi tạo stack rỗng
03.   visited[vertices.index(u)] = True  #Đánh dấu đỉnh u đã duyệt
04.   print(u, end = " ")               #In đỉnh u
05.   push(stack, u)                   #Thêm đỉnh u vào stack
06.   while not isEmptyStack(stack):  #Lặp khi stack khác rỗng
07.     p = top(stack)                 #Xem đỉnh p ở đỉnh stack
08.     found = False                  #Chưa tìm thấy
09.     for v in graph[p]:             #Lặp để lấy các đỉnh kề v của đỉnh p
```

```
10.       if not visited[vertices.index(v)]: #Nếu đỉnh v chưa duyệt
11.         found = True                     #Tìm thấy
12.         break
13.     if not found:                        #Không tìm thấy đỉnh v
14.       p = pop(stack)                     #Lấy đỉnh p ra khỏi stack
15.     else:                                #Tìm thấy đỉnh v chưa duyệt
16.       visited[vertices.index(v)] = True  #Đánh dấu đỉnh v đã duyệt
17.       print(v, end = " ")                #In đỉnh v
18.       push(stack, v)                     #Thêm đỉnh v vào stack
19. #Hàm duyệt graph dạng danh sách kề theo chiều sâu
20. def dfs(graph):
21.   global visited
22.   visited = [False] * len(graph)         #Đánh dấu các đỉnh chưa duyệt
23.   for u in graph:
24.     if not visited[vertices.index(u)]:   #Xét đỉnh u chưa duyệt
25.         dft(graph, u)                    #Duyệt đồ thị theo chiều sâu từ u
26. graph, vertices = createAdjListGraph('dothi.txt')  #Tạo đồ thị từ tệp
27. dfs(graph)                               #In kết quả duyệt theo chiều sâu
```

a) Em hãy biểu diễn đồ thị G_4 ở *Hình 4* thành danh sách kề.

b) Chạy chương trình trên để duyệt đồ thị G_4 được biểu diễn bằng danh sách kề.

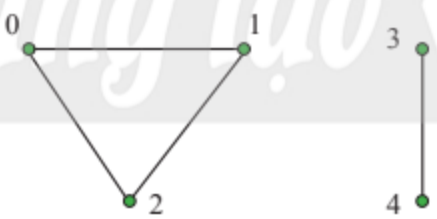
c) Chạy chương trình trên để duyệt đồ thị G_4 được biểu diễn bằng danh sách kề, bắt đầu từ đỉnh 3.

VẬN DỤNG

Nhiệm vụ. Kiểm tra đồ thị liên thông

Một đồ thị được gọi là liên thông nếu tồn tại ít nhất một đường đi giữa hai đỉnh bất kì của nó. Chẳng hạn, đồ thị ở *Hình 6a* là liên thông còn đồ thị ở *Hình 6b* là không liên thông (không có đường đi từ đỉnh 0 tới đỉnh 3).

Hình 6a. Minh hoạ đồ thị liên thông



Hình 6b. Minh hoạ đồ thị không liên thông

Hình 6. Minh hoạ kiểm tra đồ thị liên thông

Yêu cầu: Áp dụng thuật toán duyệt đồ thị theo chiều sâu. Thực hiện xây dựng thuật toán kiểm tra xem đồ thị G = (V, E) cho trước có liên thông hay không.

BÀI 3.5 THỰC HÀNH KĨ THUẬT DUYỆT ĐỒ THỊ

MỤC TIÊU **Sau bài học này, em sẽ:**

Biết ứng dụng thuật toán duyệt đồ thị theo chiều rộng và duyệt đồ thị theo chiều sâu để giải một số bài toán thực tế.

KHỞI ĐỘNG

Nêu một ứng dụng của một trong hai thuật toán duyệt đồ thị đã học.

THỰC HÀNH

Nhiệm vụ 1. Chương trình tìm đường đi bằng thuật toán duyệt đồ thị theo chiều rộng

Yêu cầu: Cho đồ thị vô hướng G. Hãy viết chương trình tìm đường đi từ đỉnh u đến đỉnh v bằng thuật toán duyệt đồ thị theo chiều rộng. Đồ thị G được biểu diễn bằng danh sách kề.

Dữ liệu vào:

– Tệp dothi.txt chứa dữ liệu của đồ thị. Hàng đầu tiên là danh sách các đỉnh của đồ thị. Các hàng kế tiếp: mỗi hàng chứa một cung gồm đỉnh gốc và đỉnh ngọn.

– Đỉnh u và đỉnh v của đường đi.

Dữ liệu ra:

– Nếu có đường đi từ đỉnh u đến đỉnh v thì hiển thị các đỉnh của đường đi này.

– Nếu không có đường đi thì hiển thị "Không có đường đi".

Hướng dẫn:

Sử dụng thuật toán duyệt đồ thị theo chiều rộng, bắt đầu từ đỉnh u. Em xây dựng mảng một chiều before với giá trị mặc định của các phần tử là –1 để lưu lại các đỉnh trong quá trình duyệt với quy ước: before[i] = j nghĩa là duyệt đỉnh j trước rồi duyệt đỉnh i sau.

Chương trình tìm đường đi sử dụng thuật toán duyệt đồ thị theo chiều rộng:

```
01. def initQueue():
02.     return []
03. def isEmptyQueue(queue):
04.     return len(queue) == 0
05. def enqueue(queue, val):
06.     queue.append(val)
07. def dequeue(queue):
08.     return queue.pop(0)
09. #Hàm xuất đường đi
10. def printPath(path, u, v):
11.   if path == None:
12.     print("Không có đường đi từ đỉnh", u, "đến đỉnh", v)
13.   else:
14.     print("Đường đi từ đỉnh", u, "đến đỉnh", v, "là:")
15.     for v in path:
16.       print(v, end = " ")
```

```
#Hàm tạo đường đi
def createPath(u, v):
  path = []
  j = v
  path.append(j)
  while before [vertices.index(j)] != u:
      path.append(before[vertices.index(j)])
      j = before[vertices.index(j)]
  path.append(u)
  path.reverse()
  return path

#Tìm đường đi từ đỉnh u đến đỉnh v trong đồ thị dùng BFS
#Hàm tìm đường đi giữa u và v sử dụng BFS
def findPathBFS(graph, u, v):
  queue = initQueue()          #Khởi tạo hàng đợi queue
  enqueue(queue, u)            #Thêm đỉnh u vào queue và đánh dấu đã duyệt
  visited[vertices.index(u)] = True
  #Lặp cho đến khi queue rỗng
  while not isEmptyQueue(queue):
    p = dequeue(queue)
    #Nếu đỉnh kề với p là v thì trả kết quả đường đi từ u đến v. Ngược lại,
thêm các đỉnh kề với p vào queue
    for neighbor in graph[p]:
      if neighbor == v:
        before[vertices.index(neighbor)] = p
        return createPath(u, v)
      elif not visited[vertices.index(neighbor)]:
        visited[vertices.index(neighbor)] = True
        enqueue(queue, neighbor)
        before[vertices.index(neighbor)] = p
  #Nếu không tìm thấy đường đi từ u đến v, trả về None.
  if before[vertices.index(v)] == -1:
    return None
#Hàm tìm đường đi từ đỉnh u đến đỉnh v
def findPath(graph, u, v):
  if not u in graph:
    print("Không có đỉnh", u)
    return
  elif not v in graph:
    print("Không có đỉnh ", v)
    return
  global visited, before
  visited = [False] * len(graph)
  before = [-1] * len(graph)
  path = findPathBFS(graph, u, v)
  return path

graph, vertices = createAdjListGraph('dothi.txt') #Tạo đồ thị dạng danh sách kề từ tệp
u, v = list(map(str, input().split()))
path = findPath(graph, u, v)
print(path)
printPath(path, u, v)
```

Với dữ liệu vào là tệp dothi.txt tương ứng với đồ thị G_1 ở *Hình 1* trong Bài 3.2, chương trình tìm đường đi từ đỉnh 0 đến đỉnh 5 có kết quả là:

```
0 5
['0', '2', '5']
Đường đi từ đỉnh 0 đến đỉnh 5 là:
0 2 5
```

Nhiệm vụ 2. Chương trình tìm đường đi bằng thuật toán duyệt đồ thị theo chiều sâu

Yêu cầu: Cho đồ thị G. Hãy viết chương trình tìm đường đi từ đỉnh u đến đỉnh v bằng thuật toán duyệt đồ thị theo chiều sâu. Đồ thị G được biểu diễn bằng danh sách kề.

Dữ liệu vào:

– Tệp dothi.txt chứa dữ liệu của đồ thị. Hàng đầu tiên là danh sách các đỉnh của đồ thị. Các hàng kế tiếp: mỗi hàng chứa một cung gồm đỉnh gốc và đỉnh ngọn.

– Đỉnh u và đỉnh v của đường đi.

Dữ liệu ra:

– Nếu có đường đi từ đỉnh u đến đỉnh v thì hiển thị các đỉnh của đường đi này.

– Nếu không có đường đi thì hiển thị "Không có đường đi".

Hướng dẫn:

Sử dụng thuật toán duyệt đồ thị theo chiều sâu để tiến hành duyệt tất cả các đỉnh mà u có thể liên kết tới trong đồ thị. Em xây dựng mảng một chiều before với giá trị mặc định của các phần tử là –1 để lưu lại các đỉnh trong quá trình duyệt với quy ước: before[i] = j nghĩa là duyệt đỉnh j trước rồi duyệt đến đỉnh i.

Chương trình tìm đường đi sử dụng thuật toán duyệt đồ thị theo chiều sâu:

```
01. def initStack():
02.     return []
03. def isEmptyStack(stack):
04.     return len(stack) == 0
05. def push(stack, val) :
06.     stack.append(val)
07. def pop(stack):
08.     return stack.pop()
09. def top(stack):
10.     return stack[len(stack)-1]
11.
12. #Tìm đường đi từ đỉnh u đến đỉnh v trong đồ thị dùng DFS
13. #Hàm tìm đường đi giữa u và v sử dụng DFS
14. def findPathDFS(graph, u, v):
15.     stack = initStack() #Khởi tạo ngăn xếp stack
16.     push(stack, u)      #Thêm đỉnh u vào stack và đánh dấu đã duyệt
17.     visited[vertices.index(u)] = True
18.
19.     #Lặp cho đến khi stack rỗng
20.     while not isEmptyStack(stack):
21.         p = top(stack)
22.         found = False
23.         for neighbor in graph[p]:
24.             if not visited[vertices.index(neighbor)]:
25.                 found = True
26.                 break
27.         if found:
28.             visited[vertices.index(neighbor)] = True
29.             before[vertices.index(neighbor)] = p
30.             if neighbor == v:
31.                 return createPath(u, v)
32.             else:
33.                 push(stack, neighbor)
34.         else:
```

```
        p = pop(stack)
    #Nếu không tìm thấy đường đi từ u đến v, trả về None.
    if before[vertices.index(v)] == -1:
      return None

#Hàm tìm đường đi từ đỉnh u đến đỉnh v
def findPath(graph, u, v):
    if not u in graph:
      print("Không có đỉnh", u)
      return
    elif not v in graph:
      print("Không có đỉnh ", v)
      return
    global visited, before
    visited = [False] * len(graph)
    before = [-1] * len(graph)
    path = findPathDFS(graph, u, v)
    return path

#Ví dụ minh hoạ
graph, vertices = createAdjListGraph('dothi.txt') #Tạo đồ thị dạng danh sách kề từ tệp
u, v = list(map(str, input().split()))
path = findPath(graph, u, v)
print(path)
printPath(path, u, v)
```

Với dữ liệu vào là tệp dothi.txt tương ứng với đồ thị G_1 ở *Hình 1* trong Bài 3.2, chương trình tìm đường đi từ đỉnh 0 đến đỉnh 5 có kết quả là:

```
0 5
['0', '1', '2', '3', '4', '5']
Đường đi từ đỉnh 0 đến đỉnh 5 là:
0 1 2 3 4 5
```

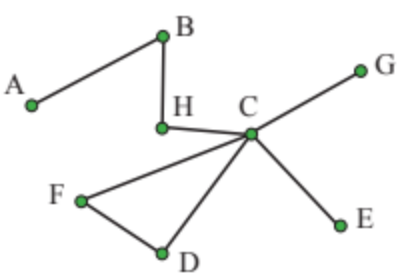
LUYỆN TẬP

Nhiệm vụ 1. Tìm đường đi bằng thuật toán duyệt đồ thị theo chiều rộng

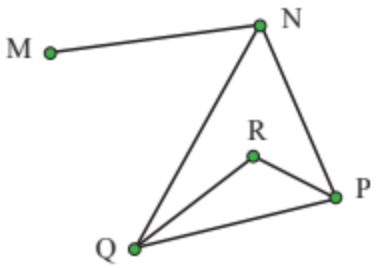
Yêu cầu: Bản đồ mô tả đường đi giữa các địa điểm du lịch trong một thành phố được biểu diễn như ở *Hình 1*. Dựa vào thuật toán duyệt đồ thị theo chiều rộng được biểu diễn bằng ma trận kề, em hãy viết chương trình tìm đường đi từ địa điểm A đến địa điểm D sao cho đi qua ít địa điểm nhất.

Dữ liệu vào: Tệp dothi.txt chứa dữ liệu của đồ thị. Hàng đầu tiên là danh sách các đỉnh của đồ thị. Các hàng kế tiếp: mỗi hàng chứa một cung gồm đỉnh gốc và đỉnh ngọn.

Dữ liệu ra: Các đỉnh của đường đi từ đỉnh A đến đỉnh D.



Hình 1. Bản đồ thành phố



Hình 2. Địa điểm du lịch

Nhiệm vụ 2. Tìm đường đi bằng thuật toán duyệt đồ thị theo chiều sâu

Yêu cầu: Cho bản đồ (*Hình 2*) gồm các thành phố M, N, P, Q, R được biểu diễn bởi đồ thị. Dựa vào thuật toán duyệt đồ thị theo chiều sâu được biểu diễn bằng ma trận kề, viết chương trình in ra màn hình đường đi từ thành phố M đến thành phố R.

Dữ liệu vào: Tệp dothi.txt chứa dữ liệu của đồ thị. Hàng đầu tiên là danh sách các đỉnh của đồ thị. Các hàng kế tiếp: mỗi hàng chứa một cung gồm đỉnh gốc và đỉnh ngọn.

Dữ liệu ra: Các đỉnh của đường đi từ đỉnh M đến đỉnh R.

VẬN DỤNG

Nhiệm vụ. Đếm số thành phần liên thông của đồ thị

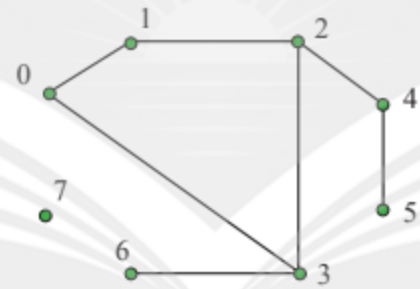
Một đồ thị G được gọi là không liên thông nếu tồn tại đỉnh u và đỉnh v thuộc G mà không có đường đi giữa hai đỉnh này. Khi đó, đỉnh u và đỉnh v thuộc hai thành phần liên thông khác nhau. Nếu tồn tại đường đi giữa đỉnh u và đỉnh v thì hai đỉnh này phải thuộc cùng một thành phần liên thông. Như vậy, đồ thị G không liên thông sẽ có ít nhất hai thành phần liên thông.

Yêu cầu: Cho đồ thị vô hướng G được biểu diễn bằng danh sách kề. Hãy viết chương trình cho biết số thành phần liên thông của đồ thị G.

Dữ liệu vào: Tệp dothi.txt chứa dữ liệu của đồ thị G. Hàng đầu tiên là danh sách các đỉnh của đồ thị. Các hàng kế tiếp: mỗi hàng chứa một cạnh gồm hai đỉnh.

Dữ liệu ra: Số thành phần liên thông của đồ thị G.

Ví dụ: Cho đồ thị ở *Hình 3*:



Hình 3. Đồ thị G_1

Dữ liệu vào và dữ liệu ra tương ứng với đồ thị G_1 như sau:

Dữ liệu vào	Dữ liệu ra
0 1 2 3 4 5 6 7 1 0 0 1 0 3 3 0 1 2 2 1 2 3 3 2 2 4 4 2 3 6 6 3 4 5 5 4	2

BẢNG GIẢI THÍCH THUẬT NGỮ

Thuật ngữ	Giải thích	Trang
Duyệt đồ thị theo chiều rộng (BFS - Breadth First Search)	Duyệt đồ thị theo chiều rộng bắt đầu từ đỉnh u chưa được duyệt. Duyệt đỉnh u, sau đó duyệt đến các đỉnh v mà chiều dài đường đi từ đỉnh u đến đỉnh v (số cạnh của đường đi) lần lượt là 1, 2, 3,… cho đến khi không tồn tại đường đi từ đỉnh u đến đỉnh v mà đỉnh v chưa được duyệt. Lặp lại cách duyệt này cho đến khi tất cả các đỉnh của đồ thị đã được duyệt.	59
Cấu trúc dữ liệu phi tuyến tính	Cấu trúc dữ liệu phi tuyến tính bao gồm các phần tử không tạo thành một dãy nối tiếp nhau. Mỗi phần tử có thể có nhiều phần tử kề đi sau nó. Chẳng hạn, cấu trúc dữ liệu cây và đồ thị là cấu trúc dữ liệu phi tuyến tính.	26
Cấu trúc dữ liệu tuyến tính	Cấu trúc dữ liệu tuyến tính bao gồm các phần tử tạo thành một dãy nối tiếp nhau và tác vụ duyệt là tuyến tính (tuần tự). Mỗi phần tử có nhiều nhất một phần tử kề đi sau nó. Ví dụ, cấu trúc dữ liệu mảng, hàng đợi, ngăn xếp là cấu trúc dữ liệu tuyến tính.	26
Cây nhị phân	Cây với mỗi nút có tối đa hai cây con (cây con trái và cây con phải).	28
Cây tìm kiếm nhị phân	Cây nhị phân với giá trị khoá của nút bất kì luôn lớn hơn giá trị khoá của tất cả các nút thuộc cây con trái và nhỏ hơn giá trị khoá của tất cả các nút thuộc cây con phải.	41
Duyệt đồ thị theo chiều sâu (DFS - Depth First Search)	Duyệt đồ thị theo chiều sâu 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.	64
Đồ thị	Đồ thị G = (V, E) là một cấu trúc gồm hai tập hợp, trong đó tập V chứa các đỉnh và tập E chứa các cạnh, mỗi cạnh kết nối hai đỉnh của đồ thị với nhau.	49
Duyệt cây nhị phân	Duyệt cây nhị phân là quá trình thăm tất cả nút và mọi nút chỉ được thăm đúng một lần.	32

Duyệt cây nhị phân theo thứ tự duyệt giữa (Duyệt giữa)	Duyệt cây nhị phân với cây con trái được duyệt đầu tiên, sau đó duyệt nút gốc và cuối cùng duyệt cây con phải.	32
Duyệt cây nhị phân theo thứ tự duyệt sau (Duyệt sau)	Duyệt cây nhị phân với cây con trái được duyệt đầu tiên, sau đó duyệt cây con phải và cuối cùng duyệt nút gốc.	32
Duyệt cây nhị phân theo thứ tự duyệt trước (Duyệt trước)	Duyệt cây nhị phân với nút gốc được duyệt đầu tiên, sau đó duyệt cây con trái và cuối cùng duyệt cây con phải.	32
Hàng đợi	Cấu trúc dữ liệu tuyến tính dùng để lưu danh sách các phần tử. Hai thao tác cơ bản trên hàng đợi là: thao tác thêm vào (enqueue) ở cuối hàng đợi (rear) và thao tác lấy ra (dequeue) ở đầu hàng đợi (front). Hai thao tác này thể hiện cơ chế hoạt động "Vào trước – Ra trước" (FIFO – First In, First Out).	5
Ngăn xếp	Cấu trúc dữ liệu tuyến tính dùng để lưu danh sách các phần tử. Hai thao tác cơ bản trên ngăn xếp là: thao tác thêm vào (push) và thao tác lấy ra (pop) đều ở đỉnh ngăn xếp (top). Hai thao tác này thể hiện cơ chế hoạt động "Vào sau – Ra trước" (LIFO – Last In, First Out).	10

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:
Tổng Giám đốc HOÀNG LÊ BÁCH

Chịu trách nhiệm nội dung:
Tổng biên tập PHẠM VĨNH THÁI

Biên tập nội dung: VŨ NHÂN KHÁNH – PHẠM MINH NHẬT
Biên tập mĩ thuật: ĐẶNG NGỌC HÀ – NGUYỄN THỊ THÁI CHÂU
Thiết kế sách: ĐẶNG NGỌC HÀ – LƯU QUANG TIẾN
Trình bày bìa: TỐNG THANH THẢO
Minh hoạ: LƯU QUANG TIẾN
Sửa bản in: MÃ TRƯỜNG VINH – BÙI THANH THUÝ VY
Chế bản: CÔNG TY CỔ PHẦN DỊCH VỤ XUẤT BẢN GIÁO DỤC GIA ĐỊNH

CHUYÊN ĐỀ HỌC TẬP TIN HỌC 12 – Định hướng Khoa học máy tính (Chân trời sáng tạo)

Mã số:

In……bản, (QĐ in số….) Khổ 19x26,5 cm.

Đơn vị in:……..

Cơ sở in:……

Số ĐKXB:

Số QĐXB:..... ngày …. tháng…. năm 20 ...

In xong và nộp lưu chiểu tháng ….năm 20….

Mã số ISBN:

HUÂN CHƯƠNG HỒ CHÍ MINH

BỘ SÁCH GIÁO KHOA LỚP 12 – CHÂN TRỜI SÁNG TẠO

1. Toán 12, Tập một
2. Toán 12, Tập hai
3. Chuyên đề học tập Toán 12
4. Ngữ văn 12, Tập một
5. Ngữ văn 12, Tập hai
6. Chuyên đề học tập Ngữ văn 12
7. Tiếng Anh 12
   Friends Global – Student Book
8. Lịch sử 12
9. Chuyên đề học tập Lịch sử 12
10. Địa lí 12
11. Chuyên đề học tập Địa lí 12
12. Giáo dục kinh tế và pháp luật 12
13. Chuyên đề học tập Giáo dục kinh tế và pháp luật 12
14. Vật lí 12
15. Chuyên đề học tập Vật lí 12
16. Hoá học 12
17. Chuyên đề học tập Hoá học 12
18. Sinh học 12
19. Chuyên đề học tập Sinh học 12
20. Tin học 12 – Định hướng Tin học ứng dụng
21. Chuyên đề học tập Tin học 12 – Định hướng Tin học ứng dụng
22. Tin học 12 – Định hướng Khoa học máy tính
23. Chuyên đề học tập Tin học 12 – Định hướng Khoa học máy tính
24. Âm nhạc 12
25. Chuyên đề học tập Âm nhạc 12
26. Hoạt động trải nghiệm, hướng nghiệp 12 (1)
27. Hoạt động trải nghiệm, hướng nghiệp 12 (2)
28. Giáo dục quốc phòng và an ninh 12

Các đơn vị đầu mối phát hành

- **Miền Bắc:** CTCP Đầu tư và Phát triển Giáo dục Hà Nội
  CTCP Sách và Thiết bị Giáo dục miền Bắc
- **Miền Trung:** CTCP Đầu tư và Phát triển Giáo dục Đà Nẵng
  CTCP Sách và Thiết bị Giáo dục miền Trung
- **Miền Nam:** CTCP Đầu tư và Phát triển Giáo dục Phương Nam
  CTCP Sách và Thiết bị Giáo dục miền Nam
  CTCP Sách và Thiết bị Giáo dục Cửu Long

Sách điện tử: http://hanhtrangso.nxbgd.vn