KẾT NỐI TRI THỨC VỚI CUỘC SỐNG

PHẠM THẾ LONG (Tổng Chủ biên)
BÙI VIỆT HÀ (Chủ biên)
NGUYỄN HOÀNG HÀ – NGUYỄN THỊ HIỀN
TRƯƠNG VÕ HỮU THIÊN – LÊ HỮU TÔN – PHẠM THỊ BÍCH VÂN

CHUYÊN ĐỀ HỌC TẬP TIN HỌC 11

ĐỊNH HƯỚNG KHOA HỌC MÁY TÍNH

NHÀ XUẤT BẢN GIÁO DỤC VIỆT NAM

PHẠM THẾ LONG (Tổng Chủ biên) - BÙI VIỆT HÀ (Chủ biên)
NGUYỄN HOÀNG HÀ - NGUYỄN THỊ HIỀN
TRƯƠNG VÕ HỮU THIÊN - LÊ HỮU TÔN - PHẠM THỊ BÍCH VÂN

CHUYÊN ĐỀ HỌC TẬP
TIN HỌC
11

ĐỊNH HƯỚNG KHOA HỌC MÁY TÍNH

NHÀ XUẤT BẢN GIÁO DỤC VIỆT NAM

Hướng dẫn sử dụng sách

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

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

NỘI DUNG BÀI HỌC

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

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

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

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

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

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

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

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

Lời nói đầu

Các em thân mến!

Các em đang cầm trên tay cuốn sách giáo khoa *Chuyên đề học tập Tin học 11 - Định hướng khoa học máy tính* thuộc bộ sách *Tin học 11 – Kết nối tri thức với cuộc sống*. Cuốn sách này dành cho các em đăng kí học chuyên đề học tập Tin học theo định hướng Khoa học máy tính.

Chuyên đề nhằm cung cấp các kiến thức, kĩ năng Tin học 11 được xác định trong Chương trình Giáo dục phổ thông năm 2018, tập trung vào một số kĩ thuật thiết kế thuật toán cơ bản nhất mà mỗi người làm khoa học máy tính cần phải biết trong tương lai.

Sách bao gồm ba chuyên đề thành phần:

– Chuyên đề 1: Giới thiệu khái niệm đệ quy – kĩ thuật quan trọng đầu tiên của thiết kế thuật toán và chương trình. Bất cứ ai muốn lập nghiệp trong tương lai bằng lập trình đều phải biết và nắm vững kĩ thuật này.

– Chuyên đề 2: Trình bày một trong những phương pháp, kĩ thuật thiết kế thuật toán cơ bản và quan trọng nhất, đó là phương pháp chia để trị.

– Chuyên đề 3: Tìm hiểu kĩ thuật duyệt quay lui – một trong những kĩ thuật khó của thiết kế thuật toán. Đây là kĩ thuật thường được sử dụng để giải quyết một số dạng bài toán khó nhưng chưa có thuật toán hiệu quả để giải quyết.

Tất cả nội dung các bài học đều bắt đầu bằng những ví dụ cụ thể, thông qua đó từng bước dẫn dắt các em đến với kiến thức cần nắm vững, phần cuối bài là các bài tập luyện tập và vận dụng. Kiến thức về thiết kế thuật toán và kĩ thuật lập trình là những nội dung mới nhưng rất hấp dẫn với các em say mê thích học lập trình và muốn khám phá những điều mới rất thú vị của môn học này cho riêng mình.

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

CÁC TÁC GIẢ.

Mục lục

Trang

Chuyên đề
1

THỰC HÀNH THIẾT KẾ THUẬT TOÁN THEO KĨ THUẬT ĐỆ QUY

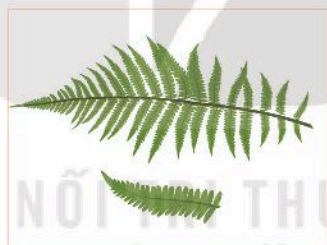
BÀI 1 ĐỆ QUY VÀ HÀM ĐỆ QUY

Sau bài này em sẽ:

- Trình bày được tính đệ quy trong một vài định nghĩa sự vật, sự việc.
- Xác định được phần cơ sở và phần đệ quy trong chương trình đệ quy.
- Nhận biết được tính ưu việt của kĩ thuật đệ quy trong định nghĩa và mô tả sự vật cũng như thiết kế chương trình.

Trong cuộc sống hằng ngày, các em thường gặp các hiện tượng sự vật, sự việc thể hiện giống hệt nhau, được lặp đi lặp lại với quy mô khác nhau. Ví dụ, búp bê Matryoshka rất nổi tiếng của Nga, khi mở búp bê mẹ ra chúng ta lại thấy búp bê con bên trong. Lá dương xỉ có mỗi nhánh lá có cấu trúc giống cấu trúc tổng thể của lá. Cây súp lơ có mỗi nhánh của cây súp lơ là hình ảnh thu nhỏ của cả cây súp lơ,... Em có thể nói gì về đặc điểm chung nhất của các búp bê Matryoshka, lá dương xỉ và cây súp lơ?

Hình 1.1. Búp bê Matryoshka



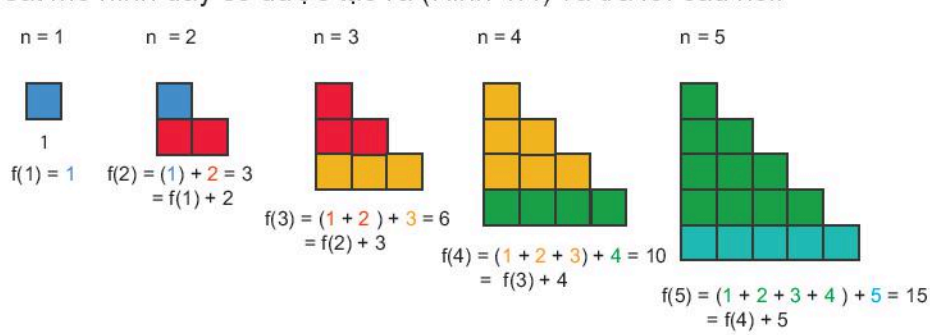
Hình 1.2. Lá dương xỉ

Hình 1.3. Cây súp lơ

1. Khái niệm đệ quy

Hoạt động 1 **Làm quen với đệ quy**

Quan sát mô hình dãy số được tạo ra (Hình 1.4) và trả lời câu hỏi:



Hình 1.4. Mô hình dãy số

1. Dãy số được tạo theo quy luật nào?
2. Em hãy xác định hình và dãy số trong trường hợp n = 6.

Hoạt động 1 là mô hình để tính tổng dãy số tự nhiên từ 1 đến k. Ta nhận thấy hình các ô vuông bước thứ k + 1 được thiết lập bằng cách bổ sung thêm phía dưới mẫu hình thứ k một hàng gồm k + 1 ô vuông (k = 1, 2,...). Như vậy, tính chất của mẫu hình là bước sau có sử dụng lại mô hình của bước trước.

Khi một sự vật, hiện tượng có tính chất lặp lại chính nó hoặc được định nghĩa theo chính sự vật, hiện tượng đó thì ta gọi là đệ quy. Trong thực tế có rất nhiều sự vật, hiện tượng được mô tả hoặc định nghĩa đệ quy. Chẳng hạn:

- Quan hệ "Họ hàng" có thể được định nghĩa như sau:

  a) Bố, mẹ, anh, chị, em và các con của tôi là họ hàng của tôi.

  b) Nếu một người là họ hàng của tôi thì bố, mẹ, anh, chị, em và các con của người đó sẽ là họ hàng của tôi.

- Dãy số tự nhiên có thể định nghĩa đơn giản như sau:

  a) Số 0 là số tự nhiên.

  b) Nếu n là số tự nhiên thì n + 1 cũng là số tự nhiên.

- Dãy số Fibonacci F(n) nổi tiếng được định nghĩa như sau:

  a) $F_0 = 0, F_1 = 1$

  b) $F_n = F_{n-1} + F_{n-2}$ với $n > 1$.

Trong các ví dụ trên, ta thấy có điểm chung để định nghĩa một đối tượng, người ta sử dụng lại chính đối tượng đó.

Có thể hiểu một cách tổng quát khái niệm đệ quy như sau: Một khái niệm A được định nghĩa theo đệ quy nếu trong định nghĩa A có sử dụng ngay chính khái niệm A.

Khái niệm đệ quy được sử dụng rất nhiều trong cuộc sống, đặc biệt trong toán học và khoa học máy tính.

Khi một sự vật, hiện tượng có tính chất lặp lại chính nó hoặc được định nghĩa theo chính sự vật, hiện tượng đó thì được gọi là đệ quy. Các định nghĩa sử dụng đệ quy trong mô tả sự vật, khái niệm thường ngắn gọn và dễ hiểu.

1. Trường hợp nào sau đây không có tính chất đệ quy?

A. Tổ ong.

B. Bắp cải.

C. Lát cắt hành.

D. Ngôi sao.

2. Phát biểu nào sau đây **sai** về đệ quy?

A. Một đối tượng được gọi là đệ quy nếu nó hoặc một phần của nó được định nghĩa thông qua khái niệm về chính nó.

B. Đối tượng đệ quy thì sự vật, hiện tượng liên quan đến đối tượng sẽ được lặp lại nhiều lần.

C. Trong đệ quy, lời giải của một bài toán phụ thuộc vào lời giải của các trường hợp nhỏ hơn của cùng một bài toán.

D. Đệ quy là cách gọi khác của lặp.

2. Công thức truy hồi

Hoạt động 2 **Tìm hiểu công thức truy hồi và các khái niệm liên quan đến đệ quy**

Đọc, quan sát các công thức sau để phát hiện các đặc điểm tương tự giữa các công thức này và khái niệm đệ quy.

a) Dãy số Fibonacci

Dãy số Fibonacci được phát hiện từ rất lâu và hiện nay đã trở nên phổ biến ở khắp các lĩnh vực khác nhau của khoa học. Dãy được định nghĩa như sau:

a) $F_0 = 0, F_1 = 1$

b) $F_n = F_{n-1} + F_{n-2}$ với $n > 1$.

Các đẳng thức a) được gọi là điều kiện ban đầu, hay cơ sở của dãy, công thức b) được gọi là công thức truy hồi (hay công thức đệ quy) của dãy. Một số phần tử ban đầu của dãy là: 0, 1, 1, 2, 3, 5, 8. Số Fibonacci có liên quan chặt chẽ đến khái niệm tỉ lệ vàng = $(1 + \sqrt{5})/2 \approx 1.618$ có nhiều ứng dụng thú vị trong toán học, nghệ thuật và trong đời sống.

b) Dãy số Lucas

Các số Lucas cũng rất nổi tiếng không chỉ vì nó rất gần với dãy số Fibonacci, mà ngay định nghĩa của dãy số này cũng giống Fibonacci. Dãy số Lucas được định nghĩa như sau:

a) $L_0 = 2, L_1 = 1$

b) $L_n = L_{n-1} + L_{n-2}$ với $n > 1$.

Như vậy, chúng ta thấy công thức truy hồi của dãy số Lucas hoàn toàn giống với công thức truy hồi của dãy số Fibonacci, chỉ khác ở phần định nghĩa cơ sở. Một số phần tử ban đầu của dãy là: 2, 1, 3, 4, 7, 11.

c) Dãy số Pell

Dãy số Pell được định nghĩa như sau:

a) $P_0 = 0$, $P_1 = 1$.

b) $P_n = 2P_{n-1} + P_{n-2}$ với $n > 1$.

Dãy số Pell gắn liền với cách tính gần đúng của $\sqrt{2}$. Dãy số này chính là dãy các mẫu số của dãy các phân số tối giản có giới hạn $\sqrt{2}$ là: 1/1, 3/2, 7/5, 17/12, 41/29.

Tất cả các công thức truy hồi đều có hai phần: phần cơ sở để xác định các giá trị ban đầu và phần truy hồi để tính các phần từ tiếp theo. Tất cả các dãy số được định nghĩa thông qua công thức truy hồi chính là được định nghĩa bằng khái niệm đệ quy.

1. Em hãy xác định phần cơ sở và phần đệ quy của n!.

$$n! = \begin{cases} 1 & \text{nếu } n = 0 \text{ hoặc } n = 1 \\ n \times (n-1)! & \text{nếu } n > 1 \end{cases}$$

2. Em hãy xác định phần cơ sở và phần đệ quy của x^n.

$$x^n = \begin{cases} 1 & \text{nếu } n = 0 \\ x \times x^{n-1} & \text{nếu } n > 0 \end{cases}$$

3. Hàm đệ quy

Hoạt động 3 Tìm hiểu và thiết lập hàm đệ quy

Bạn An được yêu cầu viết các hàm đệ quy cho các bài toán sau:

1. Viết một hàm có chức năng in ra các số đếm ngược từ n xuống 1.

2. Viết hàm tính số Fibonacci thứ n.

Bạn An đã viết các hàm giải hai bài toán trên như sau:

Chương trình 1.

```
1    def countdown(n):
2        print(n)
3        countdown(n - 1)
```

Chương trình 2.

```
1    def Fibonacci(n):
2        return Fibonacci(n - 1) + Fibonacci(n - 2)
```

Các hàm trên của bạn An có đúng không?

Nhận xét. Các hàm của bạn An viết đều có lệnh gọi đến chính mình, vậy đây là các hàm đệ quy. Tuy nhiên cả hai hàm trên đều lỗi.

– Hàm của chương trình 1 sẽ bị lặp vô hạn lần. Như vậy, muốn sửa lỗi này cần có các lệnh điều khiển làm dừng quá trình gọi đệ quy. Các lệnh này được gọi là lệnh điều khiển dừng hay **phần điều khiển dừng** của hàm. Chương trình 1 được viết lại đúng sau khi thêm phần điều khiển dừng như sau:

```
1   def countdown(n):
2       if n > 0:          <-- Lệnh điều khiển tiếp tục gọi đệ quy
3           print(n)
4           countdown(n - 1)
5       else:              <-- Lệnh điều khiển dừng đệ quy
6           return
```

Lưu ý: Các lệnh 5, 6 có thể không có mà không ảnh hưởng đến việc điều khiển dừng đệ quy. Vậy chương trình trên có thể viết lại như sau, trong đó lệnh 2 sẽ đóng vai trò phần cơ sở của đệ quy.

```
1   def countdown(n):
2       if n > 0:
3           print(n)
4           countdown(n-1)
```

– Chương trình 2 có 2 lỗi: lỗi gọi đệ quy vô hạn không dừng và lỗi không thiết lập được các giá trị ban đầu của số Fibonacci với các trường hợp n = 0 và n = 1. Như vậy, để sửa các lỗi này cần đưa vào các lệnh điều khiển dừng gọi đệ quy vô hạn và các lệnh thiết lập các giá trị ban đầu của dãy. Các lệnh thiết lập các giá trị ban đầu của hàm với tham số đầu vào nhỏ sẽ được gọi là **phần cơ sở** của hàm đệ quy.

Chương trình 2 được sửa lại sau khi bổ sung các lệnh phần cơ sở và điều khiển dừng.

```
1   def Fibonacci(n):
2       if n >= 0:         <-- Lệnh điều khiển tiếp tục gọi đệ quy
3           if n == 0:
4               return 0   <-- Các lệnh thiết lập giá trị ban đầu của dãy Fibonacci
5           elif n == 1:
6               return 1
7           else:
8               return Fibonacci(n - 1) + Fibonacci(n - 2)
```

Lưu ý: Trên thực tế khi gọi hàm đệ quy Fibonacci(n), thông số n luôn được kiểm soát bên ngoài sao cho n ≥ 0 nên lệnh tại dòng 2 có thể bỏ đi. Với tham số đầu vào

n ≥ 0 các lệnh thuộc phần cơ sở (từ dòng 3 đến dòng 6) có thêm vai trò điều khiển dừng của lời gọi đệ quy. Chương trình 2 có thể viết lại ngắn hơn như sau:

```
1   def Fibonacci(n):
2       if n == 0:
3           return 0
4       elif n == 1:
5           return 1
6       else:
7           return Fibonacci(n - 1) + Fibonacci(n - 2)
```

Lệnh điều khiển dừng và các lệnh thuộc phần cơ sở được gọi chung là **phần cơ sở** của đệ quy. Như vậy, mỗi hàm hay thủ tục đệ quy đều phải có hai phần: phần gọi đệ quy và phần cơ sở có vai trò thiết lập các giá trị ban đầu của hàm và điều khiển dừng của đệ quy.

> Hàm hay thủ tục được gọi là đệ quy nếu có lời gọi đến chính mình. Mọi hàm đệ quy phải có phần gọi đệ quy và phần cơ sở. Phần cơ sở sẽ thiết lập các giá trị ban đầu của hàm và có vai trò kiểm soát, kết thúc các lời gọi đệ quy.

1. Trong chương trình tính số Fibonacci, các lệnh nào là phần cơ sở, các lệnh nào là phần đệ quy của chương trình?
2. Một hàm đệ quy sẽ có những thành phần nào?

   A. Phần cơ sở và phần khởi tạo.

   B. Phần cơ sở và phần đệ quy.

   C. Phần đệ quy và phần khởi tạo.

LUYỆN TẬP

1. Viết chương trình in và đếm xuôi từ 1 đến 100 trên màn hình.
2. Viết chương trình tính số Lucas thứ n.

VẬN DỤNG

1. Viết chương trình nhập số n từ bàn phím và in ra n số hạng đầu tiên của dãy số Pell.
2. Viết chương trình tính số Pell thứ n.

BÀI 2 THIẾT KẾ THUẬT TOÁN ĐỆ QUY

Sau bài này em sẽ:

- Ứng dụng được kĩ thuật đệ quy trong việc thiết kế một vài thuật toán đơn giản.
- Nhận biết được tính ưu việt của kĩ thuật đệ quy trong thiết kế thuật toán.

An được giao tìm một thiết kế mới cho bài toán tính tổng S(n) = 1 + 2 + ... + n. An nhận thấy S(n) có thể được viết như sau: S(n) = 1 + 2 + ... + n = 1 + 2 + ... + n − 1 + n = S(n − 1) + n. Do đó, việc tính S(n) có thể được tính từ S(n − 1), tương tự S(n − 1) lại có thể được tính từ S(n − 2), cứ như vậy, cuối cùng sẽ dẫn đến cần tính S(0), nhưng S(0) = 0.

Em có thể giúp An hoàn thiện ý tưởng trên thành một chương trình hay không?

1. Ý tưởng thiết kế theo đệ quy

Hoạt động 1 **Tìm hiểu ý tưởng thiết kế thuật toán theo đệ quy**

Trao đổi, thảo luận và tìm hiểu ý tưởng thực hiện các tính toán sau bằng kĩ thuật đệ quy.

1. Tính tổng S(n) = 1 + 2 + ... + n.

2. Tính luỹ thừa $a^n = a \times a \times ... \times a$ (n lần).

3. Tính n giai thừa n! = 1 × 2 × ... × n.

Chúng ta sẽ xem xét lần lượt các bài toán trên.

a) Bài toán 1 trong Hoạt động 1

Phân tích

Bước 1. Bài toán yêu cầu tính tổng của n số nguyên từ 1 đến n. Cần thiết lập hàm S(n) trả về giá trị tổng cần tìm.

Bước 2. Điều kiện n ≥ 0. Với n = 0 ta có S(n) = 0. Đây là phần cơ sở cho điều kiện dừng của lời gọi đệ quy của hàm S(n).

Bước 3. Dễ thấy S(n) = n + S(n − 1) là công thức truy hồi của hàm S(n) và là cơ sở của lời gọi đệ quy của hàm.

Thuật toán 1: Hàm tính tổng.

```
1 def S(n):
2     if n == 0:
3         return 0
4     else:
5         return n + S(n - 1)
```

b) Bài toán 2 trong Hoạt động 1

Phân tích

Bước 1. Bài toán yêu cầu tính luỹ thừa a^n. Cần thiết lập hàm exp(a,n) trả về giá trị a^n.

Bước 2. Điều kiện là n ≥ 0 và theo quy ước thì exp(a,0) = 1 với mọi a. Đây chính là phần cơ sở cho điều kiện dừng của lời gọi đệ quy của hàm exp(a,n).

Bước 3. Ta có $a^n = a \times a^{n-1}$ suy ra exp(a,n) = a × exp(a,n-1), đây là công thức truy hồi tính exp(a,n). Từ đó có thể thiết lập lời gọi đệ quy của hàm này.

Thuật toán 2: Hàm tính luỹ thừa

```
1 def exp(a, n):
2     if n == 0:
3         return 1
4     else:
5         return a*exp(a, n - 1)
```

c) Bài toán 3 trong Hoạt động 1

Phân tích

Bước 1. Bài toán yêu cầu tính n giai thừa n!. Ta cần thiết lập hàm giaithua(n) trả về giá trị n!.

Bước 2. Điều kiện là n ≥ 0 và quy ước 0! = 1, tức là giaithua(0) = 1. Đây là cơ sở cho điều kiện dừng của lời gọi đệ quy của hàm giaithua(n).

Bước 3. Ta có công thức giaithua(n) = n × giaithua(n-1), đây là công thức truy hồi tính giaithua(n). Từ đó dễ dàng thiết lập lời gọi đệ quy cho hàm này.

Thuật toán 3: Hàm tính giai thừa

```
1 def giaithua(n):
2     if n == 0:
3         return 1
4     else:
5         return n*giaithua(n - 1)
```


Nhận xét:

– Cả ba bài toán trên đều có chung tính chất là từ bài toán gốc có thể đưa về trường hợp có tham số đầu vào nhỏ hơn. Điều đó sẽ dẫn đến việc có thể dùng kĩ thuật đệ quy để gọi chính hàm gốc. Nếu với các dữ liệu đầu vào nhỏ có thể giải trực tiếp (ví dụ các bài toán trên đều có lời giải cho trường hợp n = 0) thì phần cơ sở của đệ quy sẽ thực hiện lời giải trực tiếp này.

– Trong cả ba chương trình trên, các hàm được định nghĩa với tham số n mặc định phải thoả mãn n ≥ 0. Do đó, các hàm này không cần có lệnh điều khiển dừng tường minh. Nhóm các lệnh cơ sở sẽ làm luôn chức năng kiểm soát đừng của lặp đệ quy.

– Quan sát các hàm đệ quy trên chúng ta thấy rất rõ tính ưu việt của kĩ thuật lập trình đệ quy, đó là tính ngắn gọn, đơn giản và dễ hiểu của chương trình.

Ý tưởng chính của kĩ thuật đệ quy là biến đổi bài toán ban đầu về bài toán với kích thước nhỏ hơn. Nếu bài toán có kích thước nhỏ có thể giải được thì có thể thiết lập lời giải đệ quy cho bài toán này. Lời giải của các bài toán sử dụng đệ quy thường ngắn gọn và dễ hiểu.

1. Hãy chỉ ra phần cơ sở và phần đệ quy của các chương trình trên.
2. Vì sao trong ý tưởng thiết kế đệ quy trên, yêu cầu từ bài toán với kích thước lớn cần phải đưa về cùng bài toán đó với kích thước nhỏ hơn?

2. Thuật toán tìm kiếm nhị phân

Hoạt động 2 **Thiết kế thuật toán đệ quy cho bài toán tìm kiếm nhị phân**

Chúng ta đã biết thuật toán tìm kiếm nhị phân trên dãy các phần tử đã sắp xếp. Hãy tìm thiết kế mới của thuật toán này theo kĩ thuật đệ quy. Trao đổi, thảo luận và trả lời các câu hỏi sau:

1. Nêu ý tưởng chính của giải thuật tìm kiếm nhị phân sử dụng đệ quy.
2. Vị trí nào trong thuật toán có thể gợi ý cho kĩ thuật đệ quy?
3. Phần cơ sở của thiết kế đệ quy nằm ở bước nào?

Bài toán gốc của tìm kiếm nhị phân như sau:

Cho trước dãy A bao gồm n phần tử đã được sắp xếp theo thứ tự:

$A[0] \le A[1] \le \ldots \le A[n-1]$.

Cho trước giá trị cần tìm K. Cần tìm chỉ số i của dãy A sao cho A[i] = K. Nếu tìm thấy trả về i, nếu không thấy trả về −1. Chú ý: Nếu có nhiều chỉ số thoả mãn thì chỉ cần trả về một trong số chúng.

Chúng ta nhớ lại ý tưởng chính của thuật toán tìm kiếm nhị phân.

Hình 2.1. *Ý tưởng của thuật toán tìm kiếm nhị phân*

Các bước thực hiện tìm kiếm nhị phân như sau:

Bước 1: Đầu tiên thiết lập các biến left = 0, right = len(A) − 1.

Bước 2: Lặp liên tục cho đến khi left > right. Các bước lặp như sau:

Bước 2.1: Đặt mid = (left + right)//2.

Bước 2.2: Nếu A[mid] = K thì dừng chương trình, trả về giá trị mid.

Bước 2.3: Nếu A[mid] > K thì tìm tiếp trong dãy con bên trái: A(left, mid − 1).

Bước 2.4: Nếu A[mid] < K thì tìm tiếp trong dãy con bên phải: A(mid + 1, right

Bước 3: Nếu left > right thì dừng chương trình, trả về giá trị −1.

Theo mô tả ở trên, chúng ta thấy Bước 2.3 và Bước 2.4 thực hiện việc đưa bài toán tìm kiếm về chính bài toán đó với các dãy con có kích thước nhỏ hơn, cụ thể là bằng $\frac{1}{2}$ kích thước dãy ban đầu. Như vậy, lệnh gọi đệ quy sẽ thực hiện tại các bước này. Bước 3 đóng vai trò phần cơ sở của đệ quy, khi độ dài của dãy cần tìm kiếm bằng 0 thì chương trình sẽ dừng lại và thông báo không tìm thấy.

Chúng ta sẽ thiết lập hàm đệ quy cho bài toán tìm kiếm này dưới dạng:

```
binarySearch(A, left, right, K)
```

Ý nghĩa của hàm này là thực hiện tìm kiếm giá trị K trên dãy các phần tử đã sắp xếp tính từ chỉ số left đến right.

Thuật toán tìm kiếm nhị phân được mô tả bằng hàm đệ quy:

```
1  def binarySearch(A, left, right, K):
2      if left <= right:
3          mid = (left + right)//2
4          if A[mid] == K:
5              return mid
6          elif A[mid] < K:
7              return binarySearch(A, mid + 1, right, K)
8          else:
9              return binarySearch(A, left, mid - 1, K)
10     else:
11         return -1
```

Lệnh gọi hàm đệ quy cho bài toán tìm kiếm là:

```
binarySearch(A, 0, len(A) - 1, K)
```


Thuật toán tìm kiếm nhị phân có thể biểu diễn bằng kĩ thuật đệ quy trên độ dài của dãy cần tìm kiếm. Mỗi lần gọi đệ quy sẽ gọi hàm tìm kiếm trên dãy có độ dài bằng $\frac{1}{2}$ độ dài của dãy hiện tại.

1. Trong chương trình trên lệnh nào đóng vai trò là phần cơ sở của đệ quy?

2. Giả sử A = [1, 3, 7, 9] và K = 10. Nếu áp dụng chương trình trên thì cần mấy lần gọi hàm đệ quy?

LUYỆN TẬP

1. Viết chương trình theo kĩ thuật đệ quy để tính hàm SL(n) là tổng các số tự nhiên lẻ nhỏ hơn hoặc bằng n.

2. Cho trước dãy A. Viết chương trình đệ quy để in dãy A theo thứ tự ngược lại.

VẬN DỤNG

1. Viết chương trình tính tổng S = 1! + 2! + ... + n! theo hai cách:

a) Không sử dụng đệ quy.

b) Có sử dụng kĩ thuật đệ quy.

2. Chúng ta đã biết thuật toán sắp xếp chèn trên dãy A cho trước theo hàm sau:

```
def InsertionSort(A):
    n = len(A)
    for i in range(1,n):
        value = A[i]
        j = i-1
        while j >= 0 and A[j] > value:
            A[j+1] = A[j]
            j = j -1
        A[j+1] = value
```

Hãy thiết kế lại chương trình trên sử dụng kĩ thuật đệ quy.

3. Bạn An đã nghĩ ra thuật toán tìm kiếm nhị phân bằng đệ quy theo cách khác như sau:

```
def binarySearch(A, left, right, K):
    if left == right:
        if A[left] == K:
            return left
        else:
            return -1
    else:
        mid = (left + right)//2
        if A[mid] == K:
            return mid
        elif A[mid] < K:
            return binarySearch(A, mid + 1, right, K)
        else:
            return binarySearch(A, left, mid, K)
```

a) Chương trình của bạn An có đúng không?

b) Trong chương trình trên, phần cơ sở là những lệnh nào?

BÀI 3 THỰC HÀNH GIẢI TOÁN THEO KĨ THUẬT ĐỆ QUY

Sau bài này em sẽ:

- Viết được một vài chương trình có sử dụng kĩ thuật đệ quy cho một số bài toán đơn giản.

Khi áp dụng kĩ thuật đệ quy để giải các bài toán, theo em cần phải đặc biệt lưu ý đến điều gì?

NHIỆM VỤ 1 Bài toán biến đổi số nhị phân sang số thập phân.

Đầu vào của bài toán là một xâu nhị phân bất kì, ở đây xâu nhị phân được hiểu là xâu chỉ bao gồm các chữ số 0 và 1. Xâu này là biểu diễn của một số tự nhiên n trong hệ đếm nhị phân. Yêu cầu của bài toán là tìm số thập phân tương ứng với biểu diễn nhị phân của xâu đầu vào. Bài toán cần giải bằng kĩ thuật đệ quy.

Hướng dẫn:

Phân tích: Gọi s là xâu kí tự nhị phân đầu vào có độ dài n:

$$s = s_0\, s_1 \ldots s_{n-1}.$$

Vì các kí tự của xâu có thể biểu diễn thông qua chỉ số nên xâu s có thể được coi là một dãy các chữ số 0, 1 như sau:

s[0], s[1], ..., s[n − 1]. (1)

Chúng ta sẽ thiết kế hàm dec(s, n) để tính giá trị số thập phân có biểu diễn nhị phân theo dãy (1).

Với n = 1 thì s chỉ là một kí tự, do đó hàm dec(s, 1) sẽ chính là int(s[0]).

Với n > 1 là trường hợp tổng quát của bài toán. Chúng ta đã biết nếu số M được biểu diễn bằng số nhị phân dạng $a_0a_1 \ldots a_{n-1}$ thì ta phải có công thức sau:

$$M = a_0 2^{n-1} + a_1 2^{n-2} + \ldots + a_{n-2} 2^1 + a_{n-1}.$$

Biến đổi công thức trên, chúng ta có:

$$M = 2(a_0 2^{n-2} + a_1 2^{n-3} + \ldots + a_{n-3} 2^1 + a_{n-2}) + a_{n-1}. \quad (2)$$

Rõ ràng biểu thức trong ngoặc trên chính là biểu diễn thập phân của số nhị phân $a_0, a_1, \ldots, a_{n-2}$.

Vậy nếu gọi dec(s, n) là số thập phân cần tìm của dãy nhị phân (1) thì công thức (2) sẽ cho chúng ta công thức truy hồi sau:

dec(s, n) = 2dec(s, n − 1) + int(s[n − 1]). (3)

Áp dụng công thức (3) vào bài toán của chúng ta, chú ý rằng đầu vào của s là xâu nhị phân nên các phần tử s[k] đều là chữ số. Hàm đệ quy dec(s,n) được thiết kế như sau:

```
1 def dec(s, n):
2       if n == 1:
3           return int(s[0])
4       else:
5           return 2*dec(s, n - 1) + int(s[n - 1])
```

Lưu ý: Công thức tại dòng lệnh 5 ở trên chính là công thức (3).

Lệnh gọi hàm đệ quy như sau:

```
dec(s, len(s))
```

NHIỆM VỤ 2 Tính ƯCLN của hai số nguyên không âm.

Cho trước hai số nguyên a, b không âm, không đồng thời bằng 0. Ước chung lớn nhất của hai số này sẽ kí hiệu là ƯCLN (a, b) hay gcd(a, b). Bài toán yêu cầu tính gcd(a, b) với a, b cho trước.

Hướng dẫn:

1. Cách tính tự nhiên

Có rất nhiều cách tìm ƯCLN của hai số a, b cho trước. Cách tìm đơn giản như sau: Nếu a = b thì ƯCLN sẽ chính là các số này, ngược lại nếu hai số này không bằng nhau, ví dụ a > b thì chúng ta biến đổi số lớn hơn bằng cách trừ đi cho số kia, ví dụ đặt a = a – b. Quá trình như vậy sẽ tiếp tục và kết thúc khi a = b, chúng ta tìm được ƯCLN. Ví dụ cần tính gcd(12, 9), chúng ta thực hiện theo các bước sau:

1. gcd(12, 9) = gcd(3, 9).
2. gcd(3, 9) = gcd(3, 6).
3. gcd(3, 6) = gcd(3, 3).
4. gcd(3, 3) = 3.

Việc thiết lập chương trình tính gcd(a, b) theo cách trên (theo cả hai phương án đệ quy và không đệ quy) sẽ được cho trong phần vận dụng.

2. Cách tính nhanh theo thuật toán Euclid

Chúng ta sẽ tìm hiểu một cách tính khác cho bài toán trên. Cách tính này do nhà toán học Euclid tìm ra vào năm 300 trước công nguyên và là một trong những thuật toán nổi tiếng nhất. Để tìm hiểu thuật toán Euclid tính ƯCLN, chúng ta quan sát quy trình tính toán hàm gcd() theo Bảng 3.1, dựa trên ví dụ tính gcd(12, 9).

Bảng 3.1. *Quy trình tính toán hàm gcd()*

STT	a	b	a % b	Ghi chú
1	12	9	3	
2	9	3	0	
3	3	0		Nếu b = 0 thì dừng lại, thông báo ƯCLN = a

Kết quả thu được gcd(12, 9) = 3.

Dễ thấy thuật toán Euclid sẽ nhanh hơn thuật toán tự nhiên đã mô tả ở trên.

Thuật toán tìm ƯCLN có thể mô tả theo hai quy luật sau:

1. Nếu b = 0 thì gcd(a, 0) = a.

2. Nếu b > 0 thì gcd(a, b) = gcd(b, a mod b).

Từ các phân tích trên chúng ta thiết lập chương trình đệ quy tính gcd(a, b) theo thuật toán Euclid như sau:

```
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a%b)
```

LUYỆN TẬP

1. Mô tả các bước tính gcd(93, 60).

2. Viết chương trình chuyển đổi số nhị phân sang hệ thập phân (tương tự nhiệm vụ 1) nhưng dãy nhị phân đầu vào được cho dưới dạng một dãy (list) các số 0 và 1. Ví dụ nếu dãy đầu vào là A = [1, 1, 1, 1, 1, 1, 1] thì kết quả đầu ra là 127.

VẬN DỤNG

1. Bài toán tính ƯCLN của hai số nguyên dương a, b có một cách tính khác như sau:

```
def gcd(a, b):
    while a !=  b:
        if a < b:
            b = b - a
        else:
            a = a - b
    return a
```

Hãy viết lại chương trình trên theo kĩ thuật đệ quy.

2. Thiết lập chương trình tính hàm gcd(a, b) – ƯCLN của các số nguyên không âm a, b theo thuật toán Euclid nhưng không đệ quy.

3. Lớp An tiến hành đo chiều cao của cả lớp, kết quả lưu vào một tệp có tên chieucao.inp, trong tệp ghi lần lượt họ tên các bạn trong lớp và chiều cao tương ứng. Thầy hiệu trưởng yêu cầu tổng kết và gửi cho Ban Giám hiệu tên và chiều cao của bạn thấp nhất và cao nhất lớp, kết quả phải lưu ra tệp ketqua.out. Em hãy giúp bạn An viết chương trình giải quyết yêu cầu này theo kĩ thuật đệ quy.

Ví dụ thông tin đầu vào và ra của bài toán sẽ như sau:

chieucao.inp
Nguyễn Việt Hà 1.75 Bùi Quang Mơ 1.66 Trương Thị Lộc 1.50 Trần Văn Hoá 1.78

ketqua.out
Trương Thị Lộc 1.50 Trần Văn Hoá 1.78

BÀI 4 BÀI TOÁN THÁP HÀ NỘI

Sau bài này em sẽ:

- Biết và trình bày được lời giải bài toán Tháp Hà Nội sử dụng kĩ thuật đệ quy.

Năm 1883, tại một số tỉnh thành của Việt Nam và tại Pháp xuất hiện một trò chơi được quảng cáo với tên "Tháp Hà Nội" (La tour d'Hanoi). Trò chơi này được bán rộng rãi và theo một tờ quảng cáo vào thời gian đó là sẽ trao giải hàng triệu francs cho ai có thể giải được tất cả các mức từ thấp đến cao nhất là 64 đĩa. Trong tờ rơi đó cũng đưa ra con số 18446744073709551615 bước chuyển cho trường hợp 64 đĩa và khuyến cáo rằng sẽ cần hàng tỉ năm để giải được trò chơi này.

Hình 4.1. Tờ quảng cáo trò chơi "Tháp Hà Nội", 1883.

Trò chơi như sau: có ba cái cọc (ví dụ cọc 1, 2, 3) và n cái đĩa được xếp tại cọc 1 theo thứ tự to dần từ trên xuống. Yêu cầu chuyển n đĩa này sang cọc 3 với điều kiện là được dùng cọc 2 làm trung gian, mỗi lần chỉ được phép chuyển 1 đĩa và không cho phép đặt đĩa to chồng lên đĩa nhỏ.

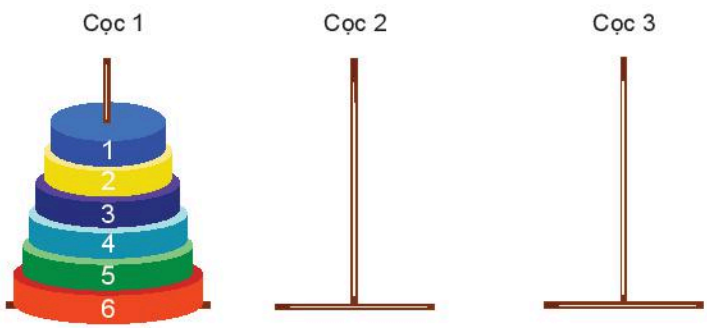
Em hãy suy nghĩ và thử giải trò chơi trên với n = 1, 2.

1. Mô tả bài toán Tháp Hà Nội

Hoạt động 1 **Tìm hiểu bài toán Tháp Hà Nội**

Đọc, tìm hiểu bài toán Tháp Hà Nội và thực hiện giải trò chơi này với số lượng đĩa nhỏ (1, 2, 3). Em có nhận xét gì về lời giải bài toán với n = 1, 2, 3?

Bài toán Tháp Hà Nội do nhà bác học Pháp tên Édouard Lucas đưa ra lần đầu tiên vào năm 1883. Bài toán như sau: Cho trước ba cọc 1, 2, 3 và n cái đĩa, kí hiệu từ 1 đến n, được xếp tại cọc 1 theo thứ tự như hình dưới đây:



***Hình 4.2.** Bài toán Tháp Hà Nội*

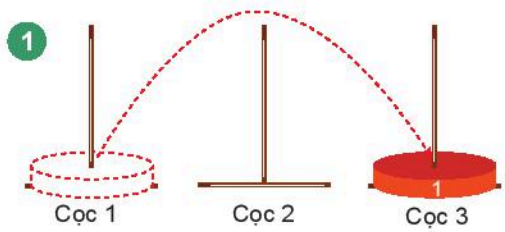
Yêu cầu: Cần chuyển n đĩa này từ cọc 1 sang cọc 3 với điều kiện sau:

- Mỗi lần chỉ được di chuyển một đĩa.
- Không được xếp đĩa to lên đĩa nhỏ hơn.
- Được phép sử dụng thêm cọc 2 làm cọc trung gian.

Với bài toán tổng quát gọi H(n) là số lần chuyển đĩa tối thiểu để giải quyết được bài toán. Nhiệm vụ của chúng ta là tìm được thuật giải đồng thời tính được giá trị H(n).

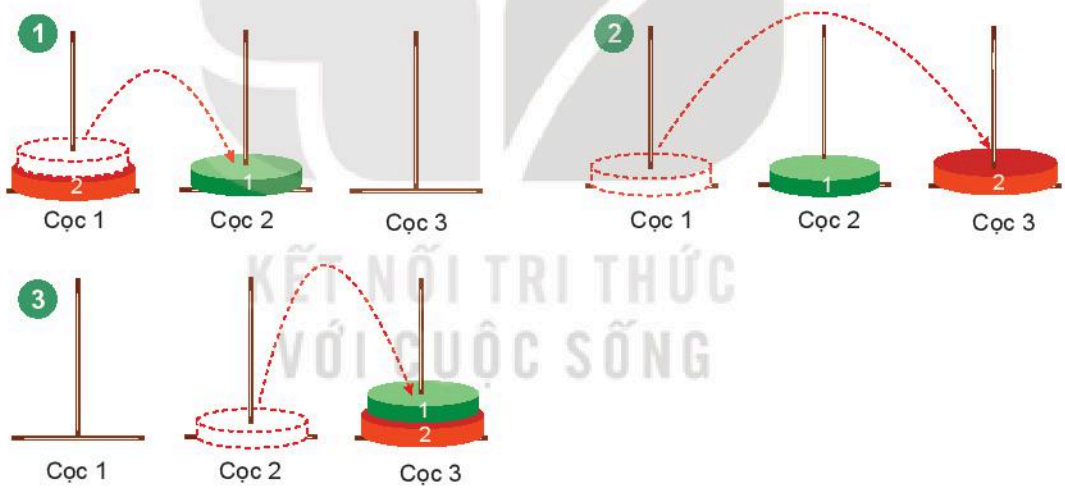
Quan sát các hình sau và mô tả các bước của bài toán với trường hợp n = 1, 2, 3.

Với n = 1, ta có H(1) = 1.



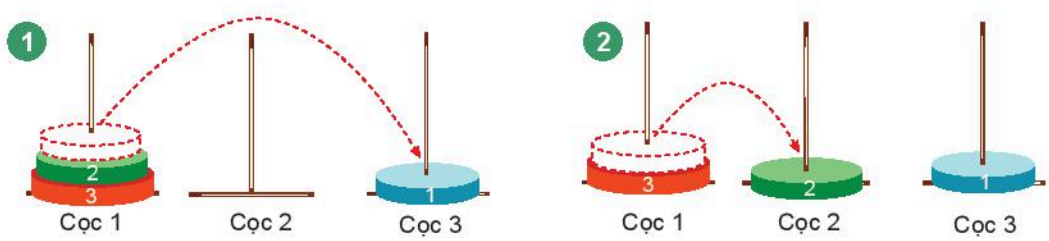
Hình 4.3. *Trường hợp n = 1*

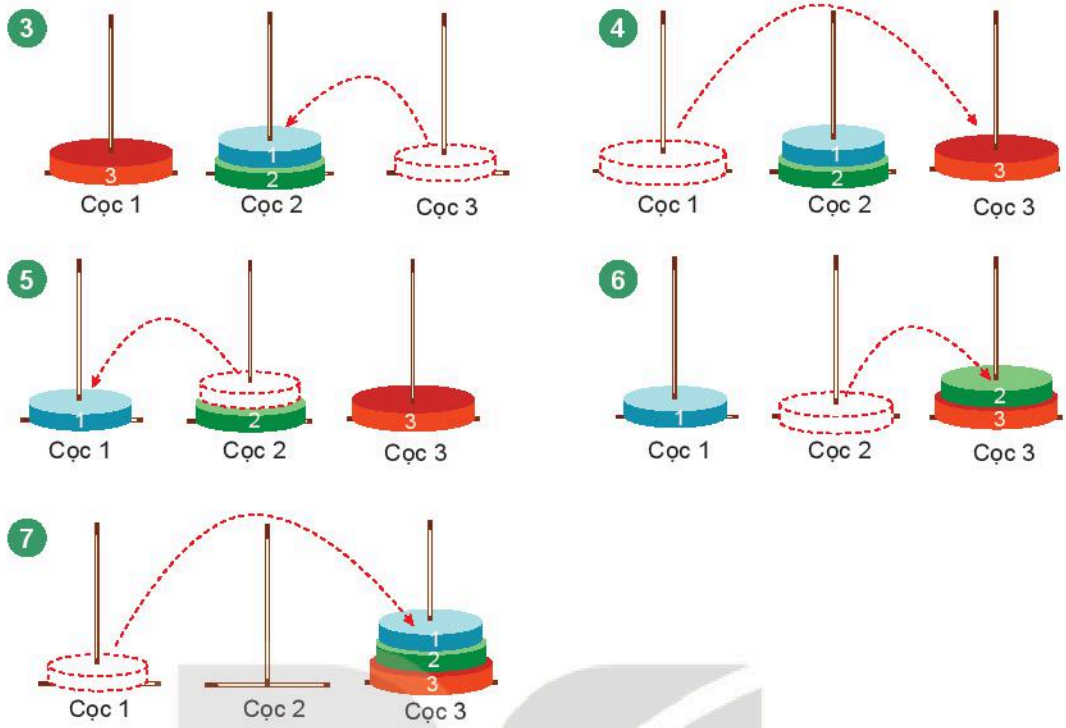
Với n = 2, ta có H(2) = 3.



Hình 4.4. *Trường hợp n = 2*

Với n = 3, ta có H(3) = 7. Quan sát lời giải sau, em có nhìn thấy quy luật gì không?





Hình 4.5. *Trường hợp n = 3*

Bài toán Tháp Hà Nội có lời giải đơn giản trong các trường hợp n = 1, 2, 3 với số lần chuyển đĩa lần lượt là 1, 3, 7.

1. Mô tả lời giải bài toán với trường hợp n = 1, 2, 3 ở trên (không dùng hình vẽ mô tả).

2. Mô tả lời giải bài toán với n = 1, 2, 3 nếu yêu cầu là di chuyển các đĩa từ cọc 1 sang cọc 2 (cọc 3 là cọc trung gian).

2. Ý tưởng lời giải bài toán Tháp Hà Nội

Hoạt động 2 **Tìm hiểu ý tưởng thiết kế đệ quy cho lời giải bài toán Tháp Hà Nội**

Đọc, trao đổi để hiểu được ý tưởng thiết kế đệ quy cho lời giải bài toán Tháp Hà Nội.

Để mô tả phép chuyển đĩa k từ cọc i đến cọc j chúng ta dùng kí hiệu sau:

k: i → j //chuyển đĩa k từ cọc i sang cọc j.

Quan sát lại các bước giải bài toán trong Hoạt động 1 và mô tả lại theo cách kí hiệu trên.

a) Trường hợp n = 1

Với n = 1, tại cọc 1 có một đĩa duy nhất, chúng ta sẽ chuyển đĩa này sang cọc 3.

1: 1 → 3 // Chuyển đĩa 1 từ cọc 1 sang cọc 3.

Như vậy, H(1) = 1.

b) Trường hợp n = 2

Với n = 2, chúng ta có hai đĩa đánh số 1, 2 tại cọc 1. Theo sơ đồ đã mô tả trong Hoạt động 1, lời giải bài toán với n = 2 sẽ như sau:

1: 1 → 2 // Chuyển đĩa 1 từ cọc 1 sang cọc 2.

2: 1 → 3 // Chuyển đĩa 2 từ cọc 1 sang cọc 3.

1: 2 → 3 // Chuyển đĩa 1 từ cọc 2 sang cọc 3.

Tổng số lần chuyển đĩa là 3, vậy H(3) = 3.

c) Trường hợp n = 3

Chúng ta có ba đĩa đánh số 1, 2, 3. Lời giải của trường hợp này sẽ sử dụng kết quả của trường hợp n = 2.

Bước 1: Chuyển 2 đĩa (đĩa 1 và 2) từ cọc 1 sang cọc 2 lấy cọc 3 làm trung gian. Việc chuyển này dựa vào trường hợp n = 2. Cụ thể sẽ cần ba thao tác như sau:

1: 1 → 3 // Chuyển đĩa 1 từ cọc 1 sang cọc 3.

2: 1 → 2 // Chuyển đĩa 2 từ cọc 1 sang cọc 2.

1: 3 → 2 // Chuyển đĩa 1 từ cọc 3 sang cọc 2.

Bước 2: Chuyển đĩa 3 từ cọc 1 sang cọc 3.

3: 1 → 3 // Chuyển đĩa 3 từ cọc 1 sang cọc 3.

Bước 3: Chuyển 2 đĩa (đĩa 1 và 2) từ cọc 2 sang cọc 3 lấy cọc 1 làm trung gian. Việc chuyển này dựa vào trường hợp n = 2. Chú ý lúc này tại cọc 3 đã có đĩa 3, nhưng đĩa này lớn nhất nên không ảnh hưởng đến việc chuyển của hai đĩa 1 và 2.

Cụ thể sẽ có ba thao tác như sau:

1: 2 → 1 // Chuyển đĩa 1 từ cọc 2 sang cọc 1.

2: 2 → 3 // Chuyển đĩa 2 từ cọc 2 sang cọc 3.

1: 1 → 3 // Chuyển đĩa 1 từ cọc 1 sang cọc 3.

Sau các bước trên việc chuyển đĩa đã hoàn thành H(3) = 7.

Nhận xét: Trong 3 bước trên, các bước 1 và 3 đều phải sử dụng lại kết quả của trường hợp n = 2, đây chính là bước áp dụng đệ quy của lời giải. Quan sát kĩ hơn lời giải của bài toán trong trường hợp n = 3, chúng ta dễ dàng tổng quát với trường hợp n bất kì.

Ý tưởng giải bài toán Tháp Hà Nội có n đĩa từ cọc 1 sang cọc 3 như sau:
Bước 1. Chuyển n – 1 đĩa từ cọc 1 sang cọc 2 lấy cọc 3 làm trung gian.
Bước 2. Chuyển đĩa n từ cọc 1 sang cọc 3.
Bước 3. Chuyển n – 1 đĩa từ cọc 2 sang cọc 3 lấy cọc 1 làm trung gian.

Viết sơ đồ chi tiết giải bài toán Tháp Hà Nội cho trường hợp n = 4. Tính H(4).

3. Thiết lập chương trình giải bài toán Tháp Hà Nội

Hoạt động 3 Thiết lập thuật toán và cài đặt chương trình cho bài toán Tháp Hà Nội

Gọi Hanoi(n, i, j, k) là bài toán yêu cầu chuyển n đĩa đang xếp ở cọc i sang cọc j lấy cọc k làm trung gian. Các đĩa được đánh số từ 1 đến n và xếp theo thứ tự từ trên xuống. Các điều kiện của việc chuyển như sau:

1. Các đĩa đánh số từ 1 đến n và có kích thước tăng dần.

2. Mỗi lần chỉ được phép chuyển một đĩa.

3. Không được phép xếp đĩa to lên trên đĩa nhỏ.

Em hãy thiết kế thuật toán đệ quy tổng quát cho bài toán trên. Yêu cầu phải mô tả chi tiết từng bước chuyển.

Lời giải đệ quy của bài toán Tháp Hà Nội như sau:

– Nếu n = 1 thì thực hiện chuyển đĩa 1 từ cọc i sang cọc j.

– Nếu n > 1 thì thực hiện theo ba bước như sau:

Bước 1: Chuyển n – 1 đĩa phía trên (từ 1 đến n – 1) từ cọc i sang cọc k lấy cọc j làm trung gian.

Bước 2: Chuyển đĩa n từ cọc i sang cọc j.

Bước 3: Chuyển n – 1 đĩa từ cọc k sang cọc j lấy cọc i làm trung gian.

Các bước 1 và 3 ở trên chính là lệnh gọi đệ quy đến bài toán gốc với giá trị n nhỏ hơn bài toán gốc. Có thể mô tả lời giải bài toán dưới dạng mã giả (pseudocode) như sau:

```
1   def Hanoi(n, i, j, k):
2    if n == 1:
3       Chuyển đĩa n từ cọc i sang cọc j
4    else:
5       Hanoi(n - 1, i, k, j)
6       Chuyển đĩa n từ cọc i sang cọc j
7       Hanoi(n - 1, k, j, i)
```

Từ chương trình, dễ dàng tính được công thức truy hồi tính số các phép chuyển đĩa của bài toán.

Với n = 1 thì H(1) = 1.

Với n > 1 bất kì, các dòng lệnh 5, 6, 7 dẫn đến công thức H(n) = 2H(n − 1) + 1.

Từ các công thức trên, có thể chứng minh và tính được giá trị của $H(n) = 2^n - 1$.

Chúng ta sẽ chuyển đổi chương trình mã giả trên thành chương trình cụ thể viết trên Python như sau:

```
def Hanoi(n, i, j, k):
    if n == 1:
        print("Chuyển đĩa", n, "từ cọc", i, "sang cọc", j)
    else:
        Hanoi(n - 1, i, k, j)
        print("Chuyển đĩa", n, "từ cọc", i, "sang cọc", j)
        Hanoi(n - 1, k, j, i)
```

Ví dụ để giải bài toán chuyển n đĩa từ cọc 1 sang cọc 3, lấy cọc 2 làm trung gian thì lệnh gọi hàm như sau:

```
Hanoi(n, 1, 3, 2)
```

Bài toán Tháp Hà Nội tổng quát có thể giải thông qua hàm Hanoi(n, i, j, k) bằng thiết kế đệ quy. Số các phép chuyển đĩa của bài toán được tính theo công thức: H(1) = 1, H(n) = 2H(n – 1) + 1. Từ đó suy ra $H(n) = 2^n - 1$.

1. Tính các giá trị H(2), H(3), H(4), H(5) của bài toán Tháp Hà Nội.

2. Viết chương trình đệ quy để tính giá trị H(n) của bài toán Tháp Hà Nội.

LUYỆN TẬP

1. Viết chương trình giải bài toán Tháp Hà Nội nhưng với tên các cọc là A, B, C.

2. Viết chương trình rút gọn của hàm Hanoi(n, i, j, k) như sau và kiểm tra kết quả.

```
def Hanoi(n, i, j, k):
    if n > 0:
        Hanoi(n - 1, i, k, j)
        print("Chuyển đĩa", n, "từ cọc", i, "sang cọc", j)
        Hanoi(n - 1, k, j, i)
```


VẬN DỤNG

1. Hãy chứng minh công thức $H(n) = 2^n - 1$ bằng quy nạp toán học. Hãy tính H(64) và so sánh với con số các bước đã được đưa ra trong tờ quảng cáo của trò chơi vào năm 1883.

2. Giả sử cần lưu dãy các bước chuyển của bài toán Tháp Hà Nội vào một danh sách để có thể sử dụng lại về sau. Mỗi bước chuyển dạng k: i → j sẽ được lưu trong một bộ ba số (k, i, j). Viết chương trình giải bài toán Tháp Hà Nội tổng quát Hanoi(n, i, j, k) chuyển n đĩa từ cọc i sang cọc j lấy cọc k làm trung gian với yêu cầu lưu tất cả các bước chuyển vào một danh sách (list). Như vậy, hàm Hanoi(n, i, j, k) sẽ trả về một danh sách bao gồm các bộ ba số dạng như đã mô tả ở trên.

BÀI 5 THỰC HÀNH THIẾT KẾ THUẬT TOÁN THEO KĨ THUẬT ĐỆ QUY

Sau bài này em sẽ:

- Thực hành thiết kế một số bài toán sử dụng kĩ thuật đệ quy.

Hãy phân tích một số ưu nhược điểm của việc áp dụng kĩ thuật đệ quy trong lập trình.

NHIỆM VỤ 1 Thuật toán sắp xếp chèn.

Viết chương trình mô tả thuật toán sắp xếp chèn theo kĩ thuật đệ quy.

Hướng dẫn:

Chúng ta sẽ phân tích lại một lần nữa thuật toán sắp xếp chèn để tìm ra quy luật cho việc thiết lập đệ quy. Thuật toán sắp xếp chèn được mô tả ngắn gọn như sau:

```
1    for i in range(1,n):
2        chèn phần tử A[i] vào vị trí đúng của các phần tử
A[0],A[1],...,A[i-1]
```

Chú ý rằng thao tác tại dòng 2 của mô tả trên chỉ phụ thuộc vào các chỉ số từ 0 đến i và không phụ thuộc vào phần còn lại của dãy. Từ nhận xét này sẽ dẫn đến ý tưởng của việc thiết kế đệ quy như sau.

Giả sử gọi insertionSortRec(A,i) là thao tác tương ứng với dòng 2 của mô tả trên, khi đó chương trình trên có thể viết lại dưới dạng sau:

```
1    def insertionSortRec(A,i):
2        if i > 0:
3            insertionSortRec(A,i-1)
4            chèn phần tử A[i] vào vị trí đúng của các phần tử
A[0],A[1],...,A[i-1]
```

Lệnh gọi đệ quy tại dòng 3 với điều kiện i > 0 như trên sẽ đảm bảo các phần tử A[0], A[1], ..., A[i-1] đã được sắp xếp đúng. Do đó việc chèn tại dòng 4 sẽ thực hiện chính xác thao tác chèn của thuật toán.

Như vậy thuật toán sắp xếp chèn có thể viết lại bằng kĩ thuật đệ quy theo hàm insertionSortRec(A,i) như sau. Chú ý các lệnh tại dòng từ 4 đến 9 chính là thao tác lõi "chèn" của thuật toán này.

```
def insertionSortRec(A,i):
    if i > 0:
        insertionSortRec(A,i-1)
        value = A[i]
        j = i-1
        while j >= 0 and A[j] > value:
            A[j+1] = A[j]
            j = j - 1
        A[j+1] = value
```

Lời gọi đệ quy của hàm là:

```
insertionSortRec(A,len(A)-1)
```

NHIỆM VỤ 2 Chuyển số thập phân sang hệ đếm nhị phân.

Cho trước số tự nhiên n ($n \geq 0$), cần đưa ra được biểu diễn nhị phân của số n. Ví dụ nếu n = 11 thì kết quả phải là chuỗi nhị phân "1011".

Hướng dẫn:

Để thiết kế được chương trình này, chúng ta quan sát Bảng 5.1 mô tả các bước tính được khai triển nhị phân của một số tự nhiên n = 11.

Bảng 5.1. Các bước triển khai nhị phân n = 11

STT	n	n//2	n%2
1	11	5	1
2	5	2	1
3	2	1	0
4	1	0	1
5	0		Số nhị phân cần tìm là 1011, dãy các số dư theo thứ tự ngược lại, từ dưới lên trên.

Quá trình thực hiện như sau: Lần lượt biến đổi n = n//2, mỗi lần như vậy cần ghi nhớ lại số dư n%2. Quá trình kết thúc khi n = 0, kết quả là dãy các số dư được ghép lại theo thứ tự ngược từ dưới lên.

Theo bảng trên, biểu diễn số 11 trong hệ nhị phân là "1011", trong đó kí tự cuối cùng là "1" chính là số dư 11%2, phần đầu của xâu là "101" chính là biểu diễn nhị phân của 5 = 11//2. Gọi binary(n) là xâu biểu diễn khai triển nhị phân của số tự nhiên n. Khi đó theo cách phân tích trên chúng ta có công thức truy hồi sau:

```
binary(n) = binary(n//2) + str(n%2)
```

Lưu ý: Theo Bảng 5.1, quá trình tính số dư sẽ dừng lại khi n = 0 và có thể đặt trường hợp cơ sở là: nếu n = 0 thì hàm trả về xâu rỗng. Nhưng nếu giá trị ban đầu n = 0 thì kết quả phải là "0" nên để hoàn thiện chương trình đệ quy, chúng ta thiết lập hai trường hợp cơ sở với n = 0 và n = 1 như sau: nếu n = 1, hàm trả lại "1"; nếu n = 0, hàm trả lại "0". Chương trình cuối cùng có thể viết như sau:

```
1   def binary(n):
2       if n == 0:
3           return "0"
4       elif n == 1:
5           return "1"
6       else:
7           return binary(n//2) + str(n%2)
```

LUYỆN TẬP

1. Viết chương trình đệ quy giải quyết nhiệm vụ 2 nhưng với yêu cầu đầu ra của hàm là một dãy (list) các số 0 và 1.

2. Viết hàm **decimal(s)** chuyển đổi xâu nhị phân s sang số thập phân tương ứng. Ví dụ nếu đầu vào là "10" thì kết quả 2, nếu đầu vào "1011" thì kết quả là 11. Yêu cầu viết theo kĩ thuật đệ quy.

VẬN DỤNG

1. Cho trước dãy số A = A[0], A[1],..., A[n − 1]. Cặp phần tử (A[i], A[j]) được gọi là nghịch đảo nếu i < j nhưng A[i] > A[j]. Viết chương trình đếm số các cặp phần tử nghịch đảo của dãy A.

a) Viết chương trình không đệ quy.

b) Viết chương trình theo kĩ thuật đệ quy.

2. Thiết kế thuật toán cho bài toán tính giá trị của đa thức dạng:

$F(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$ (1)

$$= \sum_{i=0}^{n} a_i x^i$$

Ở đây, đầu vào là các giá trị x, a_0, a_1,..., a_n.

Gọi A = [a_0, a_1,..., a_n] là dãy các hệ số của đa thức (1).

Công thức (1) có thể viết lại với định nghĩa hàm F(A, x, n) như sau:

$F(A, x, n) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$ (2)

Chuyên đề
2

THỰC HÀNH THIẾT KẾ THUẬT TOÁN THEO KĨ THUẬT CHIA ĐỂ TRỊ

BÀI 6 Ý TƯỞNG VÀ KĨ THUẬT CHIA ĐỂ TRỊ

Sau bài này em sẽ:

- Biết và giải thích được ý tưởng của kĩ thuật chia để trị thông qua một số bài toán đơn giản.
- Nêu được mối quan hệ giữa kĩ thuật chia để trị và đệ quy.

Trò chơi tìm bi giả

Có 5 viên bi giống hệt nhau, biết rằng trong các viên bi này có một viên bi giả và viên bi giả này nặng hơn các viên bi còn lại.

Chỉ với một cái cân thăng bằng, em hãy tìm ra viên bi giả đó. Cần ít nhất bao nhiêu lần cân để tìm ra viên bi giả?

Hình 6.1. Cân thăng bằng

1. Ý tưởng chia để trị

Hoạt động 1 **Tìm hiểu ý tưởng của kĩ thuật chia để trị để giải một bài toán**

1. Hãy trình bày cách giải bài toán tìm bi giả với 5 viên bi.
2. Trường hợp tổng quát có n viên bi cách làm như thế nào?
3. Ý tưởng chia để trị để giải bài toán tìm bi giả được thể hiện như thế nào?

• Với 5 viên bi, lần cân đầu tiên chúng ta lấy 4 viên bi, đặt 2 viên bi ở hai bên cân.

Trường hợp 1. Nếu cân thăng bằng thì xác định được viên bi còn lại chưa cân là bi giả. Như vậy, sau lần cân thứ nhất ta tìm được bi giả.

Trường hợp 2. Nếu cân bị lệch, ta sẽ xác định bên nặng hơn chứa bi giả. Lấy hai viên bi ở bên nặng hơn và cân mỗi bên một viên, ta sẽ xác định được viên bi giả. Như vậy, sau lần cân thứ hai ta tìm được bi giả.

• Với n viên bi.

– Nếu n lẻ, n = 2k + 1, sau lần cân thứ nhất với mỗi bên k viên, hoặc tìm thấy ngay viên bi giả, hoặc biết bên nào có chứa bi giả và tiếp tục cân với k viên bi đó.

– Nếu n chẵn, n = 2k, sau lần cân thứ nhất, ta tìm được k viên bi chứa viên bi giả.

Tổng quát, sau một lần cân, từ bài toán với n viên bi sẽ dẫn đến bài toán với [n/2] viên bi ([x] là phần nguyên của x). Khi bài toán dẫn đến còn hai hoặc ba viên bi thì chỉ cần một lần cân nữa sẽ tìm được viên bi giả.

Ý tưởng và cách giải bài toán tìm bi giả ở trên được gọi là *chia để trị*. Từ bài toán gốc luôn *chia* thành các bài toán có kích thước nhỏ hơn, ở đây là [n/2]. Khi số bi còn lại là 2 thì bài toán rất đơn giản có thể giải quyết ngay, đó là trị. Sau khi trị xong, kết hợp lại cả quá trình để tổng hợp kết quả chung sẽ giải quyết được bài toán gốc.

Chia để trị (Divide and Conquer) là một kĩ thuật rất quan trọng trong thiết kế thuật toán và chương trình, gồm ba bước như sau:

Bước 1: Chia (Divide) Chia bài toán ban đầu (phức tạp) thành hai hoặc nhiều bài toán con (đơn giản hơn). Các bài toán con này có cùng dạng với bài toán ban đầu hoặc có liên quan với nhau. Mỗi bài toán con có thể được giải quyết một cách độc lập. Áp dụng tiếp tục kĩ thuật chia để trị để chia một bài toán con thành các bài toán con đơn giản hơn nữa (một cách đệ quy) và cứ như thế cho đến khi tất cả các bài toán con đủ đơn giản, có thể được giải quyết một cách dễ dàng.

Bước 2: Trị (Conquer) Giải quyết các bài toán con, kết quả là các lời giải của các bài toán con.

Bước 3: Kết hợp (Combine) Kết hợp các lời giải của các bài toán con để có được lời giải của bài toán ban đầu.

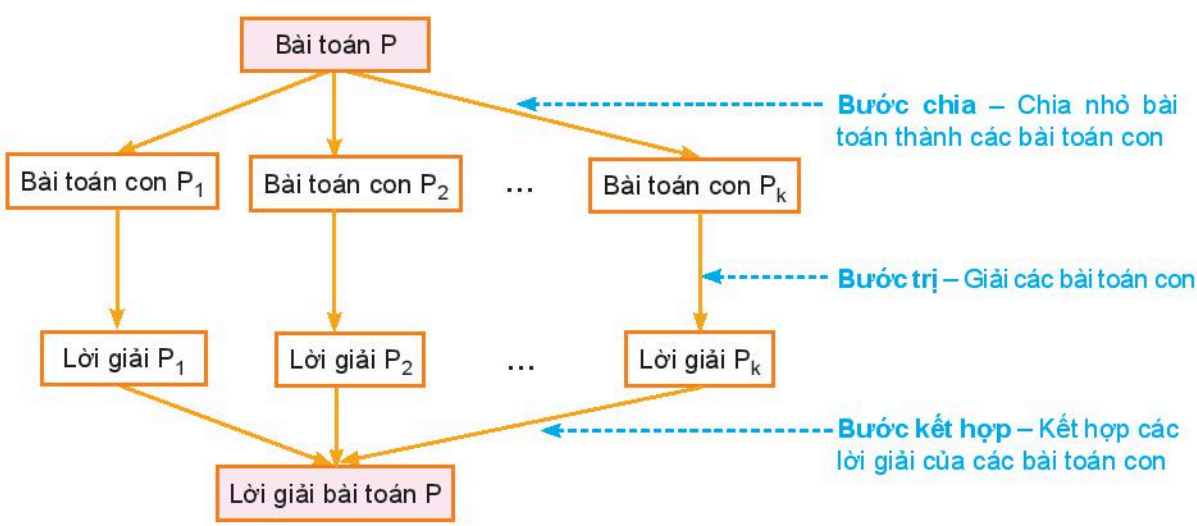
Nếu bài toán ban đầu được chia thành các bài toán con có cùng dạng với nó thì quá trình phân chia và giải quyết các bài toán này là quá trình đệ quy.

Thuật toán chia để trị: Gọi P là bài toán cần giải quyết. Hàm `ChiaDeTri(P)` giải quyết bài toán P dùng kĩ thuật chia để trị.

```
def ChiaDeTri(P):
    if P là bài toán đủ đơn giản:
        loi_giai = <Giải quyết bài toán P>
    else:
        <Chia bài toán P thành các bài toán con>                    # Chia
        <Gọi tapBTcon là tập các bài toán con>
        <Gọi tapLGcon là tập các lời giải của các bài toán con>
        for btcon in tapBTcon:
            lgBTcon = ChiaDeTri(btcon)                              # Trị
            <Ghi nhận lgBTcon vào tập tapLGcon>
        loi_giai = <Kết hợp các lời giải của tập tapLGcon>          # Kết hợp
    return loi_giai
```

Gọi hàm `ChiaDeTri(P)` với `P` là bài toán ban đầu cần giải quyết.

Sơ đồ ý tưởng chia để trị như sau:



Hình 6.2. *Sơ đồ ý tưởng chia để trị*

Kĩ thuật chia để trị là phương pháp thiết kế thuật toán theo 3 giai đoạn: chia (divide), trị (conquer) và kết hợp (combine). Bài toán gốc sẽ được chia thành các bài toán với kích thước nhỏ hơn, sau đó sẽ trị (giải quyết) các bài toán với kích thước nhỏ hơn, cuối cùng kết hợp các kết quả để giải quyết bài toán ban đầu.

Kĩ thuật chia để trị thường sử dụng kĩ thuật đệ quy ở bước chia (Divide) và bước trị (Conquer).

1. Với n = 9 bài toán tìm bi giả cần tối đa bao nhiêu lần cân?

2. Mô tả bước "kết hợp" của bài toán 9 viên bi trên.

2. Thuật toán tìm kiếm nhị phân

Hoạt động 2 **Tìm hiểu kĩ thuật chia để trị thông qua tìm kiếm nhị phân**

Quan sát lại một lần nữa thuật toán tìm kiếm nhị phân trên dãy các phần tử đã sắp xếp và liên hệ với phương pháp chia để trị.

Bài toán gốc của tìm kiếm nhị phân như sau:

Cho trước dãy A các số gồm n phần tử đã được sắp xếp theo thứ tự tăng dần:

A[0] ≤ A[1] ≤ ... ≤ A[n − 1].

Cho trước giá trị cần tìm K. Cần tìm chỉ số k của dãy A sao cho A[k] = K. Nếu tìm thấy trả về k, nếu không thấy trả về −1.

Thuật toán tìm kiếm nhị phân được mô tả bởi hàm đệ quy binarySearch(A, left, right, K) như sau:

```
1 def binarySearch(A, left, right, K):
2     if left <= right:
3         mid = (left + right)//2
4         if A[mid] == K:
5             return mid
6         elif A[mid] < K:
7             return binarySearch(A, mid + 1, right, K)
8         else:
9             return binarySearch(A, left, mid - 1, K)
10    else:
11        return -1
```

− Dòng lệnh 2 là điều kiện để thực hiện lời gọi đệ quy của chương trình. Hàm tìm kiếm chỉ thực hiện khi left ≤ right.

− Dòng 3 có nhiệm vụ chia bài toán tìm kiếm thành hai bài toán con: tìm kiếm nửa bên trái và nửa bên phải. Đây chính là bước Chia (Divide) của kĩ thuật chia để trị.

– Các dòng 7 và 9 thực hiện lệnh gọi đệ quy tương ứng với các nửa bên phải và nửa trái của dãy. Các lệnh này phụ thuộc vào lệnh tại dòng 3. Đây chính là bước Trị (Conquer) của kĩ thuật chia để trị.

– Khi thực hiện xong các dòng 7, 9 thì bài toán đã được giải xong. Ở đây bước Kết hợp (Combine) sẽ được tự động thực hiện ngay sau khi gọi đệ quy.

Thuật toán tìm kiếm nhị phân là một triển khai của kĩ thuật chia để trị của bài toán tìm kiếm trên dãy các phần tử đã sắp xếp.

1. Trong thuật toán tìm kiếm nhị phân trên, phần cơ sở là các lệnh nào?

2. Mô tả các bước thực hiện thuật toán tìm kiếm nhị phân khi left = right.

Hoạt động 3 Đánh giá độ phức tạp thời gian của thuật toán tìm kiếm nhị phân

Đọc, quan sát phân tích sau để biết được tính vượt trội của tìm kiếm nhị phân với tìm kiếm tuần tự, biết được vai trò của kĩ thuật chia để trị trong thiết kế thuật toán.

Gọi T(n) là thời gian thực hiện thuật toán tìm kiếm nhị phân trong chương trình trên, n là độ dài dãy A.

– Nếu n = 1, tức là left = right. Nếu A[left] = K thì sẽ thực hiện ngay các lệnh tại dòng 5 và dừng chương trình. Nếu A[left] ≠ K thì sẽ gọi tiếp đệ quy tới các dòng 7 hoặc 9 nhưng sẽ trả về ngay –1. Vậy trong mọi trường hợp tổng số phép toán thực hiện khi n = 1 chỉ là hằng số, ta có T(1) = O(1).

– Nếu n > 1, nếu A[mid] ≠ K thì chương trình sẽ gọi đệ quy tại các lệnh 7 hoặc 9, các lệnh gọi đệ quy này sẽ chạy trên dãy có kích thước n/2. Như vậy, trong mọi trường hợp chúng ta có công thức sau cho trường hợp tổng quát:

$T(n) = T(n/2) + O(1)$.

Vậy không mất tổng quát có thể giả thiết tồn tại hằng số C sao cho:

$T(1) = C$ (1)

$T(n) = T(n/2) + C$ (2)

Từ các công thức (1), (2) có thể chứng minh được $T(n) = O(\log n)$.

Để tham khảo chúng ta sẽ chứng minh công thức $T(n) = O(\log n)$ trong trường hợp $n = 2^k$, $k = \log_2 n$.

Theo công thức (2) ta có:

$T(n) = T(2^k) = T(2^{k-1}) + C$

$= T(2^{k-2}) + C + C = T(2^{k-2}) + 2C = \ldots$

$= T(2^0) + kC$ // theo công thức (1)

$= (k + 1)C$

$= C.k + C = C.\log_2 n + C = O(\log_2 n)$.

Thuật toán tìm kiếm nhị phân có độ phức tạp thời gian O(logn), tốt hơn hẳn so với tìm kiếm tuần tự với thời gian chạy là O(n).

1. Tìm chính xác số phép toán đơn cần thực hiện trong thuật toán tìm kiếm nhị phân nếu dãy gốc chỉ có 1 phần tử.
2. Tìm số phép toán đơn cần thực hiện trong thuật toán trên nếu dãy có 2 phần tử.

LUYỆN TẬP

1. Viết chương trình hoàn chỉnh nhập một dãy số đơn điệu tăng từ bàn phím, các số cách nhau bởi dấu cách. Sau đó, nhập số K bất kì từ bàn phím và thực hiện việc tìm kiếm số K trong dãy trên. Nếu tìm thấy thì trả lại chỉ số của phần tử có giá trị K, ngược lại trả về −1.
2. Cho trước danh sách gồm có tên, điểm thi và được sắp xếp theo thứ tự tăng dần của điểm thi, ví dụ danh sách: [["Bình", 7.5], ["Hoa", 8], ["An", 9], ["Quang", 10]]. Viết chương trình nhập một điểm số và tìm tên học sinh có điểm thi bằng điểm số đã nhập, nếu không tìm thấy thì thông báo "không có".

VẬN DỤNG

1. Phương án không đệ quy của thuật toán tìm kiếm nhị phân có phải là chia để trị không?
2. Em hãy viết chương trình cài đặt các thuật toán tìm kiếm tuần tự và nhị phân rồi tiến hành đo thời gian thực trên máy tính với hai thuật toán này. Thực hiện kiểm thử với các bộ dữ liệu n = 10, 20, 50, 100 và ghi vào bảng để so sánh thời gian chạy giữa hai thuật toán tìm kiếm này.
3. Để tính giá trị (số nguyên) gần đúng căn bậc hai của số tự nhiên n cho trước, người ta đã thiết lập hàm sau với ý tưởng gần tương tự thuật toán tìm kiếm tuần tự như sau:

```
1  def floorSqrt(n):
2      k = 1
3      while k*k <= n:
4          k = k + 1
5      return k - 1
```

Hãy thiết kế lại thuật toán tìm số nguyên lớn nhất không vượt quá căn bậc hai của n bằng kĩ thuật chia để trị.

BÀI 7 THIẾT KẾ THUẬT TOÁN THEO KĨ THUẬT CHIA ĐỂ TRỊ

Sau bài này em sẽ:

- Biết và thực hiện được một số thiết kế thuật toán bằng kĩ thuật chia để trị.

Trong bài học này em sẽ thiết kế lời giải cho hai bài toán sau:

1. Bài toán tính luỹ thừa exp(a, n) = a^n với a là số bất kì (khác 0), n là số nguyên không âm, ở đây a^n được hiểu là tích của n lần giá trị a: $a^n = a \times a \times ... \times a$ (n lần).

2. Ban giám hiệu nhà trường cần tìm một bạn lớp em có chiều cao đúng bằng 1,7 m hoặc gần với chiều cao đó nhất để tham gia tập đội hình thể thao.

Với hai bài toán trên em sẽ thực hiện như thế nào?

1. Bài toán tính luỹ thừa

Hoạt động 1 **Giải bài toán tính luỹ thừa bằng chia để trị**

Hãy thiết lập thuật toán và chương trình tính luỹ thừa a^n với a là số bất kì khác 0, n là số nguyên không âm.

Vì $a^n = a \times a^{n-1}$ nên có thể thiết lập ngay chương trình tính hàm luỹ thừa đơn giản như sau:

Chương trình 1. Tính luỹ thừa theo cách tự nhiên.

```
1  def exp(a,n):
2      p = 1
3      for i in range(n):
4          p = p*a
5      return p
```

Phân tích.

Chương trình trên chỉ có một vòng lặp tại dòng 3, lệnh thực hiện trong vòng lặp là phép nhân tại dòng 4, từ đó suy ra thời gian thực hiện chương trình là O(n).

Bây giờ chúng ta sẽ thiết lập một lời giải khác, xuất phát từ ý tưởng chia để trị. Xét các trường hợp n là chẵn hay lẻ.

+ Nếu n chẵn, tức là n = 2k ta có:

  $a^n = a^{2k} = (a^k)^2 = (a^{n//2})^2$

+ Nếu n lẻ, tức là n = 2k + 1 ta có:

  $a^n = a^{2k+1} = a.(a^k)^2 = a.(a^{n//2})^2$

Từ các nhận xét trên sẽ dẫn đến công thức truy hồi để tính luỹ thừa a^n.

$$a^n = \begin{cases} 1 & \text{nếu } n = 0 \\ (a^{n//2})^2 & \text{nếu } n > 0, \text{ n chẵn} \\ a.(a^{n//2})^2 & \text{nếu } n > 0, \text{ n lẻ} \end{cases}$$

Từ công thức trên sẽ dẫn đến thuật toán sau tính exp(a, n).

Chương trình 2. Tính luỹ thừa theo cách nhị phân nhanh (chia để trị).

```
1  def exp(a,n):
2      if n == 0:
3          return 1
4      else:
5          b = exp(a,n//2)
6          if n%2 == 0:
7              return b*b
8          else:
9              return a*b*b
```

Phân tích.

Gọi T(n) là thời gian thực hiện thuật toán trên.

– Với n = 0, dòng lệnh 2 cho ta T(0) = 1.

– Tổng quát, với n bất kì, dòng 5 sẽ mất T(n/2) thời gian. Các dòng tiếp theo 7, 9 chỉ là các phép nhân ngắn, tức là chỉ mất O(1) thời gian. Vậy trong trường hợp tổng quát ta có:

T(n) = T(n/2) + O(1).

Theo cách suy luận tương tự như đối với bài toán tìm kiếm nhị phân chúng ta có ngay kết quả T(n) = O(logn), đây là bậc thời gian tốt hơn hẳn bậc tuyến tính của chương trình 1.

Hàm exp(a,n) trên được thiết lập theo kĩ thuật chia để trị có độ phức tạp thời gian O(logn).

1. Mô tả các bước tính bằng tay phép tính luỹ thừa 2^{11} theo hai chương trình trên. Cách nào nhanh hơn?

2. Phép tính a^{21} sẽ cần dùng bao nhiêu phép nhân?

2. Bài toán tìm kiếm nhị phân mở rộng

Hoạt động 2 **Giải bài toán tìm kiếm nhị phân mở rộng bằng chia để trị**

Xây dựng thuật toán cho bài toán sau: Cho trước dãy các số đã được sắp xếp tăng dần. Với giá trị K cho trước cần tìm phần tử của dãy gốc có giá trị gần với K nhất.

Thuật toán thứ nhất dựa trên phương pháp tìm kiếm tuần tự quen thuộc đã biết. Để tìm ra phần tử của dãy gần K nhất chúng ta cần thêm các tính toán phụ tại dòng 10.

Chương trình 1. Sử dụng tìm kiếm tuần tự

```
1  def valueClosest(A,K):
2      n = len(A)
3      if K <= A[0]:
4          return A[0]
5      elif K >= A[n - 1]:
6          return A[n - 1]
7      else:
8          value = A[0]
9          for i in range(n):
10             if abs(A[i] - K) < abs(value - K):
11                 value = A[i]
12         return value
```

Chương trình trên hoàn toàn tương tự thuật toán tìm kiếm tuần tự, chỉ có một vòng lặp tại dòng 9, do đó có thời gian chạy O(n).

Chúng ta sẽ thiết kế thuật toán thứ hai dựa trên tìm kiếm nhị phân. Hàm đệ quy chính sẽ được thiết kế theo dạng valueClosest(A, left, right, K) sẽ tìm phần tử gần K nhất trong khoảng chỉ số từ left đến right. Trước tiên có một số nhận xét cho các trường hợp đặc biệt:

– Nếu n = 0, dãy A rỗng, hàm không có giá trị nào cần trả lại.

– Nếu n = 1, dãy A chỉ có một phần tử, khi đó hàm sẽ trả lại phần tử duy nhất của A.

– Nếu n = 2, dãy A có hai phần tử thì cần so sánh phần tử nào gần K hơn chính là phần tử cần tìm.

Trong trường hợp tổng quát, cách tìm hoàn toàn tương tự tìm kiếm nhị phân, với mỗi bước lặp sẽ gọi đệ quy với kích thước dãy giảm đi một nửa.

Chương trình cuối cùng có dạng như sau:

Chương trình 2. Sử dụng tìm kiếm nhị phân

```
1  def valueClosest(A,left,right,K):
2    if left > right:
3        return
4    elif left == right:
5          return A[left]
6      elif left == right - 1:
7          if abs(A[left] - K) < abs(A[right] - K):
8              return A[left]
9          else:
10              return A[right]
11     else:
12         mid = (left + right)//2
```

```
13          if A[mid] == K:
14              return A[mid]
15          elif K < A[mid]:
16              return valueClosest(A,left,mid,K)
17          else:
18              return valueClosest(A,mid,right,K)
```

Lệnh gọi hàm đệ quy là:

```
valueClosest(A, 0, len(A) - 1, K)
```

Phân tích.

- Với n = 1, dòng 4 cho ta T(n) = 1.
- Với n = 2, dòng 6 cần tính hai phép trừ và hàm abs() nên sẽ có thời gian O(1).
- Trường hợp tổng quát, các dòng lệnh từ 10 đến 16 cho ta T(n) = T(n/2) + O(1).

Sử dụng cách suy luận tương tự của tìm kiếm nhị phân chúng ta sẽ có T(n) = O(logn).

Thuật toán tìm kiếm nhị phân mở rộng có độ phức tạp thuật toán O(logn).

1. Hãy giải thích kĩ hơn chương trình 2 trên tại các dòng 4 và 6.

2. Nêu những điểm khác biệt của chương trình trên với chương trình tìm kiếm nhị phân đã biết.

LUYỆN TẬP

1. Viết chương trình không đệ quy cho bài toán tìm kiếm nhị phân mở rộng trên.

2. Viết chương trình đo thời gian thực chạy để so sánh hai phương án của bài toán tìm kiếm mở rộng.

VẬN DỤNG

1. Tìm cách thiết lập thuật toán tính a^n theo phương pháp chia để trị nhưng không sử dụng đệ quy.

2. Bài toán tìm vùng chỉ số của dãy đã sắp xếp.

Thiết lập thuật toán chia để trị để giải bài toán sau: Cho trước dãy A gồm n phần tử đã được sắp xếp theo thứ tự tăng dần, ví dụ:

A = [1, 2, 3, 3, 4, 4, 4, 5, 6, 6]

Cho trước giá trị K, cần tìm ra vùng chỉ số gồm các phần tử bằng K. Chương trình cần trả về hai chỉ số start, end là vị trí bắt đầu và kết thúc gồm toàn các giá trị K. Nếu không tìm thấy K thì phải trả về −1, −1.

Trong ví dụ trên, nếu K = 4 thì cần trả về hai chỉ số 4, 6.

BÀI 8 THỰC HÀNH THIẾT KẾ THUẬT TOÁN TÌM KIẾM THEO KĨ THUẬT CHIA ĐỂ TRỊ

Sau bài này em sẽ:

- Biết cách thiết kế và viết chương trình giải một số bài toán theo kĩ thuật chia để trị.

Cho một dãy số A bất kì. Để xác định một số C cho trước xuất hiện trong dãy A bao nhiêu lần thì làm thế nào?

NHIỆM VỤ 1 Tìm số lần lặp của một giá trị trên dãy đã sắp xếp.

Cho trước dãy số A đã sắp xếp theo thứ tự tăng dần, cho trước hằng số C. Hãy tìm xem trong dãy trên có bao nhiêu số hạng bằng C. Ví dụ nếu A = [0, 1, 2, 2, 2, 2, 4, 5, 5, 6], C = 2 thì kết quả cần tìm là 4.

Hướng dẫn:

Ý tưởng: Trước tiên, nhận xét rằng bài tập này có thể dễ dàng giải bằng phương pháp tìm kiếm tuần tự đã quen biết. Gọi count là số lần xuất hiện của C trong dãy. Thực hiện tìm kiếm tuần tự với C, mỗi lần tìm thấy C, tăng biến count lên 1.

Chương trình đơn giản như sau:

```
1   def countNum(A,C):
2         count = 0
3         for i in range(len(A)):
4               if A[i] == C:
5                     count = count + 1
6         return count
```

Thuật toán trên có một vòng lặp tại dòng 3 thực hiện n lần (n = len(A)), do đó thời gian chạy là O(n).

Bây giờ chúng ta sẽ thiết lập lời giải bài toán trên theo kĩ thuật chia để trị. Vì dãy ban đầu đã được sắp xếp theo thứ tự tăng dần nên ý tưởng của lời giải là theo cách làm của thuật toán tìm kiếm nhị phân. Gọi n là kích thước của dãy số gốc.

Chúng ta sẽ thiết lập hàm tìm kiếm dạng countNum(A, left, right, C) để thực hiện tìm kiếm số lần lặp của C trong khoảng chỉ số từ left đến right.

– Trường hợp left > right (dãy ban đầu rỗng) thì lập tức trả về 0.

– Xét trường hợp left <= right. Gọi mid = (left + right)//2 là chỉ số phần tử ở giữa dãy đang tìm. Việc tìm kiếm tiếp sẽ theo các bước sau:

+ Nếu A[mid] = C thì tìm số các phần tử bằng C bằng cách duyệt các phần tử trước và sau mid.

+ Nếu A[mid] < C thì cần tìm tiếp trong khoảng nửa dãy bên phải từ mid + 1 đến right.

+ Nếu A[mid] > C thì cần tìm tiếp trong khoảng nửa dãy bên trái từ left đến mid – 1.

Chương trình hoàn chỉnh có thể như sau:

```
def countNum(A,left,right,C):
    if left > right:
        return 0
    else:
        mid = (left+right)//2
        if A[mid] == C:
            start = mid
            while start >= left and A[start] == C:
                start = start - 1
            end = mid
            while end <= right and A[end] == C:
                end = end + 1
            return end - start - 1
        elif A[mid] < C:
            return countNum(A,mid+1,right,C)
        else:
            return countNum(A,left,mid-1,C)
```

Lệnh gọi đệ quy của hàm tìm kiếm là:

```
countNum(A,0,len(A) - 1,C)
```

Nhận xét:

– Chương trình trên hoàn toàn tương tự thuật toán tìm kiếm nhị phân, điểm khác biệt tại các dòng lệnh từ 6 đến 13 khi xử lí trường hợp A[mid] = C.

– Các dòng 2, 3 là phần cơ sở của đệ quy.

– Việc "chia" được thực hiện tại dòng 5. Các lệnh tiếp theo chính là "trị". Bài toán này khá đơn giản nên sau khi "trị" sẽ thu được luôn kết quả.

– Trong hầu hết các trường hợp việc xử lí tại dòng 6 đến dòng 13 sẽ mất O(1) thời gian. Trong một số trường hợp xấu nhất, ví dụ dãy ban đầu bao gồm toàn các số C thì việc xử lí khi A[mid] = C sẽ mất O(n) thời gian.

LUYỆN TẬP

Chỉnh sửa nâng cấp chương trình của nhiệm vụ thực hành để đưa ra kết quả là vùng các phần tử có giá trị bằng C của dãy gốc, tức là yêu cầu đưa ra chỉ số đầu, chỉ số cuối và số lượng phần tử của vùng có giá trị bằng C.

Ví dụ nếu A = [0, 1, 2, 2, 2, 2, 4, 5, 5, 6], C = 2, thì kết quả trả lại là 2, 5, 4.

VẬN DỤNG

1. Cho một dãy số bất kì (chưa được sắp xếp) và một số K, hãy tìm số lần xuất hiện của K trong dãy số trên. Yêu cầu sử dụng phương pháp chia để trị.

2. Cho một dãy số nguyên được sắp xếp theo thứ tự tăng dần, hãy tìm một vị trí thứ i trong dãy sao cho phần tử thứ i có giá trị bằng i.

3. Cho trước dãy số A đã sắp xếp theo thứ tự tăng dần, cho trước hằng số C. Cần thiết lập hai hàm sau bằng kĩ thuật chia để trị:

– Hàm firstInd(A, left, right, C) sẽ tìm chỉ số của phần tử đầu tiên của dãy A có giá trị bằng C. Nếu không thấy sẽ trả về –1.

– Hàm lastInd(A, left, right, C) sẽ tìm chỉ số của phần tử cuối cùng của dãy A có giá trị bằng C. Nếu không thấy sẽ trả về –1.

Từ hai hàm đã thiết kế trên, đưa ra một cách giải khác cho bài toán trong nhiệm vụ 1. Lời giải này có độ phức tạp O(logn).

BÀI 9 SẮP XẾP TRỘN

Sau bài này em sẽ:

- Biết và trình bày được cách thiết kế thuật toán sắp xếp trộn bằng kĩ thuật chia để trị.

Ta đã biết tìm kiếm nhị phân trên các dãy đã sắp xếp có độ phức tạp thời gian tốt hơn so với các thuật toán tìm kiếm trên dãy chưa sắp xếp. Chính vì thế, việc sắp xếp thông tin theo một trình tự nào đó luôn đóng vai trò quan trọng trong các bài toán tìm kiếm thông tin. Tuy nhiên, một số thuật toán sắp xếp mà em đã biết như sắp xếp chèn, sắp xếp chọn, sắp xếp nổi bọt,... đều có độ phức tạp thời gian $O(n^2)$ (n – kích thước dãy cần sắp xếp). Câu hỏi đặt ra là: Liệu có hay không một cách sắp xếp dãy với thời gian tốt hơn $O(n^2)$?

Liệu kĩ thuật chia để trị có thể áp dụng cho bài toán sắp xếp được không? Nếu có thì có làm tăng hiệu quả của sắp xếp được không?

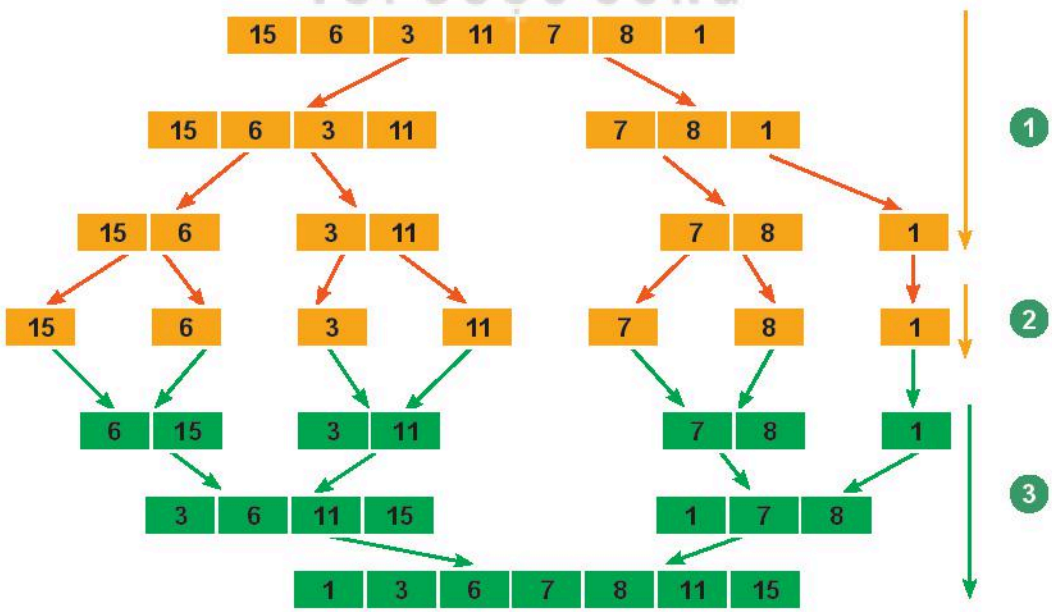
1. Ý tưởng thuật toán sắp xếp trộn

Hoạt động 1 **Tìm hiểu ý tưởng sắp xếp trộn**

Quan sát và thực hiện các bước theo ý tưởng của thuật toán sắp xếp trộn để biết thuật toán này là mô hình kĩ thuật chia để trị.

Em có nhận xét gì về đặc thù của các giai đoạn 1, 2, 3 trong sơ đồ dưới đây.

Quan sát các bước thực hiện sắp xếp dãy số ban đầu: 15, 6, 3, 11, 7, 8, 1.



Hình 9.1. *Mô phỏng sắp xếp trộn*

Giai đoạn 1. Từ dãy gốc ban đầu chúng ta tách chia đôi làm hai dãy con, mỗi dãy con có kích thước bằng $\frac{1}{2}$ kích thước của dãy gốc. Quá trình này chính là bước Chia (divide) của kĩ thuật chia để trị.

Giai đoạn 2. Khi tất cả các dãy con thu được đều chỉ còn một phần tử. Tất cả các dãy này hiển nhiên đều đã được sắp xếp đúng. Vậy việc trị đã xong. Đây chính là bước Trị (conquer) tương ứng của chiến lược chia để trị.

Giai đoạn 3. Từ các dãy đã sắp xếp xong, chúng ta sẽ trộn chúng lại với nhau, mỗi lần trộn hai dãy đã sắp xếp để tạo thành một dãy lớn hơn cũng được sắp xếp đúng. Quá trình trộn sẽ kết thúc khi nhận được đúng một dãy chính là dãy ban đầu nhưng đã sắp xếp xong. Đây chính là quá trình Kết hợp (combine) tương ứng của kĩ thuật chia để trị.

Ý tưởng của thuật toán sắp xếp trộn được thực hiện qua 3 bước: Chia nhỏ dãy gốc thành các dãy với kích thước nhỏ hơn (bằng $\frac{1}{2}$ dãy ban đầu), tiếp tục cho đến khi nhận được các dãy con đã sắp xếp đúng thì tiến hành trộn các dãy này để nhận được kết quả cuối cùng. Các bước trên chính là kĩ thuật chia để trị.

1. Hãy thực hiện thao tác trộn hai dãy sau: B = 1, 4, 7; C = 2, 3, 6.

2. Thực hiện sắp xếp dãy 3, 2, 7, 1, 6, 5 theo các bước tương tự trên.

2. Mô tả thuật toán sắp xếp trộn

Hoạt động 2 **Tìm hiểu các bước thực hiện sắp xếp trộn**

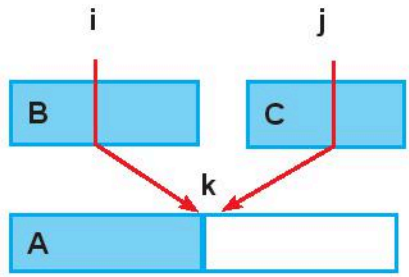
Cùng thực hiện các bước cụ thể triển khai ý tưởng và cài đặt chương trình cho thuật toán sắp xếp trộn.

Từ các ý tưởng của thuật toán sắp xếp trộn trong hoạt động trước, chúng ta sẽ tìm hiểu và thiết lập các thuật toán và chương trình có liên quan.

a) Thuật toán trộn hai dãy (merge algorithm)

Giả sử cho trước hai dãy đã được sắp thứ tự B và C có số phần tử lần lượt là m, n. Thuật toán merge(B, C) sẽ thực hiện "trộn" hai dãy trên và đưa kết quả vào dãy A. Quy trình trộn này rất tự nhiên và được minh hoạ trong sơ đồ sau:

Thiết lập sẵn dãy A bao gồm m + n phần tử. Chúng ta cần 3 biến chỉ số i, j, k chạy trên các dãy B, C và A. Tại mỗi bước lặp, so sánh B[i] và C[j], nếu B[i] < C[j] thì đưa B[i] vào vị trí A[k], tăng i lên 1 đơn vị. Ngược lại, đưa C[j] vào A[k], tăng j lên 1 đơn vị. Sau mỗi bước tăng sẽ tăng k lên 1 đơn vị. Quá trình này liên tục cho đến khi i hoặc j đi qua vị trí cuối dãy. Khi đó phần dãy còn lại sẽ được đưa nốt vào A. Do tính chất đã sắp xếp sẵn của B, C nên A thu được cũng được sắp xếp và là kết quả của phép trộn B và C. Một đặc tính quan trọng của thuật toán này là thời gian chạy chỉ O(m + n) vì chỉ duyệt trên chiều dài của B và C cộng lại.



Các biến i, j, k ban đầu đặt tại đầu của các dãy B, C, A.

Hình 9.2. *Trộn hai dãy B và C*

Thuật toán 1. Thuật toán trộn hai dãy (đã được sắp xếp).

```
1 def merge(B,C):
2       m = len(B)
3       n = len(C)
4       A = [0] * (m+n) # Thiết lập dãy A bao gồm m+n số 0
5       i = j = k = 0
6       while i < m and j < n:          <- Bắt đầu duyệt tại đây.
7           if B[i] < C[j]
8               A[k] = B[i]
9               i = i + 1
10          else:
11              A[k] = C[j]
12              j = j + 1
13          k = k + 1
14      if i == m:
15          while j < n:                <- Nếu đã duyệt xong B thì đưa nốt phần còn lại của C vào A.
16              A[k] = C[j]
17              j = j + 1
18              k = k + 1
19      else:
20          while i < m:                <- Nếu đã duyệt xong C thì đưa nốt phần còn lại của B vào A.
21              A[k] = B[i]
22              i = i + 1
23              k = k + 1
24      return A
```


b) Thuật toán sắp xếp trộn (mergeSort)

Thuật toán sắp xếp trộn được mô tả bởi hàm mergeSort(A) trong chương trình sau. Các dòng lệnh 2, 3 là phần cơ sở của đệ quy. Dòng lệnh 5 thực hiện chia dãy A làm hai phần có kích thước hơn kém nhau không quá 1. Các lệnh gọi đệ quy 6, 7 chính là phép trị để nhận được hai dãy đã sắp xếp xong có kích thước nhỏ hơn. Lệnh 8 sử dụng hàm trộn merge(B, C) thực hiện việc kết hợp để nhận được kết quả cuối cùng.

Thuật toán 2. Thuật toán sắp xếp trộn.

```
1 def mergeSort(A):
2       if len(A) <= 1:
3           return A
4       else:
5           k = len(A)//2
6           B = mergeSort(<dãy con A từ 0 đến k>)
7           C = mergeSort(<dãy con A từ k+1 đến len(A)-1>)
8           return merge(B, C)
```

Đoạn chương trình sau sẽ thực hiện sắp xếp dãy A và đưa kết quả đã sắp xếp vào dãy B. Lời gọi hàm đệ quy là:

```
1 A = [2, 1, 10, 0, -7, -20, 19, 100, -3, -2, 9, 11]
2 B = mergeSort(A)
```

Lưu ý: Có nhiều phương án thực hiện sắp xếp trộn khác nhau, trên đây chỉ mô tả một trong các phương án cài đặt của thuật toán.

Thuật toán sắp xếp trộn sẽ bao gồm một hàm mergeSort(), hàm này sẽ tiến hành các bước chính của thuật toán và gọi hàm trộn hai dãy đã sắp xếp merge() khi thực hiện bước kết hợp.

1. Trong chương trình 1 (trộn hai dãy B, C), vòng lặp tại dòng 6 có nhiều nhất là bao nhiêu bước lặp?

2. Mô tả các bước thực hiện chương trình 2 nếu dãy gốc A chỉ có một phần tử.

3. Đánh giá thuật toán sắp xếp trộn

Hoạt động 3 **Tìm hiểu đánh giá độ phức tạp thời gian của sắp xếp trộn**

Phân tích, trao đổi, thảo luận để tính độ phức tạp thời gian của thuật toán sắp xếp trộn.

Phân tích: Gọi T(n) là thời gian chạy của thuật toán sắp xếp trộn.

Với n = 1, dòng lệnh 2 trả lại ngay dãy gốc A, do đó T(1) = 1.

Trường hợp tổng quát:

- Tại bước chia (dòng 5), cần O(1) thời gian để thực hiện.
- Các dòng 6, 7 sẽ mất 2T(n/2) thời gian.
- Dòng lệnh 8 thực hiện trộn hai dãy với thời gian O(n).

Tổng kết lại chúng ta các công thức sau tính thời gian T(n).

$T(1) = 1$

$T(n) = 2T(n/2) + O(n), n > 1$ (1)

Không mất tổng quát, giả sử tồn tại hằng số $C > 0$ sao cho:

$T(n) = 2T(n/2) + Cn, n > 1$ (2)

Các công thức (1), (2) được gọi là công thức truy hồi để tính độ phức tạp thời gian T(n) của thuật toán trộn.

Người ta tính được: T(n) = O(nlogn).

> Thuật toán sắp xếp trộn sử dụng kĩ thuật chia để trị có độ phức tạp thời gian O(nlogn), bậc thời gian này là tốt hơn hẳn các thuật toán sắp xếp mà chúng ta đã biết (sắp xếp chèn, sắp xếp chọn và sắp xếp nổi bọt).

1. Tính thời gian chạy của thuật toán sắp xếp trộn nếu A = [3, 1].

2. Mô tả thực hiện các bước của sắp xếp trộn với dãy A = [5, 1, 7, 4]. Trường hợp này T(4) sẽ được tính như thế nào?

LUYỆN TẬP

1. Viết chương trình thực hiện công việc sau:

– Dữ liệu được nhập từ tệp văn bản Data.inp bao gồm hai dòng, mỗi dòng là một dãy các số nguyên đã được sắp xếp theo thứ tự tăng dần, các số cách nhau bởi dấu cách. Hai dãy này có thể không bằng nhau về kích thước.

– Chương trình sẽ thực hiện trộn hai dãy trên và đưa kết quả dãy được trộn ra tệp Data.out theo một hàng ngang.

2. Viết lại chương trình hoàn chỉnh thực hiện sắp xếp trộn với dãy A cho trước trên một tệp văn bản. Kết quả đưa ra màn hình.

VẬN DỤNG

1. Viết chương trình hoàn chỉnh nhập dãy số bất kì từ tệp văn bản, sau đó tiến hành sắp xếp dãy số này theo hai cách khác nhau, cách 1 là một trong các thuật toán sắp xếp đã biết, cách 2 là sắp xếp trộn. Đo thời gian chạy thực sự của hai cách trên, so sánh và kết luận về ưu điểm của sắp xếp trộn.

2. Viết lại thuật toán sắp xếp trộn theo cách thực hiện trực tiếp trên dãy số A cho trước, cụ thể như sau.

– Thủ tục trộn sẽ có dạng sau: merge(A,left,mid,right). Thủ tục này sẽ trộn hai phân đoạn của dãy A là A[left],..., A[mid] và A[mid + 1],..., A[right]. Hai phân đoạn này phải được sắp xếp đúng trước đó.

– Thuật toán chính có dạng mergeSoft(A, left, right) như sau:

```
def mergeSort(A, left, right):
    if left < right:
        mid = (left + right)//2
        mergeSort(A, left, mid)
        mergeSort(A, mid + 1, right)
        merge(A, left, mid, right)
```

– Lệnh gọi hàm đệ quy là:

```
mergeSort(A, 0, len(A) - 1)
```

BÀI 10 THỰC HÀNH GIẢI TOÁN BẰNG KĨ THUẬT CHIA ĐỂ TRỊ

Sau bài này em sẽ:

- Biết cách thiết kế và viết chương trình giải một bài toán theo phương pháp chia để trị.

Khi làm việc với các danh sách/mảng, nhiều trường hợp đòi hỏi cần kiểm ra các danh sách mảng đã được sắp thứ tự để áp dụng thuật toán phù hợp. Cho một dãy số, theo em làm thế nào để xác định dãy số đã được sắp xếp theo thứ tự tăng dần hoặc giảm dần?

NHIỆM VỤ Đếm số cặp nghịch đảo (inversion) của dãy số.

Cho một dãy số A bất kì. Hãy đếm số cặp nghịch đảo của dãy số đó. Một cặp phần tử A[i] và A[j] được gọi là nghịch đảo nếu i < j và A[i] > A[j] .

Ví dụ dãy A = [4, 5, 2, 10, 4] sẽ có 4 cặp nghịch đảo là: (4, 2), (5, 2), (5, 4), (10, 4).

Hướng dẫn:

Cách 1. Phương pháp duyệt đơn giản

Ý tưởng:

– Duyệt lần lượt từng phần tử của dãy số.

– Với mỗi phần tử đang xét A[i], thực hiện so sánh với tất cả các phần tử đứng sau nó A[j], nếu A[i] > A[j] ta sẽ tăng số cặp nghịch đảo lên 1 đơn vị.

Chương trình 1.

```
1   def getInvCount(A):
2         n = len(A)
3         inv_count = 0
4         for i in range(n-1):
5               for j in range(i + 1, n):
6                     if (A[i] > A[j]):
7                           inv_count = inv_count + 1
8         return inv_count
```

Thuật toán trên sử dụng hai vòng for lồng nhau tại dòng 4 và 5, do đó thời gian chạy là $O(n^2)$.

Cách 2. Sử dụng phương pháp chia để trị dựa trên thuật toán sắp xếp trộn mergesort

Ý tưởng: Thực hiện thuật toán sắp xếp trộn trên dãy đã cho, trong quá trình trộn sẽ đồng thời tiến hành đếm số các cặp phần tử nghịch đảo.

Các bước chính thực hiện giải bài toán này như sau:

- Chia dãy thành hai phần bằng nhau cho đến khi mỗi dãy chỉ còn 1 phần tử.

- Khi tiến hành hàm trộn hai dãy sẽ đồng thời đếm số các cặp nghịch đảo của hai dãy này. Kết quả vẫn tạo được dãy trộn như đã mô tả trong thuật toán sắp xếp trộn, vừa đếm được số nghịch đảo.

- Gọi đệ quy đếm số cặp nghịch đảo cho hai nửa bên trái và bên phải, tính tổng số cặp nghịch đảo khi trộn hai dãy. Kết quả sẽ thu được tổng số cặp nghịch đảo cần tìm.

Chương trình chính như sau:

```
def getInvCount(A,left,right):
    if left >= right:
        return 0
    else:
        mid = (left + right)//2
        countL = getInvCount(A,left,mid)
        countR = getInvCount(A,mid + 1,right)
        countM = merge(A,left,mid,right)
    return countL + countR + countM
```

Lưu ý:

- countL là số các cặp nghịch đảo của dãy con bên trái (A, left, mid).

- countR là số các cặp nghịch đảo của dãy con bên phải (A, mid+1, right).

- countM là số cặp nghịch đảo mà chỉ số đầu nằm bên trái, chỉ số sau nằm bên phải.

Tổng số các giá trị trên chính là đáp án cần tìm.

Chương trình sau sẽ tiến hành trộn hai nửa dãy A từ left đến mid và từ mid + 1 đến right, đồng thời đếm được số các cặp nghịch đảo dạng (i, j), trong đó i nằm trong [left, mid] và j nằm trong khoảng [mid + 1, right]. Mảng phụ temp_arr[ ] dùng để lưu và cập nhật các thay đổi trên dãy A.

```
def merge(A,left,mid,right):
    i = left
    j = mid + 1
    k = left
    inv_count = 0
    while i <= mid and j <= right:
        if A[i] <= A[j]:
            temp_arr[k] = A[i]
            k = k + 1
            i = i + 1
        else:
            temp_arr[k] = A[j]
            inv_count = inv_count + (mid – i + 1)
            k = k + 1
            j = j + 1
```

```
16      while i <= mid:
17          temp_arr[k] = A[i]
18          k = k + 1
19          i = i + 1
20      while j <= right:
21          temp_arr[k] = A[j]
22          k = k + 1
23          j = j + 1
24      for k in range(left,right + 1):
25          A[k] = temp_arr[k]
26      return inv_count
```

Lưu ý:

– Nếu gặp trường hợp A[i] > A[j], thì sẽ suy ra tất cả các số A[i], A[i + 1],..., A[mid] đều lớn hơn A[j], do đó tất cả các căp (i, j), (i + 1, j),..., (mid, j) đều là nghịch đảo. Vậy suy ra số lượng cặp chỉ số nghịch đảo sẽ tăng lên (mid – i + 1) tại thời điểm này. Điều này được mô tả tại dòng lệnh 13.

– Các dòng lệnh 24, 25 sẽ cập nhật lại dãy A từ dãy tạm temp_arr sau khi đã tiến hành trộn. Dãy temp_arr được cập nhật trong quá trình trộn tại các lệnh 8, 12, 17, 21.

Lệnh gọi trong chương trình chính:

```
1   A = [4,5,2,10,4]
2   n = len(A)
3   temp_arr = [0]*n
4   result = getInvCount(A,0,n – 1)
```

Phân tích tương tự như thuật toán mergeSort thuật toán getInvCount trên có độ phức tạp là O(nlogn).

LUYỆN TẬP

Nâng cấp chương trình của nhiệm vụ 1 với yêu cầu bổ sung: Cần đưa ra kết quả là số lượng các cặp nghịch đảo và toàn bộ dãy các cặp chỉ số nghịch đảo đã tìm thấy. Ví dụ với A = [4, 5, 2, 10, 4] thì chương trình sẽ đưa ra giá trị 4 và dãy [(4, 2), (5, 2), (5, 4), (10, 4)].

VẬN DỤNG

1. Cho dãy số A, cần tìm phần tử mốt (mode) của A. Phần tử mốt là phần tử có số lần xuất hiện nhiều nhất trong A. Nếu tồn tại nhiều thì chỉ yêu cầu tìm ra một phần tử mốt. Yêu cầu sử dụng kĩ thuật chia để trị.

2. Cho một dãy số bất kì A[0], A[1],..., A[n – 1]. Một tổng con được định nghĩa là tổng của một dãy con liên tục dạng S(i, j) = A[i] + A[i + 1] + ... + A[j]. Bài toán yêu cầu tìm và chỉ ra một tổng con và dãy con tương ứng có giá trị lớn nhất. Yêu cầu sử dụng kĩ thuật chia để trị.

Chuyên đề 3

THỰC HÀNH THIẾT KẾ THUẬT TOÁN THEO KĨ THUẬT DUYỆT

BÀI 11 BÀI TOÁN TÌM KIẾM VÀ KĨ THUẬT DUYỆT

Sau bài này em sẽ:

- Nêu được ý tưởng của kĩ thuật duyệt và ví dụ minh hoạ.

Để xác định một giá trị a có xuất hiện trong một dãy A cho trước hay không ta có thể áp dụng phương pháp tìm kiếm tuần tự: lần lượt so sánh a với từng phần tử trong A. Theo em, liệu có cách nào để giải bài toán này trong trường hợp A là một dãy bất kì hay không?

1. Kĩ thuật duyệt trong bài toán tìm kiếm

Hoạt động 1 **Kĩ thuật duyệt trong bài toán tìm kiếm**

Cho A, B, C, D lần lượt là các danh sách tên học sinh, điểm thi môn Toán, điểm thi môn Vật lí và điểm thi môn Hoá học. Danh sách điểm Toán được sắp xếp theo thứ tự tăng dần và các danh sách tên học sinh và điểm các môn còn lại được sắp xếp theo tương ứng.

A = ["Nam", "Sơn", "Hương","Huyền", "Hà", "Hùng"]

B = [8.3, 8.4, 8.7, 8.9, 9.1, 9.6]

C = [8.3, 7.8, 8.9, 9.5, 9.3, 9.0]

D = [7.9, 9.0, 8.9, 8.2, 9.5, 9.1]

Hãy thảo luận về kĩ thuật tìm kiếm được thực hiện với mỗi yêu cầu sau:

a) Tìm một học sinh có điểm Toán lớn hơn điểm Vật lí.

b) Tìm tất cả các học sinh có điểm Vật lí lớn hơn điểm Hoá học.

c) Tìm tất cả các học sinh có cả 3 điểm đều lớn hơn hoặc bằng 9.

Phần lớn các bài toán tìm kiếm trong thực tế đều có thể giải quyết bằng cách sử dụng kĩ thuật duyệt. Sau khi xác định được miền tìm kiếm, kĩ thuật duyệt sẽ lần lượt xét các dữ liệu trong miền tìm kiếm để đưa ra được phần tử thoả mãn. Các bài toán tìm kiếm có thể:

– Đa dạng về yêu cầu tìm kiếm như tìm một nghiệm, tìm nghiệm theo yêu cầu, tìm nghiệm tối ưu, tìm tất cả các nghiệm,...

– Đa dạng về miền tìm kiếm như mảng 1 chiều, mảng 2 chiều, danh sách liên kết, dữ liệu chưa sắp xếp, dữ liệu đã sắp xếp,…

Tuỳ thuộc vào yêu cầu tìm kiếm, miền tìm kiếm, người ta thực hiện các kĩ thuật duyệt theo những cách khác nhau.

Ví dụ, với bài toán yêu cầu tìm kiếm một nghiệm, có thể xét những phần tử có nhiều khả năng nhất và dừng lại khi tìm thấy nghiệm đầu tiên. Với bài toán tìm nghiệm tối ưu hoặc tìm tất cả các nghiệm, chúng ta cần duyệt tất cả các phần tử trong miền tìm kiếm.

Với miền tìm kiếm là danh sách thông thường, chưa được sắp xếp, ta có thể thực hiện tìm kiếm tuần tự. Với miền tìm kiếm là danh sách đã được sắp xếp, ta có thể thực hiện thuật toán tìm kiếm nhị phân để tăng tốc độ tìm kiếm.

Trong yêu cầu a, của Hoạt động 1, chỉ cần tìm một học sinh có điểm Toán lớn hơn điểm Vật lí nên có thể thực hiện tìm kiếm duyệt tuần tự từ học sinh đầu tiên trong dãy. Khi gặp học sinh có điểm Toán không lớn hơn điểm Vật lí thì tiếp tục duyệt học sinh tiếp theo. Khi gặp học sinh đầu tiên có điểm Toán lớn hơn điểm Vật lí thì dừng chương trình và in tên học sinh đó lên màn hình.

Trong yêu cầu b, của Hoạt động 1, cần tìm tất cả các học sinh có điểm Vật lí lớn hơn điểm Hoá học. Như vậy, ta cần duyệt tất cả các học sinh, ngay cả khi đã tìm thấy một học sinh có điểm Vật lí lớn hơn điểm Hoá học thì chương trình vẫn tiếp tục duyệt để tìm ra tất cả học sinh thoả mãn điều kiện.

Trong yêu cầu c, của Hoạt động 1, do danh sách tên và điểm các môn khác được sắp xếp theo điểm Toán tăng dần nên ta có thể duyệt từ cuối dãy và dừng tìm kiếm khi gặp điểm Toán nhỏ hơn 9.

Chương trình mô tả thuật toán tìm kiếm theo ba yêu cầu trên có thể được thực hiện như sau:

```
A = ["Nam", "Sơn", "Hương","Huyền", "Hà", "Hùng"]
B = [ 8.3, 8.4, 8.7, 8.9, 9.1, 9.6]
C = [ 8.3, 7.8, 8.9, 9.5, 9.3, 9.0]
D = [7.9, 9.0, 8.9, 8.2, 9.5, 9.1]
kqa=""
for i in range(0, len(A)):
    if B[i] > C[i]:
            kqa=A[i]
            break
if kqa !="":
  print ("Tên một học sinh có điểm Toán lớn hơn điểm Vật lí là:", kqa)
kqb = []
for i in range (0, len(A)):
    if C[i] > D[i]:
            kqb.append(A[i])
```

```
if len(kqc) > 0:
  print ("Tên các học sinh có điểm Vật lí lớn hơn điểm Hoá là:", kqb)
kqc = []
for i in range (len(A)-1, -1, -1):
   if B[i] >= 9:
        if C[i] >= 9 and D[i] >= 9:
            kqc.append(A[i])
    else:
        break
if len(kqb) > 0:
  print ("Tên các học sinh có cả ba điểm lớn hơn 9 là:", kqc)
```

Các bài toán tìm kiếm có thể được giải quyết bằng cách sử dụng kĩ thuật duyệt. **Kĩ thuật duyệt** là lần lượt kiểm tra các phần tử trong miền tìm kiếm để xác định xem phần tử đó có thoả mãn điều kiện tìm kiếm hay không. Tuỳ vào yêu cầu tìm kiếm, miền tìm kiếm mà kĩ thuật duyệt có thể được thiết kế theo các cách khác nhau.

1. So sánh số vòng lặp cần thực hiện để thực hiện các yêu cầu a, b, c trong ví dụ trên.
2. Phân tích và viết chương trình để thực hiện các yêu cầu sau:

a) Tìm học sinh có điểm Toán bằng 8.9.

b) Tìm một học sinh có tổng điểm ba môn Toán, Vật lí, Hoá học lớn hơn 26.5.

c) Tìm học sinh có tổng điểm ba môn Toán, Vật lí, Hoá học nhỏ nhất.

2. Tìm kiếm vét cạn

Hoạt động 2 Khái niệm duyệt vét cạn

Với các bài toán sau, em hãy thảo luận với bạn để tìm kĩ thuật tìm kiếm đã học (tìm kiếm trên các mảng 1 hoặc 2 chiều) để giải.

1. Cho trước số tự nhiên n. Tìm và in ra tất cả các xâu nhị phân có độ dài n.

2. Viết chương trình tìm và liệt kê tất cả các hoán vị của tập hợp [1, 2,..., n] với n là số tự nhiên cho trước.

Tuỳ thuộc vào yêu cầu tìm kiếm, ta có thể duyệt một phần hoặc duyệt toàn bộ các phần tử trong miền tìm kiếm. Kĩ thuật duyệt toàn bộ dữ liệu trong miền tìm kiếm được gọi là *duyệt vét cạn*. Kĩ thuật duyệt vét cạn có thể áp dụng cho hầu hết các bài toán tìm kiếm. Chúng ta cùng quan sát và phân tích khả năng áp dụng kĩ thuật duyệt vét cạn cho các bài toán trên.

Bài toán 1. Miền tìm kiếm là toàn bộ các xâu nhị phân có độ dài n hay tập hợp các dãy kí tự có dạng s[0], s[1],..., s[n – 1], trong đó mỗi kí tự s[k] sẽ bằng "0" hoặc "1". Số lượng các dãy như vậy là 2^n. Giá trị này quá lớn để có thể lưu dữ liệu vào một mảng 1 chiều hay 2 chiều. Do vậy sử dụng kĩ thuật duyệt vét cạn cho bài toán này là rất khó.

Bài toán 2. Miền tìm kiếm của bài toán là các dãy số A[0], A[1],..., A[n – 1], với mỗi dãy là một hoán vị của [1, 2,..., n]. Số lượng các hoán vị là n!, là một số rất lớn. Do vậy cũng rất khó áp dụng kĩ thuật duyệt vét cạn cho bài toán này.

Hai bài toán trên là ví dụ cho lớp những bài toán thực tế khó có thể thực hiện kĩ thuật tìm kiếm thông thường, kể cả duyệt vét cạn.

Tìm kiếm vét cạn là kĩ thuật duyệt trên toàn bộ miền tìm kiếm để giải quyết các yêu cầu đa dạng của các bài toán tìm kiếm. Tuy nhiên, có nhiều bài toán tìm kiếm dùng duyệt vét cạn cũng không hiệu quả.

1. Tìm kiếm tuần tự trên một dãy n phần tử có phải là duyệt vét cạn hay không?
2. Một mảng hai chiều kích thước m × n thì duyệt vét cạn sẽ phải duyệt qua tổng số bao nhiêu phần tử?
3. Theo em, thuật toán tìm kiếm nhị phân có sử dụng duyệt vét cạn hay không?

LUYỆN TẬP

1. Viết chương trình cho phép người dùng nhập một số nguyên dương N từ bàn phím rồi in ra số có nhiều ước số nhất trong các số nhỏ hơn N.
2. Với bài toán trong Hoạt động 1, em hãy viết thêm các lệnh để tìm ra 3 học sinh có tổng điểm lớn nhất.

VẬN DỤNG

1. Cho trước dãy n số nguyên. Viết chương trình đếm và liệt kê tất cả các bộ 3 phần tử liền nhau của dãy thoả mãn điều kiện ba số này là 3 số nguyên liên tiếp (có thể tăng dần hoặc giảm dần).
2. Viết chương trình cho phép người dùng nhập một số nguyên dương N từ bàn phím, sau đó in ra toàn bộ các số hoàn hảo nhỏ hơn N. Số hoàn hảo là số có giá trị bằng tổng số các ước số của nó, không kể chính nó.

BÀI 12 THỰC HÀNH KĨ THUẬT DUYỆT CHO BÀI TOÁN TÌM KIẾM

Sau bài này em sẽ:

- Áp dụng được kĩ thuật duyệt để giải quyết một vài bài toán tìm kiếm cụ thể

Có lẽ em đã biết đến tính năng liệt kê danh sách bạn chung giữa những người dùng khác nhau để đề xuất kết bạn của Facebook. Nếu cần lập danh sách bạn chung của em và của một bạn khác trên Facebook thì em sẽ làm như thế nào?

NHIỆM VỤ 1 Cho tên các tài khoản Facebook và danh sách bạn bè của tài khoản Facebook đó được lưu trong tệp. Mỗi dòng bao gồm tên tài khoản Facebook được ngăn cách với tên bạn bè bằng dấu hai chấm, tên bạn bè được ngăn cách nhau bởi dấu phẩy như sau:

ban_fb.inp

```
Nam: Sơn, Hương, Quang, Dũng, Tuấn, Tùng
Sơn: Mai, Oanh, Quang, Dũng, Vân, Tùng
Hương: Phương, Hương, Hà, Dũng, Huyền, Tùng
Huyền: Ngọc, Hương, Quang, Lâm, Đức, Tùng
Hà: Trung, Hương, Phương, Dũng, Hải, Tùng
Hùng: Thành, Hương, Linh, Dũng, Tuấn, Nam
```

Viết chương trình đọc dữ liệu danh sách bạn bè từ tệp, sau đó cho phép người dùng nhập tên hai tài khoản Facebook từ bàn phím và in ra số bạn chung của hai tài khoản đó.

Hướng dẫn:

Phân tích: Ở bài toán này để tìm ra được số bạn chung của hai tài khoản Facebook, chúng ta cần thực hiện hai bước sau:

Bước 1: Xác định danh sách bạn bè của hai tài khoản Facebook đó. Việc này được thực hiện bằng cách sử dụng thuật toán tìm kiếm để tìm ra số thứ tự của hai tài khoản trong danh sách tên Facebook, từ đó suy ra được danh sách bạn bè của hai tài khoản.

Bước 2: Sau khi đã xác định được danh sách bạn bè của hai tài khoản, chúng ta cần xác định số phần tử chung của hai danh sách đó. Việc này có thể thực hiện bằng cách duyệt từng phần tử trong một danh sách xem phần tử này có xuất hiện trong danh sách còn lại hay không? Nếu có xuất hiện, tăng số bạn chung lên một, nếu không xuất hiện, tiếp tục duyệt đến phần tử tiếp theo trong danh sách cho đến hết.

Bài toán tìm số bạn chung có thể được thực hiện như sau:

banchung.py

```
def banchung(A,B): # tính số phần tử chung có trong 2 danh sách
    kq = 0
    for i in range(len(A)):
        if A[i] in B:
            kq= kq + 1
    return kq

input_file = open("ban_fb.inp",encoding = "utf-8")
ten_fb=[]
ban_fb=[]
for line in input_file.readlines():
    ten, ban=line.split(":")
    ten_fb.append(ten)
    ban = ban.strip().split(",")
    for i in range(len(ban)):
        ban[i] = ban[i].strip()
    ban_fb.append(ban)
input_string = input("Nhập tên 2 tài khoản Facebook cần kiểm tra:")
ten1,ten2 = input_string.split()
index1= index2 = -1
for i in range(0,len(ten_fb)): # tìm vị trí của tài khoản thứ nhất trong danh sách
    if ten1==ten_fb[i]:
        index1 = i
for i in range(0,len(ten_fb)): # tìm vị trí của tài khoản thứ hai trong danh sách
    if ten2==ten_fb[i]:
        index2 = i
if (index1==-1) or (index2==-1):
```

```
27      print("Không tồn tại tài khoản cần tìm trong danh sách")
28 else:
29      print("Số bạn chung của hai tài khoản Facebook là:",
banchung(ban_fb[index1],ban_fb[index2]))
```

NHIỆM VỤ 2 Cho trước tên các tài khoản Facebook và danh sách nhóm mà tài khoản đó tham gia, được lưu trong tệp. Mỗi dòng bao gồm tên tài khoản Facebook được ngăn cách với tên các nhóm bằng dấu hai chấm, tên các nhóm được ngăn cách nhau bởi dấu phẩy như sau:

group_fb.inp

```
Nam: học tập, phim ảnh, đá bóng, kpop, cpop, thời trang
Sơn: học tập, chụp ảnh, đá bóng, du lịch, cpop, game
Hương: thể dục, phim ảnh, tiếng Anh, kpop, thư giãn, thời
trang
Huyền: học tập, tiếng Trung, cpop, thư giãn, du lịch
Hà: thể dục, game, tiếng Pháp, kpop, piano, trà sữa
Hùng: phượt, phim ảnh, tiếng anh
```

Viết chương trình gợi ý kết bạn cho các tài khoản Facebook để gợi ý các bạn có cùng sở thích với tài khoản này. Hai tài khoản được coi là có chung sở thích nếu tham gia ít nhất 3 nhóm chung. In ra màn hình danh sách kết bạn đề xuất cho từng tài khoản.

Hướng dẫn:

Phân tích: Để thực hiện đề xuất kết bạn cho tất cả các tài khoản Facebook chúng ta cần duyệt toàn bộ các tài khoản Facebook, với mỗi tài khoản, kiểm tra số nhóm chung với tất cả các tài khoản Facebook còn lại, nếu có trên 3 nhóm chung thì đưa vào danh sách đề xuất kết bạn.

Bài toán đề xuất kết bạn có thể được thực hiện như sau:

ketban.py

```
1 def nhomchung(A,B): # Tính số phần tử chung có trong danh hai
danh sách
2     kq = 0
3     for i in range(len(A)):
4         if A[i] in B:
5             kq= kq + 1
6     return kq
7
```

```
input_file = open("group_fb.inp", encoding = "utf-8")
ten_fb=[]
group_fb=[]
for line in input_file.readlines():
    ten, group=line.split(":")
    ten_fb.append(ten)
    group = group.strip().split(",")
    for i in range(len(group)):
        group[i] = group[i].strip()
    group_fb.append(group)
print("Danh sách đề xuất kết bạn cho các tài khoản như sau:")
for i in range(len(ten_fb)):
    dexuat=[]
    for j in range(len(ten_fb)):
        if i!=j: # không đề xuất kết bạn với chính mình
            if nhomchung(group_fb[i],group_fb[j]) >= 3:
                dexuat.append(ten_fb[j])
    print("Đề xuất kết bạn cho",ten_fb[i],"là:",dexuat)
```


LUYỆN TẬP

Sửa chương trình trong Nhiệm vụ 1 để in ra được danh sách bạn chung của hai tài khoản.

VẬN DỤNG

1. Trong bài toán đề xuất danh sách kết bạn, không tìm thấy đối tượng phù hợp cho hai bạn Hà, Hùng. Hãy viết chương trình tìm nhóm gợi ý cho một bạn bất kì (được nhập vào từ bàn phím). Nhóm được gợi ý là nhóm các bạn này chưa tham gia và đang có nhiều thành viên nhất.
2. Viết chương trình để tìm ra hai bạn có nhiều bạn chung nhất, hai bạn có nhiều nhóm chung nhất.

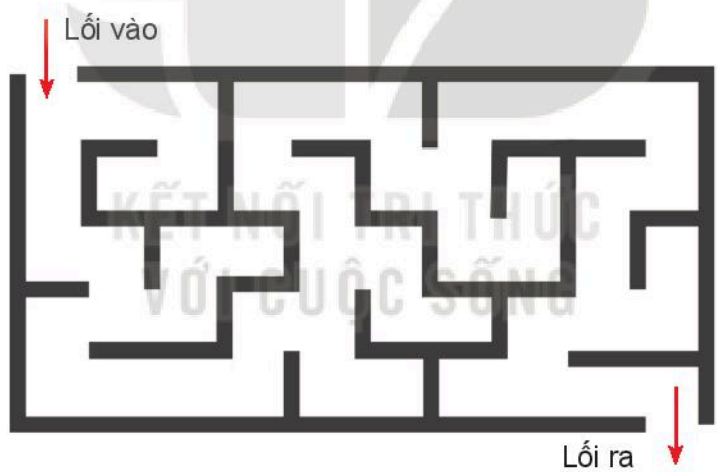
BÀI 13 KĨ THUẬT DUYỆT QUAY LUI

Sau bài này em sẽ:

- Biết và trình bày được ý tưởng của kĩ thuật quay lui thông qua một số ví dụ đơn giản.
- Nhận ra được mối liên quan giữa thiết kế thuật toán theo kĩ thuật quay lui và kĩ thuật đệ quy.

Chúng ta đã biết từ bài học trước, thiết lập các thuật toán duyệt sẽ phụ thuộc hoàn toàn vào mô hình và cấu trúc của miền dữ liệu cần tìm kiếm. Từ lâu các nhà khoa học đã nhìn thấy rất nhiều bài toán khó không tìm được cách duyệt hữu hiệu, điển hình nhất là bài toán tìm đường đi trong mê cung.

Bài toán tìm đường đi trong mê cung lần đầu tiên được đưa ra trong cuốn sách Récréations Mathématiques của tác giả Édouard Lucas năm 1882 tại Pháp. Cũng trong cuốn sách đó Lucas đã đưa ra phác thảo đầu tiên của một phương pháp giải bài toán tìm đường đi trong mê cung mà bây giờ chúng ta gọi là thuật toán duyệt quay lui, hay đơn giản là thuật toán quay lui (backtracking).



Hình 13.1. *Mê cung*

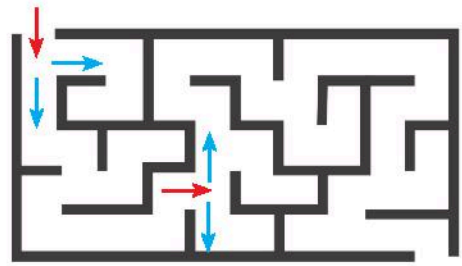
Trong trò chơi mê cung (xem hình) em cần tìm một đường đi xuất phát từ lối vào và ra khỏi mê cung tại lối ra. Em có đề xuất gì để giải bài toán này.

1. Kĩ thuật duyệt quay lui

Hoạt động 1 **Tìm hiểu ý tưởng của thuật toán quay lui**

Đọc, trao đổi và thảo luận về ý tưởng thuật toán quay lui của bài toán tìm đường đi trong mê cung.

Một đặc thù của bài toán mê cung là tại một vị trí chúng ta có thể đi tiếp theo nhiều hướng. Trong Hình 13.2 chỉ ra sau mũi tên màu đỏ có thể có hai hướng đi tiếp theo (theo mũi tên màu xanh).



Hình 13.2. Cách đi trong mê cung

Ý tưởng của thuật toán quay lui là thiết lập một hàm (thủ tục) xuất phát từ một vị trí hiện thời để tìm kiếm các bước đi tiếp theo. Khuôn dạng của hàm này được mô tả bằng mã giả (pseudocode) như sau:

```
1Tìm kiếm bước đi tiếp theo //<hàm>
2    Nếu vị trí hiện thời là đích
3        Thông báo tìm thấy nghiệm
4        Dừng chương trình
5    Lặp trên tập hợp các khả năng có thể đi tiếp
6        Nếu bước đi là khả thi
7            Thực hiện bước đi cụ thể từ vị trí hiện thời.
8            Cập nhật thông tin vào bước đi
9            Tìm kiếm bước đi tiếp theo
10           Xoá thông tin bước đi và quay lại trạng thái trước
```

Giải thích. Hàm có chức năng tìm bước đi tiếp theo xuất phát từ <vị trí> hiện thời. Nếu vị trí hiện thời là đích thì thông báo tìm thấy nghiệm tại dòng 3 và dừng chương trình. Dòng 5 sẽ tìm tất cả các phương án có thể đi tiếp. Nếu tìm thấy một phương án đi khả thi tại dòng 6 thì thực hiện ngay bước đi này tại dòng 7 để cập nhật thông tin vào <vị trí> mới. Sau gọi đệ quy lại hàm gốc để đi tiếp tại dòng 9. Nếu không thể đi tiếp (lệnh gọi đệ quy dòng 9 không thể thực hiện) thì quay lui tại dòng 10, xoá thông tin bước vừa đi để quay lại vòng lặp 5.

Ý tưởng của thuật toán duyệt quay lui là luôn tìm cách đi tiếp theo. Xuất phát từ vị trí gốc, thuật toán sẽ gọi hàm tìm bước đi tiếp theo. Nếu thực hiện được một bước đi thì gọi lại hàm để tìm bước đi tiếp theo. Nếu không tìm thấy đường đi thì cần "quay lui" về vị trí trước đó để tìm đường đi khác. Thuật toán sẽ sử dụng kĩ thuật đệ quy khi gọi hàm cho bước đi tiếp theo.

1. Khi đã thực hiện hết các bước lặp tại dòng 2 ở trên thì hàm có dừng không?
2. Lệnh gọi hàm chính của chương trình trên là gì?
3. Nếu yêu cầu bổ sung thêm 1 lệnh "Nếu thấy <lỗi ra> thì thông báo và dừng chương trình" thì lệnh này sẽ đặt ở đâu trong chương trình trên?

2. Mô hình tổng quát của thuật toán duyệt quay lui

Hoạt động 2 **Tìm hiểu mô hình tổng quát của kĩ thuật duyệt quay lui**

Quan sát, thực hiện và thảo luận các bước thiết kế mô hình tổng quát của kĩ thuật duyệt quay lui.

Mô hình tổng quát của thuật toán duyệt quay lui cần được thiết kế sao cho có thể dễ dàng cài đặt trên máy tính theo ngôn ngữ lập trình bậc cao bất kì, ví dụ như Python.

Chúng ta sẽ xét tập hợp các dãy phần tử có dạng sau:

$(x_1, x_2, ..., x_n) \in D_1 \times D_2 \times ... \times D_n$ (1)

Trong đó $D_1, D_2, ..., D_n$ là các tập hợp thuộc miền tìm kiếm của bài toán.

Các dãy (1) sẽ luôn được phép mở rộng ra để tìm nghiệm của bài toán gốc. Chú ý số lượng các tập hợp D_k và bản thân mỗi tập hợp này có thể rất lớn, do đó việc tìm kiếm vét cạn trên toàn bộ các dãy (1) là không khả thi. Do vậy cần thiết lập thuật toán duyệt quay lui để giảm thiểu tối đa việc tìm kiếm trên các dãy (1).

Chúng ta sẽ định nghĩa tập hợp $S_k \subset D_k$ theo yêu cầu sau:

$S_k = \{x \in D_k$, dãy $(x_1, x_2, ..., x_{k-1})$ có thể mở rộng thành $(x_1, x_2, ..., x_k)\}$ (2)

Ý nghĩa của việc đưa S_k vào để giảm bớt tối đa khả năng tìm kiếm x_k. Như vậy, với mỗi k, tập hợp S_k sẽ phụ thuộc vào dãy các phần tử đã tìm được trước đó $(x_1, x_2, ..., x_{k-1})$, S_k do đó sẽ nhỏ đi đáng kể so với tập hợp gốc D_k.

Bài toán duyệt quay lui có 2 phương án: (i) tìm tất cả các nghiệm và (ii) chỉ cần tìm đúng một nghiệm. Sau đây là thiết kế hàm đệ quy chính của thuật toán duyệt quay lui trên mô hình (1) và (2) và thực hiện bài toán tìm tất cả các nghiệm.

Mô hình tổng quát duyệt quay lui sử dụng đệ quy như sau.

```
1 def Backtracking([x1,x2,...,xk-1],k):
2     if (x1,x2,…,xk-1) là nghiệm:
3         Lưu và thông báo nghiệm
4     Tính toán Sk theo (x1,x2…,xk-1)
5     for xk in Sk:  # Duyệt theo thứ tự của Sk
6         Backtracking([x1,x2,…,xk],k + 1)
```

Lệnh gọi hàm gốc như sau:

```
Backtracking([], 0)
```

Giải thích.

– Vị trí xuất phát từ đầu là dãy rỗng và k = 0.

– Nếu trạng thái hiện thời $(x_1, x_2, ..., x_{k-1})$ là nghiệm của bài toán thì thông báo ngay.

– Tính toán lại S_k dựa trên trạng thái hiện thời $(x_1, x_2, ..., x_{k-1})$ tại dòng 4.

– Vòng lặp tại dòng 5 sẽ duyệt trên S_k để tìm phần tử đưa vào dãy và gọi tiếp hàm đệ quy cho bước tiếp theo tại dòng 6.

Lưu ý: Vòng lặp for tại dòng 5 sẽ thực hiện duyệt theo thứ tự các phần tử đang có của S_k. Như vậy, yêu cầu các tập hợp D_k phải được sắp thứ tự trước.

Mô hình thuật toán quay lui tổng quát quy định việc tìm trên các dãy số nguyên $(x_1, x_2, ..., x_k)$, sử dụng lệnh gọi đệ quy để mô tả bước đi tiếp theo với k + 1, nếu không tìm được bước đi tiếp theo thì quay lui để tìm hướng đi khác.

1. Trạng thái "quay lui" của thuật toán trên nằm ở đâu?
2. Có cách nào đếm được tất cả các nghiệm từ thuật toán trên được không? Nếu có thì làm cách nào?

3. Bài toán sinh xâu nhị phân

Hoạt động 3 **Thiết kế chương trình sinh tất cả các dãy nhị phân**

Cùng thực hiện, trao đổi, thảo luận thiết kế chương trình sinh tất cả các dãy nhị phân độ dài n bằng kĩ thuật quay lui.

Chúng ta sẽ áp dụng thuật toán duyệt quay lui đã được mô tả trong Hoạt động 2 để giải bài toán tìm tất cả các dãy nhị phân độ dài n. Nghiệm của bài toán sẽ được lưu trong một dãy dạng A = A[0], A[1], ..., A[n − 1]. Chúng ta sẽ thiết lập trước dãy A bao gồm n số 0 và tiến hành tìm và gán từng phần tử vào A. Miền xác định ở đây là các tập hợp D = {0, 1}. Vì phần tử A[k] không phụ thuộc vào các phần tử phía trước nên ta luôn có S_k = [0, 1]. Thuật toán được mô tả bằng hàm đệ quy như sau:

Chương trình 1.

```
1  def genBinary(A,k):
2      if k == n:
3          print(A)
4      else:
5          for i in range(2):
6              A[k] = i
7              genBinary(A,k + 1)
```

Lưu ý: Lệnh nạp dữ liệu mới sau mỗi bước đi tiếp là dòng 6. Vì dãy A đã được thiết lập từ trước có đủ n phần tử nên tại bước này chỉ là lệnh gán giá trị A[k] = i. Muốn chạy chương trình gốc, thiết lập n và dãy A gồm n số 0, sau đó gọi hàm chính như sau:

```
1 n = 4
2 A = [0]*n
3 genBinary(A,0)
```

Chúng ta thiết lập thêm một phương án nữa cho thuật toán trên, khi dãy gốc ban đầu đặt là rỗng A = []. Chú ý các thay đổi quan trọng so với chương trình trên tại dòng 6 và 8 của chương trình 2 dưới đây. Vì A sẽ được bổ sung dần, nên dòng 6 sẽ phải dùng phương thức append(). Sau khi kết thúc lệnh gọi đệ quy tại dòng 7, khi quay lui cần xoá phần tử đã nhập từ bước trước nên cần lệnh pop() tại dòng 8.

Chương trình 2.

```
1  def genBinary(A,k):
2      if k == n:
3          print(A)
4      else:
5          for i in range(2):
6              A.append(i)
7              genBinary(A,k + 1)
8              A.pop()
```

Lời gọi hàm ở Chương trình chính sẽ như sau:

```
1 n = 4
2 genBinary([],0)
```

Bài toán sinh tất cả các xâu nhị phân có thể giải quyết bằng mô hình của thuật toán quay lui tổng quát.

1. Trong chương trình 1, động tác "quay lui" nằm ở đâu?
2. Giải thích ý nghĩa của lệnh A.pop() tại dòng 8 của chương trình 2. Vì sao lệnh này không có trong chương trình 1?

LUYỆN TẬP

1. Sửa các chương trình trên bổ sung thêm chức năng: sau khi in ra tất cả các xâu nhị phân thì thông báo tổng số nghiệm.
2. Viết chương trình sinh tất cả các xâu (hoặc dãy) bao gồm n kí tự dạng "R","G" và "B".

VẬN DỤNG

1. Viết chương trình sinh tất cả các số hex (hệ đếm 16) có 3 chỉ số.
2. Viết chương trình sinh xâu nhị phân thực sự có độ dài n, tức là kết quả in ra phải là các xâu kí tự chứ không phải là danh sách (list) như trong các chương trình trên.

BÀI 14 THỰC HÀNH KĨ THUẬT DUYỆT QUAY LUI

Sau bài này em sẽ:

- Biết cách sử dụng kĩ thuật quay lui giải quyết một số bài toán phù hợp với kĩ thuật này và cài đặt thuật toán.

Theo em kĩ thuật duyệt quay lui thường được áp dụng cho những loại bài toán nào? Em có thể nêu ra một vài ví dụ không?

NHIỆM VỤ 1 Phân tử ADN gồm chuỗi các nucleotit thuộc bốn dạng A, T, G, và X. Viết chương trình in ra tất cả các dạng mạch đơn của một đoạn phân tử ADN với chiều dài gồm n các nucleotit, trong đó n được người dùng nhập từ bàn phím. Lưu ý do sự bùng nổ của tổ hợp, chỉ nên kiểm thử chương trình với số n nhỏ hơn 10.

Hướng dẫn:

Phân tích:

Bài toán là dạng tổng quát của bài toán sinh các dãy nhị phân mà chúng ta đã biết. Điểm khác biệt là cần sinh tổ hợp của 4 phần tử "A", "T", "G", "X". Chúng ta định nghĩa dictDNA là xâu nhị phân "ATGX" lưu các kí tự từ điển DNA.

Chúng ta sẽ thiết lập hàm sinh các dãy nucleotit này theo kĩ thuật quay lui, kết quả lưu trong dãy A. Dãy A ban đầu được khởi tạo bao gồm n kí tự rỗng. Hàm sinh dãy nucleotit là genDNASection(A,k) có tính năng sinh các phần tử tại vị trí thứ k.

Chương trình 1.

```
1   # Hàm đệ quy sinh các chuỗi ADN
2   def genDNASection(A,k):
3       if k == n:
4           print(A)
5       else:
6           for i in range(4):
7               A[k] = dictDNA[i]
8               genDNASection(A,k+1)
9
10  # Chương trình chính
11  n = int(input("Hãy nhập độ dài đoạn ADN:"))
12  dictDNA = "ATGX"
13  A = [' ']*n
14  genDNASection(A,0)
```

Giải thích:

– Tại dòng 13, khởi tạo mảng biểu diễn chuỗi ADN gồm n kí tự rỗng, rồi gọi hàm đệ quy ở dòng 14.

– Trong hàm genDNASection, tham số k thể hiện vị trí nucleotit đã được thiết lập. Nếu k bằng chiều dài n thì đã hoàn thành 1 nghiệm bài toán (là chuỗi AND gồm n nucleotit) nên chương trình sẽ in ra ở dòng 4. Nếu k < n thì lần lượt gán phần tử thứ k cùng A với 4 loại nucleotit (ATGX) và gọi đệ quy hàm genDNASection để sinh phần tử tiếp theo.

NHIỆM VỤ 2 Một câu trong các ngôn ngữ tự nhiên được xây dựng bằng cách sắp xếp các từ vựng lại với nhau. Xét trường hợp đơn giản: cho một tập hợp từ vựng, hãy viết chương trình sinh ra các câu văn gồm tất cả các từ đó, mỗi từ chỉ xuất hiện một lần.

Hướng dẫn:

Phân tích: Chúng ta có thể giải quyết bài toán sinh ra các câu văn ở trên bằng cách bắt đầu với một câu gồm tất cả các từ đã cho trong từ điển, sau đó thực hiện duyệt quay lui để hoán vị tất cả các vị trí của các từ.

Chương trình 2.

```
1   # Định nghĩa hàm đệ quy duyệt quay lui sinh ra các câu
2   def genSentence(words, dictionary, k):
3       if k == len(dictionary):
4           sentence = ' '.join(words)  # Sinh câu từ danh sách từ vựng
5           print(sentence)
6       else:
7           for w in dictionary:
8               if not w in words:
9                   words.append(w)
10                  genSentence(words, dictionary, k + 1)
11                  words.pop()
```

Chương trình chính sẽ chạy như sau:

```
1   dictionary = ["tôi", "quý", "bạn"]
2   words = []
3   genSentence(words, dictionary, 0)
```


LUYỆN TẬP

1. Sửa lại chương trình trong Nhiệm vụ 1 với yêu cầu thay đổi là cần in ra kết quả là các xâu kí tự chỉ bao gồm các kí tự "A", "T", "G", "X".
2. Trong Nhiệm vụ 2, động tác "quay lui" nằm ở đâu? Việc hoán vị được thực hiện như thế nào?

VẬN DỤNG

1. Viết chương trình sử dụng kĩ thuật duyệt quay lui để kiểm tra xem một biểu thức có hợp lệ về sử dụng các dấu ngoặc đơn hay không.
2. Viết chương trình in ra tất cả các hoán vị của tập hợp S = {1, 2, ..., n} với n được nhập từ bàn phím.
3. Cho các hệ số a_k, a_{k-1},..., a_1, a_0, hãy viết chương trình sinh tất cả các đa thức bậc k có thể thành lập từ các hệ số trên, mỗi hệ số sử dụng một lần. Một ví dụ của đa thức trên là $a_kx^k + a_{k-1}x^{k-1} + a_1x + a_0$.

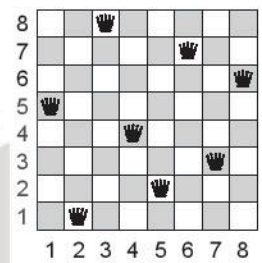
BÀI 15 BÀI TOÁN XẾP HẬU

Sau bài này em sẽ:

- Biết và thực hiện được chương trình giải bài toán xếp Hậu trên bàn cờ.

Trên bàn cờ vua chúng ta đều biết Hậu là quân cờ mạnh nhất vì nó có thể di chuyển theo tất cả các hướng ngang, dọc và chéo. Một bài toán vui rất nổi tiếng là tìm cách sắp xếp 8 quân Hậu trên bàn cờ sao cho không quân Hậu nào khống chế con nào. Em hãy thử tìm một cách xếp quân Hậu khác với cách xếp như hình sau:

Bài toán tìm tất cả các cách xếp 8 quân Hậu trên bàn cờ vua sao cho các quân Hậu không khống chế lẫn nhau được gọi là bài toán xếp Hậu (n-Queen Problem). Bài toán này được nhà bác học Đức Carl Friedrich Gauss nghiên cứu từ những năm 1850. Bài toán đã được mở rộng trên bàn cờ kích thước bất kì và vẫn đang được tiếp tục phát triển cho đến ngày nay.



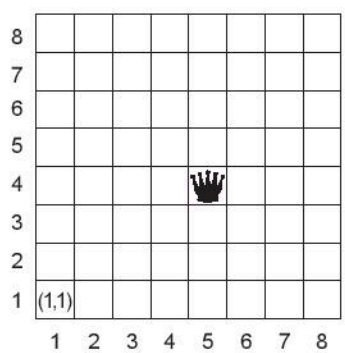
Hình 15.1. *Bàn cờ vua*

1. Mô tả bài toán xếp Hậu trên bàn cờ vua

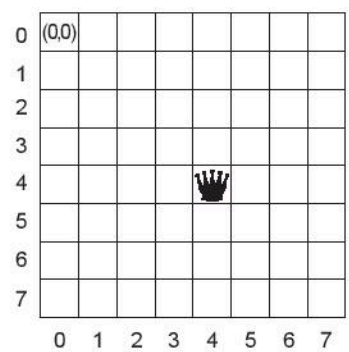
Hoạt động 1 **Tìm hiểu mô hình bài toán xếp Hậu trên bàn cờ vua tổng quát**

Đọc, quan sát, trao đổi và thảo luận về bài toán xếp Hậu tổng quát và cách tiếp cận quay lui để giải bài toán.

Chúng ta sẽ mô phỏng mô hình bài toán xếp Hậu tổng quát trên lưới ô vuông kích thước n × n. Trên thực tế các hàng và cột của bàn cờ Vua được đánh chỉ số từ 1 đến n, theo thứ tự từ dưới lên và từ trái sang phải. Nhưng chúng ta sẽ thiết lập lại việc đánh chỉ số từ 0 đến n – 1 và sẽ đánh số theo thứ tự từ trên xuống và từ trái sang phải.



Hình 15.2. *Mô hình bàn cờ Vua gốc với các chỉ số hàng, cột từ 1 đến n.*



Hình 15.3. *Mô hình bàn cờ Vua trên máy tính với các chỉ số hàng, cột từ 0 đến n − 1.*

Vì các quân Hậu không thể khống chế lẫn nhau suy ra mỗi cột trên lưới ô vuông sẽ chỉ có thể có đúng một quân Hậu. Do vậy thông tin vị trí các quân Hậu có thể được cho bởi dãy n giá trị A[0], A[1], ..., A[n – 1], trong đó A[k] là chỉ số hàng của quân Hậu tại cột thứ k.

Chúng ta sẽ đi tìm tất cả các nghiệm của bài toán bằng cách tìm tất cả các dãy:

A[0], A[1], ..., A[n – 1] (1)

Việc tìm kiếm này sẽ được tiến hành lần lượt tìm A[0], A[1], ..., khi tìm đến chỉ số k = n – 1 thì thông báo nghiệm. Nếu không tìm được tiếp thì cần "quay lui" về vị trí bên trái để tìm tiếp. Quy trình tìm kiếm nghiệm này sẽ thực hiện theo kĩ thuật quay lui đã biết. Mệnh đề sau cho chúng ta biết cách tìm phần tử A[k] nếu đã biết các phần tử đứng trước A[0], A[1], ..., A[k − 1].

Mệnh đề. Điều kiện xếp được quân Hậu tại cột thứ k.

Điều kiện để xếp được quân Hậu tại cột k là giá trị A[k] cần thoả mãn các điều kiện sau:

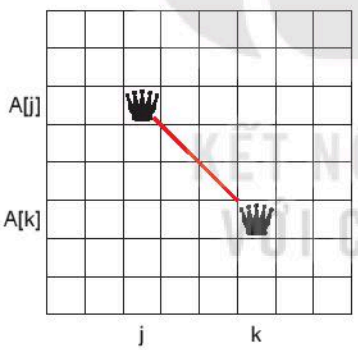
(i) $A[k] \neq A[j]$ với $\forall j < k$

(ii) $|A[k] – A[j]| \neq |k – j|$ với $\forall j < k$

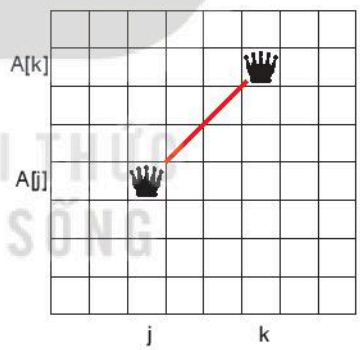
Chứng minh:

– Vì quân Hậu thứ k không thể nằm trên cùng hàng với các quân Hậu nằm trên các hàng thứ A[0], A[1], A[2], ..., A[k − 1], do đó điều kiện (i) hiển nhiên phải có.

– Bây giờ chúng ta xét điều kiện để quân Hậu thứ k không thể khống chế quân Hậu thứ j theo các đường chéo là gì. Xét hai trường hợp sau:



Hình 15.4. *Trường hợp A[j] > A[k], khi đó điều kiện để 2 quân cờ nằm trên đường chéo là A[k] – A[j] = j – k.*



Hình 15.5. *Trường hợp A[k] > A[j], khi đó điều kiện để 2 quân cờ nằm trên đường chéo là A[k] – A[j] = k – j.*

Vậy trong mọi trường hợp điều kiện để quân cờ A[k] không thể khống chế quân cờ tại vị trí A[j] là $|A[k] – A[j]| \neq |k – j|$. Mệnh đề được chứng minh.

Ví dụ nếu n = 5 và A[0] = 0 và A[1] = 2 thì vị trí tiếp theo chỉ có thể A[2] = 4.

Ý tưởng tiếp cận duyệt quay lui giải bài toán xếp Hậu tổng quát là tìm kiếm trên tất cả các dãy dạng A[0], A[1], ..., A[n – 1], trong đó A[k] là chỉ số hàng của quân Hậu tại cột thứ k.

1. Giả sử n = 4, A[0] = 2, A[1] = 0. Hãy tìm A[2].

2. Nếu n = 5, A[0] = 0, A[1] = 3. Tìm các khả năng của A[2].

2. Thiết lập lời giải bài toán xếp Hậu tổng quát

Hoạt động 2 Tìm hiểu thuật toán và cách giải bài toán xếp Hậu trên bàn cờ vua

Đọc, quan sát, trao đổi và thảo luận về thuật toán và thiết lập chương trình hoàn chỉnh giải bài toán.

Chúng ta sẽ thiết kế thuật toán duyệt quay lui cho trường hợp tổng quát. Như vậy, số tự nhiên n được cố định trước, tập các nghiệm sẽ được tìm từ các dãy độ dài n như sau:

A[0], A[1], ..., A[n – 1]

Mảng A được thiết lập trước có n phần tử và bao gồm các số 0.

Thuật toán chính là trynext(k) sẽ tìm ra vị trí đầu tiên để gán cho A[k] tại thời điểm coi như đã biết vị trí k – 1 quân Hậu trước đó tại A[0], A[1], ..., A[k – 1]. Hàm này được mô tả theo mô hình tổng quát của kĩ thuật duyệt quay lui như sau:

```
1   def trynext(k):
2       if k == n:
3           <Thông báo nghiệm>
4       else:
5           for i in range(n):
6               if <Có thể đặt Hậu tại hàng i cột k>:
7                   A[k] = i
8                   trynext(k + 1)
```


Giải thích.

– Dòng 3 sẽ thông báo nghiệm nếu k = n, tức là đã tìm được đủ n vị trí A[0], A[1], ..., A[n – 1]. Hàm showQueen(A) sẽ thể hiện trên màn hình phương án sắp xếp các quân Hậu theo bộ dữ liệu của A. Chú ý các hàng sẽ đánh chỉ số từ trên xuống, các cột đánh chỉ số từ trái sang.

```
1   def showQueen(A):
2       n = len(A)
3       for i in range(n):
4           for j in range(n):
5               if A[j] == i:
6                   print(1, end = " ")
7               else:
8                   print(0, end = " ")
9           print()
```

– Tại dòng 6 của trynext(k) sẽ cần một hàm kiểm tra xem có thể đặt Hậu tại hàng i, cột k hay không, hay có thể gán A[k] = i được hay không. Hàm này dễ dàng thiết kế dựa trên mệnh đề đã được chứng minh trong Hoạt động 1. Hàm có tên check(A, i, j) sẽ kiểm tra và trả lại True nếu có thể gán A[k] = i và nếu ngược lại trả về False.

```
1   def check(A,i,j):
2       for k in range(j):
3           if A[k] == i:
4               return False
5           if abs(A[k] - i) == abs(j - k):
6               return False
7       return True
```

Cuối cùng nếu tất cả các hàm trên đã được thiết kế đầy đủ thì phần chương trình chính sẽ gồm các lệnh sau. Chương trình sẽ in ra trên màn hình tất cả các cách sắp xếp n quân Hậu trên lưới.

```
1   n = 8
2   A = [0]*n
3   trynext(0)
```

Chương trình đầy đủ sau sẽ bổ sung thêm chức năng đếm số các phương án xếp quân hậu và in ra tất cả các phương án này.

Chương trình hoàn chỉnh đếm và in ra tất cả các nghiệm của bài toán xếp Hậu tổng quát có thể như sau:

```
1   def showQueen(A):
2       n = len(A)
3       for i in range(n):
4           for j in range(n):
5               if A[j] == i:
6                   print(1, end = " ")
7               else:
8                   print(0, end = " ")
9           print()
10
11  def check(A,i,j):
12      for k in range(j):
13          if A[k] == i:
14              return False
15          if abs(A[k] - i) == abs(j - k):
16              return False
17      return True
```

```
18
19  def trynext(k):
20      global ncount
21      if k == n:
22          ncount = ncount + 1
23          print("Phương án",ncount,":")
24          showQueen(A)
25      else:
26          for i in range(n):
27              if check(A,i,k):
28                  A[k] = i
29                  trynext(k + 1)
```

Ví dụ muốn đếm và in ra tất cả các cách xếp quân Hậu cho bàn cờ 8 × 8 thì thực hiện theo các lệnh sau:

```
1   n = 8
2   A = [0]*n
3   ncount = 0
4   trynext(0)
```

Lời giải bài toán tìm tất cả các phương án xếp quân Hậu trên bàn cờ Vua tổng quát n × n có thể thực hiện bằng kĩ thuật duyệt quay lui khá đơn giản. Chương trình sẽ in ra tất cả các phương án nghiệm.

1. Với n = 3 bài toán xếp Hậu có nghiệm không?
2. Vì sao chương trình trên cần khai báo biến ncount với từ khoá **global** bên trong hàm trynext().

LUYỆN TẬP

1. Hãy tìm bằng tay (không cần máy tính) cả hai phương án của bài toán xếp Hậu với n = 4.
2. Nếu chúng ta mô phỏng lưới ô vuông đánh chỉ số các hàng từ dưới lên thì chương trình trên còn đúng không? Nếu phải thay đổi thì cần sửa chỗ nào?

VẬN DỤNG

1. Gọi Q(n) là số các cách xếp n quân Hậu lên bàn cờ kích thước n × n sao cho các quân Hậu không khống chế nhau. Sử dụng thuật toán đã được học, em hãy viết chương trình tính các giá trị Q(n) với n = 4, 5, 6, 7, 8, 9, 10.
2. Tính Q(n) với n = 11, 12, 13.

BÀI 16 THỰC HÀNH THIẾT KẾ THUẬT TOÁN THEO KĨ THUẬT DUYỆT QUAY LUI

Sau bài này em sẽ:

- Ôn lại kĩ thuật duyệt quay lui để thiết kế thuật toán cho các bài toán tìm kiếm.
- Thực hành kĩ thuật duyệt quay lui với một số bài toán phức tạp hơn

Chắc em đã nghe nói nhiều bài toán tìm đường đi trong mê cung. Nếu áp dụng kĩ thuật duyệt quay lui cho bài toán này thì làm thế nào để tìm ra các bước đi tiếp theo từ một vị trí?

NHIỆM VỤ *Giải bài toán mê cung.* Mê cung là một hình chữ nhật gồm m hàng, n cột ô được định nghĩa bằng một tệp dạng văn bản có tên maze.inp, trong đó các ô 0 là ô được đi, ô 1 tương ứng với tường và không được đi. Tại một vị trí có thể chọn một trong bốn hướng đi tiếp (lên, xuống, trái, phải). Giả sử vị trí xuất phát là ô bên trái phía trên, hãy tìm một đường đi đến ô bên phải phía dưới để thoát khỏi mê cung. Nếu tìm ra nghiệm thì in ra đường đi dưới dạng bảng trong đó vết của đường đi gồm các ô được thể hiện bằng số 1, các ô còn lại là số 0. Kết quả được thể hiện trên màn hình và ghi ra tệp maze.out. Ví dụ một thể hiện của dữ liệu đầu vào và ra như bảng bên.

maze.inp
0 1 1 0 1
0 0 1 1 1
1 0 0 0 1
0 0 1 0 0

maze.out
1 0 0 0 0
1 1 0 0 0
0 1 1 1 0
0 0 0 1 1

Hướng dẫn:

Phân tích: Ý tưởng thuật toán như sau: Quá trình tìm đường được bắt đầu tại vị trí bên phải phía trên. Tại mỗi vị trí, thuật toán thử đi tiếp một bước theo một trong các hướng lên xuống trái phải. Nếu đi được thì cập nhật đường đi, nếu không thì thử hướng khác. Nếu không có hướng nào hợp lệ thì quay lui. Chúng ta sẽ thiết lập mảng maze để lưu dữ liệu đầu vào của mê cung. Thiết lập mảng sol để lưu vết đường đi trong mê cung. Ban đầu mảng sol sẽ gồm toàn số 0.

Hàm quay lui để tìm đường đi chính là solveMaze(maze, m, n, x, y, sol), mê cung có m hàng, n cột, x, y là toạ độ của ô hiện thời để tìm đường đi tiếp. Hàm sẽ trả lại True nếu từ vị trí hiện thời (x, y) tìm được đường đi thoát khỏi mê cung, ngược lại sẽ trả về False. Sau đây là mã giả mô tả nhanh hàm solveMaze().

```
1    def solveMaze(maze,m,n,x,y.sol):
2        if <(x,y) nằm ở vị trí đích và rỗng>:
3            đặt sol[x][y] = 1 # thiết lập bước đi và kết thúc
4            return True
5        if <ô (x,y) nằm trong mê cung và đang rỗng>:
6            if <ô (x,y) đã từng được đi qua trước đó>:
7                return False
8            Đặt sol[x][y] = 1 # đánh dấu đã đi qua ô này
9           Lần lượt thử đi tiếp sang các ô trái, phải, trên, dưới
bằng cách gọi đệ quy hàm solveMaze()
10           Nếu thành công thì
```

```
            return True
        # Tới đây quay lui vì các bước đi tiếp ở trên đều thất bại
        sol[x][y] = 0 # thiết lập trạng thái trước khi thực hiện solveMaze()
        return False
    else # ô (x,y) nằm ngoài mê cung hoặc là tường
        return False
```

Chương trình chính có dạng như sau:

```
maze = mảng được đọc từ tệp maze.inp
m, n = số hàng và số cột của maze
khởi tạo ma trận sol kính thước giống maze với các phần tử đều bằng 0 để chứa đường đi cần tìm
solveMaze(maze, m, n, 0, 0, solve) # gọi hàm solve tìm đường từ vị trí góc trái bên trên
Thông báo kết quả
```

Sau đây là toàn bộ chương trình:

Chương trình

```
# Hàm readmaze/writemaze để đọc/ghi mê cung
def readmaze():
    file = open("maze.inp")
    maze = []
    for row in file:
        h = [int(x) for x in row.split()]
        maze.append(h)
    file.close()
    return maze
def writemaze(outmaze):
    f = open("maze.out","w")
    for i in outmaze:
        for j in i:
            print(j,file = f,end =" ")
        print(file = f)
    f.close()

# Hàm tìm đệ quy tìm đường đi trong mê cung theo phương pháp quay lui
def solveMaze (maze,  m, n, x, y, sol):
    # Nếu đã đến ô bên phải phía dưới đã tìm được đường và kết thúc
    if x == n - 1 and y == m - 1 and maze[y][x] == 0:
        sol[y][x] = 1
        return True
    # Nếu ô x,y này rỗng thì tìm đường đi tiếp
    if x >= 0 and x < n and y >= 0 and y < m and maze[y][x] == 0:
        if sol[y][x] == 1:
            return False
        sol[y][x] = 1 # Tạm đánh dấu đường ở ô này
        if solveMaze(maze, m, n, x + 1, y, sol):
            return True
        if solveMaze(maze, m, n, x, y + 1, sol):
```

Đọc từng dòng từ tệp maze.inp rồi chèn vào mảng maze

Ghi từng dòng mê cung ra tệp

Thử đi tiếp theo các hướng lên xuống trái phải bằng cách gọi đệ quy hàm solveMaze

```
32              return True
33          if x > 0 and solveMaze(maze, m, n, x - 1, y, sol):
34              return True
35          if y > 0 and solveMaze(maze, m, n, x, y - 1, sol):
36              return True
37          sol[y][x] = 0 # Quay lui, thiết lập trạng thái ban
đầu               tại ô (x, y)
38          return False
39      else:
40          return False
41  # Chương trình chính
42  maze = readmaze()
43  m = len(maze)
44  n = len(maze[0])
45  print("Kích thước mê cung:",m,"hàng",n,"cột")
46  sol = [[ 0 for j in range(n) ] for i in range(m)]
47  if solveMaze(maze, m, n, 0, 0, sol) == False:
48      print("Không tồn tại đường đi")
49  else:
50      writemaze(sol)
51      print("Đường đi khỏi mê cung:")
52      for i in sol:
53          for j in i:
54              print(str(j) + " ", end ="")
55          print() # in kí tự xuống dòng
```

Giải thích:

– Hàm readmaze() dùng để đọc dữ liệu từ tệp maze.inp, kết quả thông tin mê cung đưa vào mảng 2 chiều maze.

– Hàm writemaze(outmaze) ghi mảng kết quả outmaze ra tệp maze.out.

– Nếu ô hiện tại chưa là đích, tại dòng 25 kiểm tra xem ô này có được phép đi không, nếu có thì gán 1 tại ô hiện thời (dòng 27) và thử đi tiếp một bước theo một trong các hướng (lên, xuống, trái, phải). Nếu không đi được hoặc ô hiện tại không hợp lệ, trả lại giá trị False và quay lui về bước đi trước đó. Trước khi quay lui thì gán 0 tại ô hiện thời tại dòng 37.

– Từ dòng 46, khởi tạo biến sol để lưu nghiệm và gọi hàm solveMaze để giải.

LUYỆN TẬP

1. Nếu sửa yêu cầu đề bài đặt vị trí xuất phát tại ô giữa của mê cung (ví dụ vị trí m//2, n//2), vị trí thoát của mê cung là ô trái trên hoặc phải dưới của mê cung thì cần sửa chương trình như thế nào?
2. Trên dữ liệu đầu ra của bài toán chưa thể hiện thông tin của các ô là tường. Hãy sửa lại chương trình để trên dữ liệu đầu ra các ô là tường sẽ được đánh dấu bằng "x".

VẬN DỤNG

1. Cải tiến nhiệm vụ thực hành để chương trình in ra màn hình tất cả các đường đi để thoát ra khỏi mê cung.
2. Giải bài toán xếp Hậu tổng quát m hàng n cột trong đó m và n là các số tự nhiên bất kì ($m \geq n$).
3. Bài toán "Mã đi tuần" được phát biểu như sau: cho vị trí ban đầu của quân mã trên bàn cờ vua 8×8, hãy tìm một hành trình của quân mã sao cho nó đi hết các ô bàn cờ mà không đi qua bất kì ô nào hai lần. Hãy dùng chiến lược quay lui để tìm lời giải cho bài toán này

BẢNG GIẢI THÍCH THUẬT NGỮ

	Thuật ngữ	Giải thích	Trang
C	Chia để trị	Mô hình thiết kế thuật toán để giải một bài toán bằng cách chia bài toán ban đầu thành các bài toán nhỏ hơn thuộc cùng thể loại. Quá trình phân chia này có thể lặp lại nhiều lần, cho đến khi bài toán thu được đủ đơn giản để có thể giải quyết trực tiếp. Sau đó lời giải của các bài toán nhỏ được kết hợp lại để thu được lời giải cho bài toán ban đầu.	28
	Công thức truy hồi	Công thức tính giá trị phần tử của một dãy thông qua các phần tử đứng trước nó.	7
Đ	Đệ quy	Hiện tượng, sự vật có tính chất lặp lại chính nó hoặc được định nghĩa theo chính hiện tượng, sự vật đó. Trong lập trình, đệ quy được thể hiện bằng hàm đệ quy.	5
P	Phương pháp làm mịn dần	Phương pháp thiết kế thuật toán và chương trình theo nhiều giai đoạn, đi từ tổng thể đến chi tiết, giai đoạn sau chi tiết hơn giai đoạn trước.	31
Q	Quay lui	Kĩ thuật duyệt một cách có hệ thống trên không gian tìm kiếm, cho phép quay lui để chuyển sang nhánh tìm kiếm khác với mục đích không bỏ sót lời giải. Vì thế kĩ thuật này còn được gọi là quay lui - vét cạn.	56

Nhà xuất bản Giáo dục Việt Nam xin trân trọng cảm ơn các tác giả có tác phẩm, tư liệu được sử dụng, trích dẫn trong cuốn sách này.

Chịu trách nhiệm xuất bản:
Chủ tịch Hội đồng Thành viên NGUYỄN ĐỨC THÁI
Tổng Giám đốc HOÀNG LÊ BÁCH

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

Biên tập nội dung: NGUYỄN THỊ NGUYÊN THUÝ – PHẠM THỊ THANH NAM
Biên tập mĩ thuật: NGUYỄN BÍCH LA
Thiết kế sách: PHAN THỊ THU HƯƠNG
Trình bày bìa: NGUYỄN BÍCH LA
Minh hoạ: NGUYỄN THỊ HUẾ
Sửa bản in: PHAN THỊ THANH BÌNH - PHẠM THỊ TÌNH
Chế bản: CÔNG TY CỔ PHẦN MĨ THUẬT VÀ TRUYỀN THÔNG

CHUYÊN ĐỀ HỌC TẬP TIN HỌC 11 – ĐỊNH HƯỚNG KHOA HỌC MÁY TÍNH
Mã số:
In ... bản, (QĐ ...) khổ 19 x 26,5 cm.
Đơn vị in: ...
Địa chỉ: ...
Số ĐKXB: ....-.../CXBIPH/1....GD
Số QĐXB: .../QĐ-GD – HN ngày ... tháng ... năm 20...
In xong và nộp lưu chiểu tháng ... năm 20...
Mã số ISBN: 978-604-.....-...