HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÀI GIẢNG

KỸ THUẬT LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Học phần tốt nghiệp CNPM 1

Biên soạn: TS. NGUYỄN DUY PHƯƠNG

ThS. NGUYỄN MẠNH SƠN

HÀ NỘI 2020

MỤC LỤC

LỜI NÓI ĐẦU

Môn học Kỹ thuật lập trình Hướng đối tượng là môn học Thay thế tốt nghiệp 1 dành cho sinh viên năm cuối chuyên ngành Công nghệ phần mềm. Kiến thức và kỹ năng yêu cầu cho môn học này là sự tổng hợp kiến thức của các môn học:

- Lập trình hướng đối tượng với ngôn ngữ Java
- Toán rời rạc 1 và Toán rời rạc 2
- Cấu trúc dữ liệu và giải thuật

Theo đề cương môn học, sinh viên cần ôn tập kiến thức và giải quyết được các bài tập lập trình cơ bản và lập trình thuật toán với ngôn ngữ lập trình Java. Bài giảng này sẽ giúp sinh viên hệ thống kiến thức theo từng nội dung và tiếp cận các bài tập theo thứ tự từ dễ đến khó để quá trình luyện tập kỹ năng được thuận lợi hơn.

Cấu trúc các chương trong bài giảng bao gồm:

- Chương 1: Kỹ thuật lập trình với ngôn ngữ Java
- Chương 2: Lý thuyết tổ hợp
- Chương 3: Các mô hình thuật toán cơ bản
- Chương 4: Lý thuyết đồ thị
- Chương 5: Các cấu trúc dữ liệu cơ bản.

Cuối tài liệu sẽ có phụ lục trình bày danh mục các bài tập cơ bản cho môn học. Các bài tập được trình bày bao gồm:

- Tên bài
- Mô tả đề bài
- Các ràng buộc với dữ liệu vào và kết quả
- Test ví dụ để hiểu đề

Tất cả các bài tập đều đã được đưa lên cổng thực hành trực tuyến của Khoa CNTT1. Trên cổng thực hành đã có các thảo luận và gợi ý cho từng bài. Bộ dữ liệu để chấm trên cổng thực hành đã được sinh cho phù hợp với các đặc trưng của ngôn ngữ Java và khuyến khích sinh viên sử dụng thư viện Java Collection.

Rất mong nhận được sự góp ý của quý thầy cô và các em sinh viên.

Hà Nội, tháng 12 năm 2020

CHƯƠNG 1. KỸ THUẬT LẬP TRÌNH VỚI NGÔN NGỮ JAVA

1.1. Lập trình Java cơ bản

1.1.1. Các kiểu dữ liệu cơ bản

Khai báo biến

Cú pháp khai báo biến:

dataType varName;

Trong đó, dataType là kiểu dữ liệu của biến, varName là tên biến. Trong Java, việc đặt tên biến phải tuân theo các quy tắc sau:

- Chỉ được bắt đầu bằng một kí tự (chữ), hoặc một dấu gạch dưới, hoặc một kí tự dollar.
- Không có khoảng trắng giữa tên.
- Bắt đầu từ kí tự thứ hai, có thể dùng các kí tự (chữ), chữ số, dấu dollar, dấu gạch dưới.
- Không trùng với các từ khoá.
- Có phân biệt chữ hoa chữ thường

Phạm vi hoạt động của biến

Một biến có phạm vi hoạt động trong toàn bộ khối lệnh mà nó được khai báo. Một khối lệnh bắt đầu bằng dấu "{" và kết thúc bằng dấu "}":

Nếu biến được khai báo trong một cấu trúc lệnh điều khiển, biến đó có phạm vi hoạt động trong khối lệnh tương ứng.

Nếu biến được khai báo trong một phương thức (Không nằm trong khối lệnh nào), biến đó có phạm vi hoạt động trong phương thức tương ứng: có thể được sử dụng trong tất cả các khối lệnh của phương thức.

Nếu biến được khai báo trong một lớp (Không nằm trong trong một phương thức nào), biến đó có phạm vi hoạt động trong toàn bộ lớp tương ứng: có thể được sử dụng trong tất cả các phương thức của lớp.

Các kiểu dữ liệu

Trong Java, kiểu dữ liệu được chia thành hai loại:

Các kiểu dữ liệu cơ bản

Các kiểu dữ liệu đối tượng

Kiểu dữ liệu cơ bản

Java cung cấp các kiểu dữ liệu cơ bản như sau:

byte: Dùng để lưu dữ liệu kiểu số nguyên có kích thước một byte (8 bít). Phạm vi biểu diễn giá trị từ -128 đến 127. Giá trị mặc định là 0.

char: Dùng để lưu dữ liệu kiểu kí tự hoặc số nguyên không âm có kích thước 2 byte (16 bít). Phạm vi biểu diễn giá trị từ 0 đến u\ffff. Giá trị mặc định là 0.

boolean: Dùng để lưu dữ liệu chỉ có hai trạng thái đúng hoặc sai (độ lớn chỉ có 1 bít). Phạm vi biểu diễn giá trị là {"True", "False"}. Giá trị mặc định là False.

short: Dùng để lưu dữ liệu có kiểu số nguyên, kích cỡ 2 byte (16 bít). Phạm vi biểu diễn giá trị từ - 32768 đến 32767. Giá trị mặc định là 0.

int: Dùng để lưu dữ liệu có kiểu số nguyên, kích cỡ 4 byte (32 bít). Phạm vi biểu diễn giá trị từ -2,147,483,648 đến 2,147,483,647. Giá trị mặc định là 0.

float: Dùng để lưu dữ liệu có kiểu số thực, kích cỡ 4 byte (32 bít). Giá trị mặc định là 0.0f.

double: Dùng để lưu dữ liệu có kiểu số thực có kích thước lên đến 8 byte. Giá trị mặc định là 0.00d

long: Dùng để lưu dữ liệu có kiểu số nguyên có kích thước lên đến 8 byte. Giá trị mặc định là 0l.

Ép kiểu (Type casting)

Ví dụ, nhiều khi gặp tình huống cần cộng một biến có dạng **integer** với một biến có dạng **float.** Để xử lý tình huống này, Java sử dụng tính năng ép kiểu (type casting) của C/C++. Đoạn mã sau đây thực hiện phép cộng một giá trị dấu phẩy động (float) với một giá trị nguyên (integer).

```
float c = 35.8f;
int b = (int)c + 1;
```

Đầu tiên giá trị dấu phảy động **c** được đổi thành giá trị nguyên 35. Sau đó nó được cộng với 1 và kết quả là giá trị 36 được lưu vào **b.**

Trong Java có hai loại ép kiểu dữ liệu:

Nới rộng (widening): quá trình làm tròn số từ kiểu dữ liệu có kích thước nhỏ hơn sang kiểu có kích thước lớn hơn. Kiểu biến đổi này không làm mất thông tin. Ví dụ

chuyển từ **int** sang **float**. Chuyển kiểu loại này có thể được thực hiện ngầm định bởi trình biên dịch.

Thu hẹp (narrowwing): quá trình làm tròn số từ kiểu dữ liệu có kích thước lớn hơn sang kiểu có kích thước nhỏ hơn. Kiểu biến đổi này có thể làm mất thông tin như ví dụ ở trên. Chuyển kiểu loại này không thể thực hiện ngầm định bởi trình biên dịch, người dùng phải thực hiện chuyển kiểu tường minh.

Các toán tử

Java cung cấp các dạng toán tử sau:

Toán tử số học
Toán tử bit

Toán tử quan hệ
Toán tử logic

Toán tử điều
kiện Toán tử gán

Toán tử số học

Các toán hạng của các toán tử số học phải ở dạng số. Các toán hạng kiểu boolean không sử dụng được, song các toán hạng ký tự cho phép sử dụng loại toán tử này. Một vài kiểu toán tử được liệt kê trong bảng dưới đây.

Toán tử	**Mô tả**
+	Cộng. Trả về giá trị tổng hai toán hạng
-	Trừ Trả về kết quả của phép trừ.
*	Nhân Trả về giá trị là tích hai toán hạng.
/	Chia

	Trả về giá trị là thương của phép chia
%	Phép lấy modul Giá trị trả về là phần dư của phép chia
++	Tăng dần Tăng giá trị của biến lên 1. Ví dụ a++ tương đương với a = a + 1
--	Giảm dần Giảm giá trị của biến 1 đơn vị. Ví dụ a-- tương đương với a = a - 1
+=	Cộng và gán giá trị Cộng các giá trị của toán hạng bên trái vào toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ c += a tương đương c = c + a

=	Trừ và gán giá trị Trừ các giá trị của toán hạng bên trái vào toán toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ c -= a tương đương với c = c - a
*=	Nhân và gán Nhân các giá trị của toán hạng bên trái với toán toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ c *= a tương đương với c = c*a
/=	Chia và gán Chia giá trị của toán hạng bên trái cho toán toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ c /= a tương đương với c = c/a
%=	Lấy số dư và gán Chia giá trị của toán hạng bên trái cho toán toán hạng bên phải và gán giá trị **số dư** vào toán hạng bên trái. Ví dụ c %= a tương đương với c = c%a

Bảng 1.1. Các toán tử số học

Toán tử Bit

Các toán tử dạng bit cho phép ta thao tác trên từng bit riêng biệt trong các kiểu dữ liệu nguyên thuỷ.

Toán tử	**Mô tả**
~	Phủ định bit (NOT) Trả về giá trị phủ định của một bít.
&	Toán tử AND bít Trả về giá trị là 1 nếu các toán hạng là 1 và 0 trong các trường hợp khác
|	Toán tử OR bít Trả về giá trị là 1 nếu một trong các toán hạng là 1 và 0 trong các trường hợp khác.
^	Toán tử Exclusive OR bít Trả về giá trị là 1 **nếu chỉ một** trong các toán hạng là 1 và trả về 0 trong các trường hợp khác.
>>	Dịch sang phải bít

	Chuyển toàn bộ các bít cuả một số sang phải một vị trí, giữ nguyên dấu của số âm. Toán hạng bên trái là số bị dịch còn số bên phải chỉ số vị trí mà các bít cần dịch.
<<	Dịch sang trái bít Chuyển toàn bộ các bít cuả một số sang trái một vị trí, giữ nguyên dấu cuả số âm. Toán hạng bên trái là số bị dịch còn số bên phải chỉ số vị trí mà các bít cần dịch.

Bảng 1.2. Các toán tử Bit

Các toán tử quan hệ

Các toán tử quan hệ kiểm tra mối quan hệ giữa hai toán hạng. Kết quả của một biểu thức có dùng các toán tử quan hệ là những giá trị Boolean (logic "đúng" hoặc "sai"). Các toán tử quan hệ được sử dụng trong các cấu trúc điều khiển.

Toán tử	**Mô tả**
= =	So sánh bằng Toán tử này kiểm tra sự tương đương của hai toán hạng
!=	So sánh khác Kiểm tra sự khác nhau của hai toán hạng
>	Lớn hơn Kiểm tra giá trị của toán hạng bên phải lớn hơn toán hạng bên trái hay không
<	Nhỏ hơn

	Kiểm tra giá trị của toán hạng bên phải có nhỏ hơn toán hạng bên trái hay không
>=	Lớn hơn hoặc bằng Kiểm tra giá trị của toán hạng bên phải có lớn hơn hoặc bằng toán hạng bên trái hay không
<=	Nhỏ hơn hoặc bằng Kiểm tra giá trị của toán hạng bên phải có nhỏ hơn hoặc bằng toán hạng bên trái hay không

Bảng 1.3. Các toán tử quan hệ

Các toán tử logic

Các toán tử logic làm việc với các toán hạng Boolean. Một vài toán tử kiểu này được chỉ ra dưới đây

Toán tử	**Mô tả**
&&	Và (AND) Trả về một giá trị "Đúng" (True) nếu chỉ khi cả hai toán tử có giá trị "True"
||	Hoặc (OR) Trả về giá trị "True" nếu ít nhất một giá trị là True

^	XOR Trả về giá trị True nếu và chỉ nếu chỉ một trong các giá trị là True, các trường hợp còn lại cho giá trị False (sai)
!	Toán hạng đơn tử NOT. Chuyển giá trị từ True sang False và ngược lại.

Bảng 1.4. Các toán tử logic

1.1.2. Các cấu trúc điều khiển

Cấu trúc điều kiện

Cấu trúc chung của câu lệnh điều kiện:

```
if (conditon)
{
     action1 statements;
}
else
{
    action2 statements;
}
```

Condition: Biểu thức boolean như toán tử so sánh.

action 1: Khối lệnh được thực thi khi giá trị điều kiện là True

action 2: Khối lệnh được thực thi nếu điều kiện trả về giá trị False

Đoạn chương trình sau kiểm tra xem các số có chia hết cho 5 hay không.

```
package vidu;
class CheckNumber
{
public static void main(String args[])
{

      int num = 10;
      if(num%5 == 0)
            System.out.println (num + " is divisable for 5!");
      else
            System.out.println (num + " is indivisable for
      5!");
}
}
```

đoạn chương trình trên num được gán giá trị nguyên là 10. Trong câu lệnh **if-else** điều kiện **num** %5 trả về giá trị 0 và điều kiện thực hiện là True. Thông báo "10 is divisable for 5!" được in ra. Lưu ý rằng vì chỉ có một câu lệnh được viết trong đoạn "if" và "else", bởi vậy không cần thiết phải được đưa vào dấu ngoặc móc "{" và "}".

Câu lệnh lựa chọn

Khối lệnh switch-case có thể được sử dụng thay thế câu lệnh if-else trong trường hợp một biểu thức cho ra nhiều kết quả. Cú pháp:

```
swich (expression)

{

      case 'value1': action 1 statement;

                        break;
```

```
        case 'value2': action 2 statement;

                        break;

        ……

        case 'valueN': actionN statement;

                        break;

        default: default_action statement;

}
```

expression - Biến chứa một giá trị xác định

value1,value 2,….valueN: Các giá trị hằng số phù hợp với giá trị trên biến **expression**

action1,action2…actionN: Khối lệnh được thực thi khi trường hợp tương ứng có giá trị True

break: Từ khoá được sử dụng để bỏ qua tất cả các câu lệnh sau đó và giành quyền điều khiển cho cấu trúc bên ngoài **switch**

default: Từ khóa tuỳ chọn được sử dụng để chỉ rõ các câu lệnh nào được thực hiện chỉ khi tất cả các trường hợp nhận giá trị False

default - action: Khối lệnh được thực hiện chỉ khi tất cả các trường hợp nhận giá trị False

Đoạn chương trình sau xác định giá trị trong một biến nguyên và hiển thị ngày trong tuần được thể hiện dưới dạng chuỗi. Để kiểm tra các giá trị nằm trong khoảng từ 0 đến 6, chương trình sẽ thông báo lỗi nếu nằm ngoài phạm vi trên.

```
package vidu;

class SwitchDemo
{
   public static void main(String agrs[])
   {
      int day = 2;
      switch(day)
      {
    case 0 : System.out.println("Sunday");
                 break;

     case 1 : System.out.println("Monday");
                 break;

    case 2 : System.out.println("Tuesday");
                 break;

     case 3 : System.out.println("Wednesday");
                   break;

    case 4 : System.out.println("Thursday");
                 break;

     case 5: System.out.println("Friday");
                 break;

  case 6 : System.out.println("Satuday");
```

```
                break;

    default:
          System.out.println("Invalid day of week");
     }
 }

}
```

Nếu giá trị của biến **day** là 2, chương trình sẽ hiển thị **Tuesday**, và cứ tiếp như vậy .

Các vòng lặp

Vòng lặp While

Vòng lặp **while** thực thi khối lệnh khi điều kiện thực thi vẫn là True và dừng lại khi điều kiện thực thi nhận giá trị False. Cú pháp:

```
while(condition)
{
      action statements;


}
```

condition: có giá trị bool; vòng lặp sẽ tiếp tục cho nếu điều kiện vẫn có giá trị True.

action statement: Khối lệnh được thực hiện nếu **condition** nhận giá trị True Đoạn chương trình sau tính tổng của 5 số tự nhiên đầu tiên dùng cấu trúc while.

```
package vidu;
class WhileDemo
{
   public static void main(String args[])
   {
      int a = 5, sum = 1;
      while (a >= 1)
```

```
        {
           sum +=a;
           a--;
        }
        System.out.println("The sum is " + sum);
     }
  }
```

ví dụ trên, vòng lặp được thực thi cho đến khi điều kiện a>=1 là **True**. Biến **a** được khai báo bên ngoài vòng lặp và được gán giá trị là 5. Cuối mỗi vòng lặp, giá tri của **a** giảm đi 1. Sau năm vòng giá trị của **a** bằng 0. Điều kiện trả về giá trị False và vòng lặp kết thúc. Kết quả sẽ được hiển thị **" The sum is 15"**

Vòng lặp do-while

Vòng lặp do-while thực thi khối lệnh khi mà điều kiện là True, tương tự như vòng lặp while, ngoại trừ do-while thực hiện lệnh ít nhất một lần ngay cả khi điều kiện là False. Cú pháp:

```
do{
        action statements;
}while(condition);
```

condition: Biểu thức bool; vòng lặp sẽ tiếp tục khi mà điều kiện vẫn có giá trị True.

action statement: Khối lệnh luôn được thực hiện ở lần thứ nhất, từ vòng lặp thứ hai, chúng được thực hiện khi **condition** nhận giá trị True.

Ví dụ sau tính tổng của 5 số tự nhiên đầu tiên dùng cấu trúc do-while.

```
package vidu;
class DoWhileDemo
{
   public static void main(String args[])
   {
      int a = 1, sum = 0;
```

```
        do{
           sum += a;
           a++;
        }while (a <= 5);
        System.out.println("Sum of 1 to 5 is " + sum);
    }
}
```

Biến **a** được khởi tạo với giá trị 1, sau đó nó vừa được dùng làm biến chạy (tăng lên 1 sau mỗi lần lặp) vừa được dùng để cộng dồn vào biến **sum**. Tại thời điểm kết thúc, chương trình sẽ in ra **Sum of 1 to 5 is 15**.

Vòng lặp for

Vòng lặp for cung cấp một dạng kết hợp tất cả các đặc điểm chung của tất cả các loại vòng lặp: giá trị khởi tạo của biến chạy, điều kiện dừng của vòng lặp và lệnh thay đổi giá trị của biến chạy. Cú pháp:

```
for(initialization statements; condition; increment statements)
{
        action statements;
}
```

initialization statements: khởi tạo giá trị ban đầu cho các biến chạy, các lệnh khởi tạo được phân cách nhau bởi dấu phẩy và chỉ thực hiện duy nhất một lần vào thời điểm bắt đầu của vòng lặp.

condition: Biểu thức bool; vòng lặp sẽ tiếp tục cho đến khi nào điều kiện có giá trị False.

increment statements: Các câu lệnh thay đổi giá trị của biến chạy. Các lệnh này luôn được thực hiện sau mỗi lần thực hiện khối lệnh trong vòng lặp. Các lệnh phận biệt nhau bởi dấu phẩy.

Đoạn chương trình sau hiển thi tổng của 5 số đầu tiên dùng vòng lặp for.

```
package vidu;
 class ForDemo
 {
```

```
    public static void main(String args[])
    {
       int sum = 0;
       for (int i=1; i<=5; i++)
          sum += i;
       System.out.println (“The sum is “ + sum);
    }
}
```

---

ví dụ trên, i và sum là hai biến được gán các giá trị đầu là 1 và 0 tương ứng. Điều kiện được kiểm tra và khi nó còn nhận giá trị True, câu lệnh tác động trong vòng lặp được thực hiện. Tiếp theo giá trị của i được tăng lên 2 để tạo ra số chẵn tiếp theo. Một lần nữa, điều kiện lại được kiểm tra và câu lệnh tác động lại được thực hiện. Sau năm vòng, i tăng lên 6, điều kiện trả về giá trị False và vòng lặp kết thúc. Thông báo: **The sum is 15** được hiển thị.

1.2. Vào ra chuẩn

Java sử dụng khái niệm stream cho các luồng I/O. Gói java.io chứa các lớp cần thiết cho hoạt động input và output. Một stream là một dãy dữ liệu. Trong java, một stream bao gồm các byte.

Xuất dữ liệu ra Luồng ra chuẩn

Một chương trình sẽ thực hiện 3 thao tác cơ bản: nhận dữ liệu, xử lý dữ liệu và xuất kết quả. Trong các chương trình Java đầu tiên, chúng ta đã làm quen với đối tượng xuất dữ liệu đến thiết bị chuẩn (cụ thể là màn hình) là System.out và các phương thức xuất dữ liệu cơ bản là print và println. Tuy nhiên, print và println có một số hạn chế trong vấn đề định dạng dữ liệu như ví dụ sau:

```
double PI = (double) 3.14f;
System.out.println("PI = " + PI);
```

Kết quả:

```
PI = 3.140000104904175
```

Giả sử chúng ta muốn hiển thị giá trị biến PI = 3.14, tức là chỉ cần hai chữ số phần thập phân thì print hay println sẽ không giải quyết được. Đối tượng System.out có hỗ trợ phương thức printf giúp chúng ta định dạng dữ liệu như ý muốn.

```
double PI = (double) 3.14f;
System.out.printf("PI = %.2f", PI);
```

Kết quả:

```
PI = 3.14
```

Nhập dữ liệu từ luồng vào chuẩn

Để nhập dữ liệu từ thiết bị nhập chuẩn (như bàn phím,…), Java cung cấp lớp *Scanner*. Để dùng lớp này, đầu tiên chúng ta phải tạo một đối tượng nhập (ví dụ *console*) và kết nối nó với thiết bị nhập chuẩn như đoạn mã sau:

```
Scanner console = new Scanner(System.in);
```

Để sử dụng lớp *Scanner* chúng ta cần *import* lớp *Scanner* đến dự án:

```
import java.util.Scanner;
```

Sau khi tạo đối tượng từ lớp *Scanner*, chúng ta sẽ sử dụng các phương thức của lớp *Scanner* để nhận dữ liệu từ thiết bị nhập chuẩn. Một số phương thức phổ biến:

Phương thức	Mô tả	Ví dụ
nextInt()	*Nhận số nguyên*	int a = console.nextInt();
nextDouble()	*Nhận số thực*	double b = console.nextDouble();
next()	*Nhận một chuỗi ký tự*	String str = console.next();
nextLine()	*Nhận một chuỗi ký tự trên dòng kế tiếp*	String str = console.nextLine();
next().charAt(0)	*Nhận một ký tự*	char ch = console.next().charAt(0);

Đoạn chương trình sau đây sẽ minh họa cách tính tổng hai số nguyên được nhập từ bàn phím và hiển thị tổng đó ra màn hình:

```
Scanner console = new Scanner(System.in);
System.out.print("Enter a = ");
int a = console.nextInt();
```

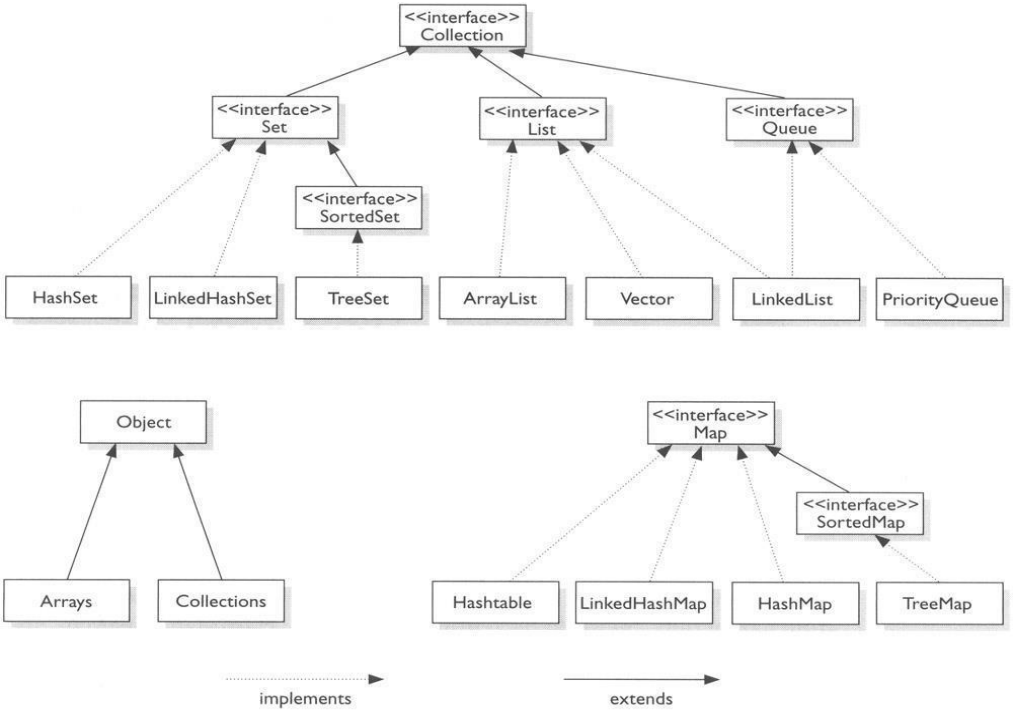
```
System.out.print("Enter b = ");

int b = console.nextInt();

System.out.println();

int c = a + b;

System.out.printf("a + b = %d", c);
```

1.3. Thư viện Java Collections

1.3.1. Các thành phần của Collection

Hình vẽ dưới đây mô tả cấu trúc chung của các thành phần trong thư viện collection nói chung. Trong đó:

- Phía trên là các interface mô tả các kiểu cơ bản và được hiện thực hóa trong các lớp ở mức dưới.
- Các lớp trong cấu trúc cây sẽ có tập hành động tương tự nhau dựa trên việc cài đặt các hàm thống nhất (giới thiệu trong phần sau).



Hình 1.1: Cấu trúc các lớp trong thư viện Collection

1.2.2. Duyệt Collection

Có 2 cách duyệt Collection: (1) dùng cấu trúc for- each; (2) sử dụng Iterators

Sử dụng for – each:

Cấu trúc for-each cho phép duyệt Collection hoặc Mảng sử dụng vòng for. Ví dụ sau in ra mỗi phần tử trên 1 dòng.

VD: for (Object o : collection)

System.out.println(o);

Sử dụng Iterator:

Iterator là 1 đối tượng cho phép duyệt Collection, xóa phần tử trong Collection 1 cách có chọn lựa. Tạo 1 đối tượng Iterator bằng cách gọi phương thức iterator. Dưới đây là interface Iterator:

public interface Iterator<E> {

boolean hasNext();

E next();

void remove(); //optional

}

Phương thức hasNext trả về true nếu còn phần tử, và phương thức next trả về phần tử tiếp theo của vòng lặp. Phương thức remove xóa phần tử được trả về bởi next. Mỗi lần gọi hàm next chỉ được gọi 1 lần hàm remove, và exception sẽ sinh ra nếu quy tắc này không được tuân thủ.

Sử dụng Iterator thay vì cấu trúc for – each trong các trường hợp:

- Xóa phần tử hiện tại. Cấu trúc for- each ẩn vòng lặp đi, vì thế không thể gọi hàm remove. Do đó, cấu trúc for-each không sử dụng để lọc được.
- Lặp song song qua nhiều Collection

Phương thức sau đây chỉ ra cách sử dụng 1 Iterator để lọc 1 Collection tùy ý (duyệt qua Collection và xóa phần tử chỉ định)

static void filter(Collection<?> c) {

for (Iterator<?> it = c.iterator(); it.hasNext();)

if (!cond(it.next()))

it.remove();

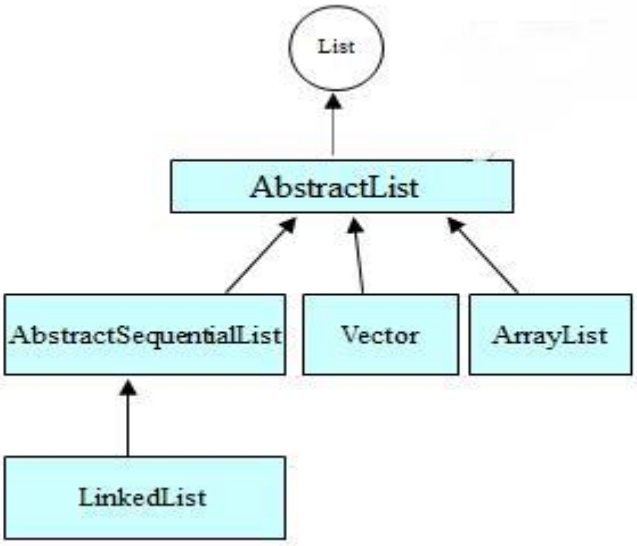
}

1.2.3 Thao tác số lượng lớn (Bulk)

Thao tác Bulk thực hiện trên toàn bộ Collection. Có thể cài đặt những thao tác Bulk bằng những thao tác cơ bản. Các thao tác Bulk phổ biến:

- containsAll – trả về true nếu Collection đích chứa toàn bộ các phần tử có trong Collection chỉ định.
- addAll – thêm tất cả phần tử trong Collection chỉ định vào Collection đích.
- removeAll – xóa tất cả những phần tử trong Collection đích mà có trong Collection chỉ định.
- retainAll - xóa tất cả những phần tử trong Collection đích mà không có trong Collection chỉ định. Nghĩa là, nó chỉ giữ lại những phần tử có ở trong Collection chỉ định.
- clear – xóa toàn bộ phần tử trong Collection.

1.2.4 Giới thiệu List Interface



Hình 1.2. Các lớp danh sách trong Collection

List là một Collection có thứ tự. List có thể chứa các thành phần giống nhau. Ngoài chức năng thừa hưởng từ Colection.

Ví dụ List interface:

public interface List<E> extends Collection<E> {

Positional
access E get(int
index);

optional

E set(int index, E element);

// optional

boolean add(E element);

// optional

void add(int index, E element);

// optional

E remove(int index);

// optional

boolean addAll(int index, Collection<? extends E> c);

// Search

int indexOf(Object o);

int lastIndexOf(Object o);

// Iteration

ListIterator<E> listIterator();

ListIterator<E> listIterator(int index);

// Range-view

List<E> subList(int from, int to);

}

Trong Java, INTERFACE LIST, bao gồm 3 phần là ARRAY LIST, VECTOR VÀ LINKED LIST. ArrayList giúp việc thực hiện hoạt động tốt hơn, và LinkedList cung cấp hiệu suất tốt hơn trong những tình huống nhất định. Ngoài ra, Vector đã được thêm để thực hiện List .

1.2.5. Hàng đợi (Queue)

Định nghĩa: Một Queue là một Collection để lưu trữ các phần tử trước khi cài đặtviêc truy cập. Bên cạnh các hoạt động Collection cơ bản , queue cung cấp việc chèn thêm , loại bỏ , và các hoạt động kiểm tra . Interface queueđược cho sau đây:

public interface Queue<E> extends Collection<E> {

element();
boolean
offer(E e); E
peek();

E poll();

E remove();

}

Mỗi phương thức Queue tồn tại dưới hai hình thức: (1) một ném một Exception (ngoại lệ) nếu hoạt động bị lỗi , và (2) ngoài ra là trả về một giá trị đặc biệt nếu hoạt động bị lỗi (hoặc null hoặc sai , tùy thuộc vào hoạt động) . Cấu trúc thông thường của interface được minh họa trong bảng dưới đây .

CẤU TRÚC INTERFACE HÀNG ĐỢI

Loại hoạt động	**Ném trả lại ngoại lệ**	**Trả lại giá trị đặc biệt**
Insert	add(e)	Offer(e)
Remove	remove()	Poll()
Examine	Element()	Peek()

- Phương thức add, thừa kế từ Collection ,là chèn một phần tử trừ khi nó sẽ vi phạm các hạn chế khả năng của queue, trong trường hợp nó ném ExceptionIllegalStateException.
- Phương thức offer, được sử dụng trên Queue bị chặn, khác với add duy nhất ở chỗ trả về false nếu không thể thêm được phần tử vào.
- Các phương thức remove và poll, cả hai đều xóa và trả lại (giá trị) đầu queue. Các phương remove và poll thể hiện sự khác nhau chỉ khi queue rỗng . Trong hoàn cảnh này, phương thức remove ném ra NoSuchElementException , trong khi poll trả về null.
- Các phương thức peek và element trả về, nhưng không xóa phần tử đứng đầu của hàng đợi. Chúng khác nhau ở điểm : Nếu queue rỗng , phương thức element ném ra Exception NoSuchElementException , trong khi peek trả về null.

Trong ví dụ sau đây , một Queue ưu tiên (priority Queue) được sử dụng để SẮP XẾP một tập hợp các phần tử . Mục đích của đoạn chương trình là để kiểm tra sự sắp xếp của Queue ưu tiên:

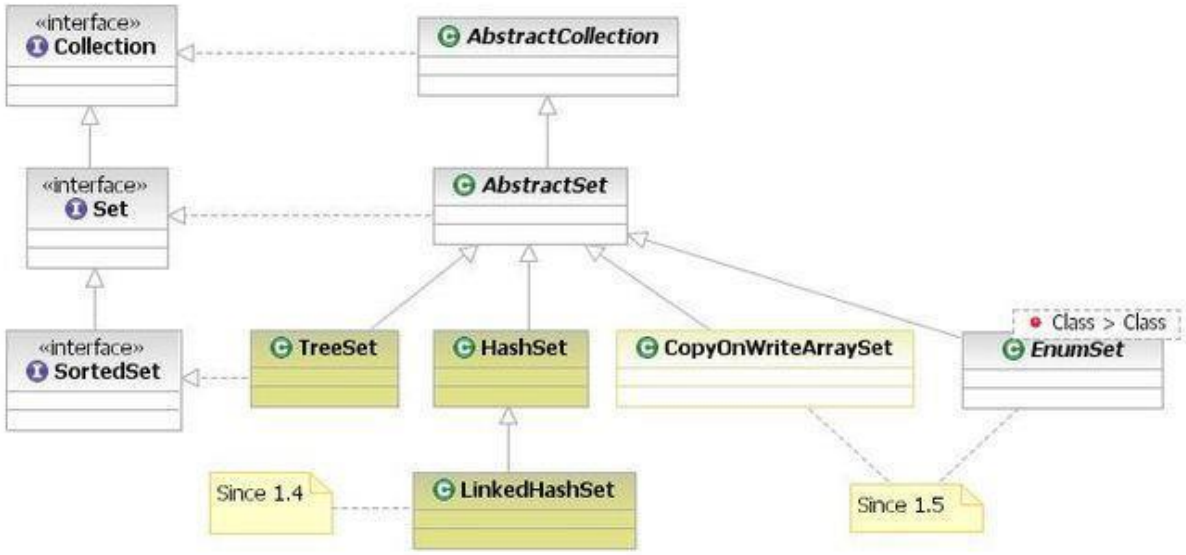
Ví dụ:

```
static <E> List<E> heapSort(Collection<E> c) {
Queue<E> queue = new PriorityQueue<E>(c);
List<E> result = new ArrayList<E>();
while (!queue.isEmpty())
result.add(queue.remove());
```

return result;

}

1.2.6. Set Interface:



Hình 1.3. Cấu trúc Set Interface

Set là một loại Collections không chứa phần tử trùng nhau. Set biểu diễn một cách trừu tượng khái niệm tập hợp trong toán học. Trong interface Set chỉ bao gồm các phương thức được thừa kế từ Collections và thêm vào giới hạn là không cho phép có phần tử trùng nhau. Set cũng có những phương thức là equals và hashCode , cho phép những thể hiện của Set có thể dễ dàng so sánh với nhau ngay cả trong trường hợp chúng thuộc những loại thực thi khác nhau. Hai thể hiện được gọi là bằng nhau nếu chúng chứa những phần tử như nhau.

Dưới đây những phương thức nằm trong Set interface :

public interface Set<E> extends Collection<E> {

Toan tu co ban
int size();
boolean
isEmpty();

boolean contains(Object element);

Tùy chọn

boolean add(E element);

// Tùy chọn

boolean remove(Object element);

Iterator<E> iterator();

// Toan tu so luong lon

boolean containsAll(Collection<?> c);

// Tùy chọn

boolean addAll(Collection<? extends E> c);

// Tùy chọn

boolean removeAll(Collection<?> c);

// Tùy chọn

boolean retainAll(Collection<?> c);

// Tùy chọn

void clear();

Toan tu cua mang
Object[] toArray();
<T> T[] toArray(T[]
a);

}

Nền tảng Java bao gồm 3 lớp thực thi chính của Set là HashSet, TreeSet, LinkedHashSet.

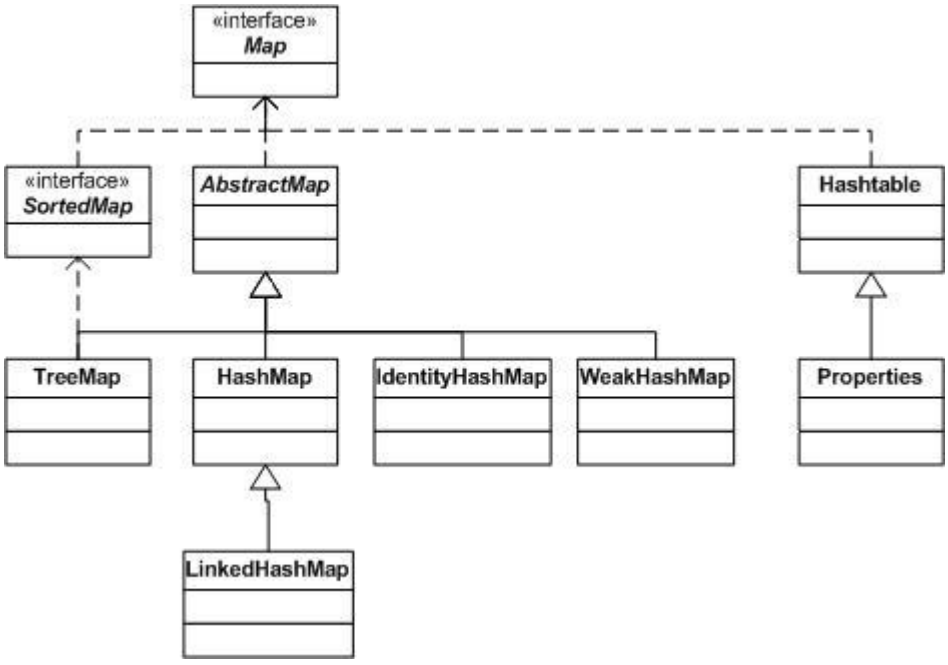
- HashSet lưu trữ phần tử trong bảng băm (hash table). Đây là lớp thực thi cho tốc độ tốt nhất, tuy nhiên lại không đảm bảo thứ tự của phần tử.
- TreeSet lưu trữ phần tử trong cây đỏ-đen (red – black tree), sắp xếp phần tử dựa trên giá trị của chúng, và do đó chậm hơn đáng kể so với HashSet.
- LinkedHashSet cũng dùng bảng băm để lưu phần tử, ngoài ra còn sử dụng danh sách liên kết để sắp xếp phần tử theo thứ tự mà phần tử được chèn vào tập hợp (insertion-order). LinkedHashSet giúp người dùng loại bỏ những

trật tự hỗn độn không đoán trước gây ra bởi HashSet và với chi phí chỉ cao hơn một chút.

Những toán tử cơ bản trong Set Interface (Basic Operations)

- int size() : trả về số phần tử trong Set (lực lượng của chúng)
- boolean isEmpty() : kiểm tra xem Set rỗng hay không , trả về true nếu Set rỗng
- boolean add(E element) : thêm phần tử vào trong Set nếu nó chưa có trong Set đó, trả về true nếu chưa có phần tử trùng trong Set và false trong trường hợp ngược lại .
- boolean remove(Object element) : xóa một phần tử được chỉ định trong Set. Trả về true nếu phần tử đó tồn tại trong Set và false trong trường hợp ngược lại.
- Iterator<E> iterator() : trả về kiểu Iterator của Set

1.2.7. Map Interface



Hình 1.4. Cấu trúc Map Interface

Map là 1 đối tượng ánh xạ khóa tới giá trị. 1 Map không thể chứa khóa trùng nhau: mỗi khóa có thể ánh xạ đến nhiều nhất 1 giá trị. Map mô hình hóa khái niệm trừu tượng "hàm" trong Toán học. Dưới đây là interface Map:

public interface Map<K,V> {

// Thao tác cơ bản

V put(K key, V value);

```
V get(Object key);
V remove(Object key);
boolean containsKey(Object key);
boolean containsValue(Object value);
int size();
boolean isEmpty();
// Thao tác số lượng lớn
void putAll(Map<? extends K, ? extends V> m);
void clear();
    Collection Views public
  Set<K> keySet(); public
  Collection<V> values();
public Set<Map.Entry<K,V>> entrySet();
    Interface for entrySet elements
public interface Entry {
     getKey(); V getValue()
    V setValue(V value);
}
}
```

Có 3 cài đặt chính của Map trong nền tảng Java: HashMap, TreeMap và LinkedHashMap. Bổ sung bên cạnh là đối tượng Hashtable.

Multimaps

Một multimap giống như 1 Map nhưng mỗi khóa có thể ánh xạ đến nhiều hơn 1 giá trị. Java Collections Framework không bao gồm multimap interface do nó không được thường xuyên sử dụng. Cho ví dụ , đọc 1 danh sách từ, mỗi từ 1 dòng (tất cả chữ thường) và in ra những nhóm đảo chữ thỏa mãn kích cỡ. Một nhóm đảo chữ (anagram group) là 1 tập các từ, tất cả những chữ cái giống nhau nhưng thứ tự xuất hiện khác nhau. Chương trình sử dụng 2 tham số dòng lệnh (1) tên file input và (2) kích cỡ nhỏ nhất của nhóm đảo chữ in ra. Nhóm nào có ít từ hơn giá trị chỉ định sẽ không được in ra.

Có 1 số mẹo trong việc tìm nhóm đảo chữ: Với mỗi từ trong file input, sắp xếp nó lại theo thứ tự bảng chữ cái (alphabetize), đưa nó vào multimap và ánh xạ nó với từ nguyên gốc. Ví dụ, từ bad tạo ra sắp xếp cặp khóa – giá trị là <abd, bad> và đưa vào multimap. Nhiệm vụ sau cùng khá đơn giản, đó là duyệt qua multimap và đưa ra các nhóm thỏa mãn yêu cầu kích cỡ.

Chương trình sau được cài đặt thuần túy theo phương pháp trên:

```
import java.util.*;
import java.io.*;
public class Anagrams {
public static void main(String[] args) {
int minGroupSize = Integer.parseInt(args[1]);

// Read words from file and put into a simulated multimap
Map<String, List<String>> m = new HashMap<String, List<String>>();
try {
Scanner s = new Scanner(new File(args[0]));
while (s.hasNext()) {
String word = s.next();
String alpha = alphabetize(word);
List<String> l = m.get(alpha);
if (l == null)
m.put(alpha, l=new ArrayList<String>());
l.add(word);
}} catch (IOException e) {
System.err.println(e);
System.exit(1);
}
        Print all permutation groups above size
        threshold for (List<String> l : m.values())
```

```
if (l.size() >= minGroupSize)
System.out.println(l.size() + ": " + l);
}
private static String alphabetize(String s) {
char[] a = s.toCharArray();
Arrays.sort(a);
return new String(a);
}
}
```

CHƯƠNG 2. LÝ THUYẾT TỔ HỢP

2.1. Những kiến thức cơ bản về lý thuyết tập hợp

2.1.1. Khái niệm & định nghĩa

Các tập hợp dùng để nhóm các đối tượng lại với nhau. Thông thường, các đối tượng trong tập hợp có các tính chất tương tự nhau. Ví dụ, tất cả sinh viên mới nhập trường tạo nên một tập hợp, tất cả sinh viên thuộc khoa Công nghệ thông tin là một tập hợp, các số tự nhiên, các số thực . . . cũng tạo nên các tập hợp. Chú ý rằng, thuật ngữ đối tượng được dùng ở đây không chỉ rõ cụ thể một đối tượng nào, sự mô tả một tập hợp nào đó hoàn toàn mang tính trực giác về các đối tượng.

Định nghĩa 1. Tập các đối tượng trong một tập hợp được gọi là các phần tử của tập hợp. Các tập hợp thường được ký hiệu bởi những chữ cái in hoa đậm như A, B, X, Y . . ., các phần tử thuộc tập hợp hay được ký hiệu bởi các chữ cái in thường như a, b, c, u, v . . . Để chỉ a là phần tử của tập hợp A ta viết $a \in A$, trái lại nếu a không thuộc A ta viết $a \notin A$.

Tập hợp không chứa bất kỳ một phần tử nào được gọi là tập rỗng (kí hiệu là ϕ hoặc { })

Tập hợp A được gọi là bằng tập hợp B khi và chỉ khi chúng có cùng chung các phần tử và được kí hiệu là A=B. Ví dụ tập A={ 1, 3, 5 } sẽ bằng tập B = { 3, 5, 1 }.

Định nghĩa 2. Tập A được gọi là một tập con của tập hợp B và ký hiệu là $A \subseteq B$ khi và chỉ khi mỗi phần tử của A là một phần tử của B. Hay $A \subseteq B$ khi và chỉ khi lượng từ

$$\forall x\ (x \in A \rightarrow x \in B)$$ cho ta giá trị đúng.

Từ định nghĩa trên chúng ta rút ra một số hệ quả sau:

- Tập rỗng ϕ là tập con của mọi tập hợp.
- Mọi tập hợp là tập con của chính nó.
- Nếu $A \subseteq B$ và $B \subseteq A$ thì A=B hay mệnh đề :

  $x\ (x \in A \rightarrow x \in B) \vee \forall x\ (x \in B \rightarrow x \in A)$ cho ta giá trị đúng.
- Nếu $A \subseteq B$ và $A \neq B$ thì ta nói A là tập con thực sự của B và ký hiệu là $A \subset B$.

Định nghĩa 3. Cho S là một tập hợp. Nếu S có chính xác n phần tử phân biệt trong S, với n là số nguyên không âm thì ta nói S là một tập hữu hạn và n được gọi là bản số của S. Bản số của S được ký hiệu là |S |.

Định nghĩa 4. Cho tập hợp S. Tập luỹ thừa của S ký hiệu là P(S) là tập tất cả các tập con của S.

Ví dụ $S = \{ 0, 1, 2 \} \Rightarrow P(S) = \{ \phi, \{0\}, \{1\}, \{2\}, \{0,1\}, \{0, 2\}, \{1, 2\}\ \{0, 1, 2\}\}$.

Định nghĩa 5. Dãy sắp thứ tự $(a_1, a_2, .., a_n)$ là một tập hợp sắp thứ tự có a_1 là phần tử thứ nhất, a_2 là phần tử thứ 2, .., a_n là phần tử thứ n.

Chúng ta nói hai dãy sắp thứ tự là bằng nhau khi và chỉ khi các phần tử tương ứng của chúng là bằng nhau. Nói cách khác $(a_1, a_2, .., a_n)$ bằng $(b_1, b_2, .., b_n)$ khi và chỉ khi $a_i = b_i$ với mọi i =1, 2, ..n.

Định nghĩa 6. Cho A và B là hai tập hợp. Tích đề các của A và B được ký hiệu là A×B, là tập hợp của tất cả các cặp (a,b) với a∈A, b ∈B. Hay có thể biểu diễn bằng biểu thức:

$$A \times B = \{ (a, b) \mid a \in A \wedge b \in B \}$$

Định nghĩa 7. Tích đề các của các tập $A_1, A_2, . ., A_n$ được ký hiệu là $A_1 \times A_2 \times .. \times A_n$ là tập hợp của dãy sắp thứ tự $(a_1, a_2,.., a_n)$ trong đó $a_i \in A_i$ với i = 1, 2,..n. Nói cách khác:

$$A_1 \times A_2 \times .. \times A_n = \{ (a_1, a_2,.., a_n) \mid a_i \in A_i \text{ với } i = 1, 2,..n \}$$

2.1.2. Các phép toán trên tập hợp

Các tập hợp có thể được tổ hợp với nhau theo nhiều cách khác nhau thông qua các phép toán trên tập hợp. Các phép toán trên tập hợp bao gồm: Phép hợp (Union), phép giao (Intersection), phép trừ (Minus).

Định nghĩa 1. Cho A và B là hai tập hợp. Hợp của A và B được ký hiệu là A∪B, là tập chứa tất cả các phần tử hoặc thuộc tập hợp A hoặc thuộc tập hợp B. Nói cách khác:

$$A \cup B = \{ x \mid x \in A \vee x \in B \}$$

Định nghĩa 2. Cho A và B là hai tập hợp. Giao của A và B được ký hiệu là A∩B, là tập chứa tất cả các phần tử thuộc A và thuộc B. Nói cách khác:

$$A \cup B = \{ x \mid x \in A \wedge x \in B \}$$

Định nghĩa 3. Hai tập hợp A và B được gọi là rời nhau nếu giao của chúng là tập rỗng ($A \cap B = \phi$).

Định nghĩa 4. Cho A và B là hai tập hợp. Hiệu của A và B là tập hợp được ký hiệu là A-B, có các phần tử thuộc tập hợp A nhưng không thuộc tập hợp B. Hiệu của A và B còn được gọi là phần bù của B đối với A. Nói cách khác:

$$A - B = \{ x \mid x \in A \wedge x \notin B \}$$

Định nghĩa 5. Cho tập hợp A. Ta gọi $\overline{A}$ là phần bù của A là một tập hợp bao gồm những phần tử không thuộc A. Hay :

$$\overline{A} = \{x \mid x \notin A\}$$

Định nghĩa 6. Cho các tập hợp $A_1, A_2, . ., A_n$. Hợp của các tập hợp là tập hợp chứa tất cả các phần tử thuộc ít nhất một trong số các tập hợp A_i (i=1, 2, . ., n). Ký hiệu:

$$\bigcup_{i=1}^{n} A_i = A_1 \cup A_2 \cup \cdots \cup A_n$$

Định nghĩa 7: Cho các tập hợp $A_1, A_2, . ., A_n$. Giao của các tập hợp là tập hợp chứa các phần tử thuộc tất cả n tập hợp A_i (i=1, 2, . ., n).

$$\bigcup_{i=1}^{n} A_i = A_1 \cap A_2 \cap .. A_n$$

2.1.3. Các hằng đẳng thức trên tập hợp

Mỗi tập con của tập hợp tương ứng với một tính chất xác định trên tập hợp đã cho được gọi là mệnh đề. Với tương ứng này, các phép toán trên tập hợp được chuyển sang các phép toán của logic mệnh đề:

- Phủ định của A, ký hiệu $\overline{A}$ (hay NOT A) tương ứng với phần bù $\overline{A}$
- Tuyển của A và B, ký hiệu A ∨ B (hay A or B) tương ứng với A ∪ B
- Hội của A và B, ký hiệu A ∧ B (hay A and B) tương ứng với A ∩ B

Các mệnh đề cùng với các phép toán trên nó lập thành một đại số mệnh đề (hay đại số logic). Như thế, đại số tập hợp và đại số logic là hai đại số đẳng cấu với nhau (những mệnh đề phát biểu trên đại số logic tương đương với mệnh đề phát biểu trên đại số tập hợp). Với những trường hợp cụ thể, tuỳ theo tình huống, một bài toán có thể được phát biểu bằng ngôn ngữ của đại số logic hay ngôn ngữ của đại số tập hợp. Bảng 1.5 thể hiện một số hằng đẳng thức của đại số tập hợp.

Ta gọi U là tập hợp vũ trụ hay tập hợp của tất cả các tập hợp.

Bảng 2.1: Một số hằng đẳng thức trên tập hợp

HẰNG ĐẲNG THỨC	TÊN GỌI
A ∪ ϕ = A A ∩ U = A (U là tập vũ trụ)	Luật đồng nhất
A ∪ U = U A ∩ ϕ = A	Luật nuốt
A∩A = A A ∪ A = A	Luật luỹ đẳng
$\overline{\overline{A}}$ = A	Luật bù
A ∩ B = B ∩ A A ∪ B = B ∪ A	Luật giao hoán
A ∪ (B ∪ C) = (A ∪B)∪C A∩ (B ∩ C) = (A∩B) ∩ C	Luật kết hợp
A ∪ (B ∩ C) = (A ∪ B) ∪ (A ∩ C) A ∩ (B ∪ C) = (A ∪ B) ∩ (A ∪ C)	Luật phân phối
$\overline{A \cup B} = \overline{A} \cap \overline{B}$ $\overline{A \cap B} = \overline{A} \cup \overline{B}$	Luật De Morgan

2.2. Những nguyên lý đếm cơ bản

2.2.1 Nguyên lý cộng

Giả sử có hai công việc. Việc thứ nhất có thể tiến hành bằng n_1 cách, việc thứ hai có thể tiến hành bằng n_2 cách và nếu hai việc này không thể tiến hành đồng thời. Khi đó sẽ có $n_1 + n_2$ cách để giải giải quyết một trong hai việc trên.

Chúng ta có thể mở rộng qui tắc cộng cho trường hợp nhiều hơn hai công việc. Giả sử các việc T_1, T_2,.., T_m có thể làm tương ứng bằng n_1, n_2, .., n_m cách và giả sử không có hai việc T_i, T_j nào làm việc đồng thời (i,j = 1, 2, .., m ; $i \neq j$). Khi đó, có $n_1 + n_2 + .. + n_m$ cách thực hiện một trong các công việc T_1, T_2, . ., T_m.

Qui tắc cộng được phát biểu dưới dạng của ngôn ngữ tập hợp như sau:

- Nếu A và B là hai tập rời nhau ($A \cap B = \phi$) thì : $N(A \cup B) = N(A) + N(B)$.
- Nếu A_1, A_2, .., A_n là những tập hợp rời nhau thì:

$$N(A_1 \cup A_2 \cup . . \cup A_n) = N(A_1) + N(A_2) + .. + N(A_n).$$

Ví dụ 1. Giả sử cần chọn hoặc một cán bộ hoặc một sinh viên tham gia một hội đồng của một trường đại học. Hỏi có bao nhiêu cách chọn vị đại biểu này nếu như có 37 cán bộ và 63 sinh viên.

Giải: gọi việc thứ nhất là chọn một cán bộ từ tập cán bộ ta có 37 cách. Gọi việc thứ hai là chọn một sinh viên từ tập sinh viên ta có 63 cách. Vì tập cán bộ và tập sinh viên là rời nhau, theo nguyên lý cộng ta có tổng số cách chọn vị đại biểu này là 37 + 63 = 100 cách chọn.

Ví dụ 2. một đoàn vận động viên gồm môn bắn súng và bơi được cử đi thi đấu ở nước ngoài. Số vận động viên nam là 10 người. Số vận động viên thi bắn súng kể cả nam và nữ là 14 người. Số nữ vận động viên thi bơi bằng số vận động viên nam thi bắn súng. Hỏi đoàn có bao nhiêu người.

Giải: chia đoàn thành hai tập, tập các vận động viên nam và tập các vận động viên nữ. Ta nhận thấy tập nữ lại được chia thành hai: thi bắn súng và thi bơi. Thay số nữ thi bơi bằng số nam thi bắn súng , ta được số nữ bằng tổng số vận động viên thi bắn súng. Từ đó theo nguyên lý cộng toàn đoàn có 14 + 10 = 24 người.

Ví dụ 3. giá trị của biến k sẽ bằng bao nhiêu sau khi thực hiện đoạn chương trình sau :

```
k := 0
for i1:= 1to n1
        k:=k+1
for i2:= 1to n2
        k:=k+1
. . . . . .
. . . . . .

. . . . . .
```

for i_m:= 1 to n_m

k:=k+1

Giải: coi mỗi vòng for là một công việc, do đó ta có m công việc T_1, T_2, . ., T_m. Trong đó T_i thực hiện bởi n_i cách (i= 1, 2, .., m). Vì các vòng for không lồng nhau hay các công việc không thực hiện đồng thời nên theo nguyên lý cộng tổng tất cả các cách để hoàn thành T_1, T_2,.., T_m là k= n_1 + n_2 +.. + n_m.

2.2.2. Nguyên lý nhân

Giả sử một nhiệm vụ nào đó được tách ra hai công việc. Việc thứ nhất được thực hiện bằng n_1 cách, việc thứ hai được thực hiện bằng n_2 cách sau khi việc thứ nhất đã được làm, khi đó sẽ có $n_1.n_2$ cách thực hiện nhiệm vụ này.

Nguyên lý nhân có thể được phát biểu tổng quát bằng ngôn ngữ tập hợp như sau:

Nếu A_1, A_2, .., A_m là những tập hợp hữu hạn, khi đó số phần tử của tích đề các các tập này bằng tích số các phần tử của mỗi tập thành phần. Hay đẳng thức:

$$N (A_1 \times A_2 \times .. A_m) = N (A_1) N (A_2) . . . N (A_m).$$

Nếu $A_1 = A_2 =. . A_m$ thì $N(A^k) = N(A)^k$

Ví dụ 1. giá trị của k sẽ bằng bao nhiêu sau khi ta thực hiện đoạn chương trình sau:

k:=0

for i_1 = 1 to n_1

for i_2 = 1 to n_2

……

. . ……

for i_n =1 to n_m

k:=k +1

Giải : Giá trị khởi tạo k=0. Mỗi vòng lặp kồng nhau đi qua giá trị của k được tăng lên 1 đơn vị. Gọi T_i là việc thi hành vòng lặp thứ i. Khi đó, số lần vòng lặp là số cách thực hiện công việc. Số cách thực hiện công việc T_j là n_j (j=1,2, . ., n). Theo qui tắc nhân ta vòng lặp kép được duyệt qua n_1 +n_2 +..+n_m lần và chính là giá trị của k.

Ví dụ 2. Người ta có thể ghi nhãn cho những chiếc ghế của một giảng đường bằng một chữ cái và sau đó là một số nguyên nhỏ hơn 100. Bằng cách như vậy hỏi có nhiều nhất bao nhiêu chiếc ghế có thể ghi nhãn khác nhau.

Giải: có nhiều nhất là 26 x 100 = 2600 ghế được ghi nhãn. Vì kí tự gán nhãn đầu tiên là một chữ cái vậy có 26 cách chọn các chữ cái khác nhau để ghi kí tự đầu tiên, tiếp theo sau là một số nguyên dương nhỏ hơn 100 do vậy có 100 cách chọn các số nguyên để gán tiếp sau của một nhãn. Theo qui tắc nhân ta nhận được 26 x 100 = 2600 nhãn khác nhau.

Ví dụ 3. Có bao nhiêu xâu nhị phân có độ dài 7.

Giải: một xâu nhị phân có độ dài 7 gồm 7 bít, mỗi bít có hai cách chọn (hoặc giá trị 0 hoặc giá trị 1), theo qui tắc nhân ta có $2.2.2.2.2.2.2 = 2^7 = 128$ xâu bít nị phân độ dài 7.

Ví dụ 4. Có bao nhiêu hàm đơn ánh xác định từ một tập A có m phần tử nhận giá trị trên tập B có n phần tử.

Giải : Trước tiên ta nhận thấy, nếu m >n thì tồn tại ít nhất hai phần tử khác nhau của A cùng nhận một giá trị trên B, như vậy với m>n thì số các hàm đơn ánh từ $A \rightarrow B$ là 0. Nếu m<=n, khi đó phần tử đầu tiên của A có n cách chọn, phần tử thứ hai có n-1 cách chọn, . ., phần tử thứ k có n-k+1 cách chọn. Theo qui tắc nhân ta có n(n-1) (n-2) . . .(n-m+1) hàm đơn ánh từ tập A sang tập B.

Ví dụ 5. Dạng của số điện thoại ở Bắc Mỹ được qui định như sau: số điện thoại gồm 10 chữ số được tách ra thành một nhóm mã vùng gồm 3 chữ số, nhóm mã chi nhánh gồm 3 chữ số và nhóm mã máy gồm 4 chữ số. Vì những nguyên nhân kỹ thuật nên có một số hạn chế đối với một số con số. Ta giả sử, X biểu thị một số có thể nhận các giá trị từ 0..9, N là số có thể nhận các chữ số từ 2..9, Y là các số có thể nhận các chữ số 0 hoặc 1.

Hỏi theo hai dự án đánh số NYX NNX XXXX và NXX NXX XXXX có bao nhiêu số điện thoại được đánh số khác nhau ở Bắc Mỹ.

Giải: đánh số theo dự án NYX NNX XXXX được nhiều nhất là :

$$8 \times 2 \times 10 \times 8 \times 8 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 = 2 \times 8^3 \times 10^6 = 1\,024. 10^6$$

đánh số theo dự án NXX NXX XXXX được nhiều nhất là :

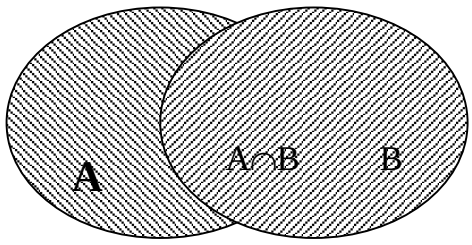
$$8 \times 10 \times 10 \times 8 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 = 8^2 \times 10^8 = 64. 10^8$$

Ví dụ 6. Dùng qui tắc nhân hãy chỉ ra rằng số tập con của một tập S hữu hạn là $2^{N(S)}$.

Giải: ta liệt kê các phần tử của tập S là $s_1, s_2, .., s_{N(S)}$. Xây dựng một xâu bít nhị phân dài N(S) bít, trong đó nếu bít thứ i có giá trị 0 thì phần tử $s_i \notin S$, nếu bít thứ i có giá trị 1 thì phần tử $s_i \in S$ (i=1, 2, .., N(S)). Như vậy, theo nguyên lý nhân, số tập con của tập hợp S chính là số xâu bít nhị phân có độ dài N(S). Theo ví dụ 3, chúng ta có $2^{N(S)}$ xâu bít nhị phân độ dài N(S).

2.2.3. Nguyên lý bù trừ

Trong một số bài toán đếm phức tạp hơn. Nếu không có giả thiết gì về sự rời nhau giữa hai tập A và B thì $N(A \cup B) = N(A) + N(B) - N(A \cap B)$.



Ví dụ 1. lớp toán học rời rạc có 25 sinh viên giỏi tin học, 13 sinh viên giỏi toán và 8 sinh viên giỏi cả toán và tin học. Hỏi lớp có bao nhiêu sinh viên nếu mỗi sinh viên hoặc giỏi toán hoặc học giỏi tin học hoặc giỏi cả hai môn?

Giải: Gọi A tập là tập các sinh viên giỏi Tin học, B là tập các sinh viên giỏi toán. Khi đó $A \cap B$ là tập sinh viên giỏi cả toán học và tin học. Vì mỗi sinh viên trong lớp hoặc giỏi toán, hoặc giỏi tin học hoặc giỏi cả hai nên ta có tổng số sinh viên trong lớp là $N(A \cup B)$. Do vậy ta có:

$$N(A \cup B) = N(A) + N(B) - N(A \cap B) = 25 + 13 - 8 = 30.$$

Ví dụ 2. Có bao nhiêu số nguyên không lớn hơn 1000 chia hết cho 7 hoặc 11.

Giải : Gọi A là tập các số nguyên không lớn hơn 1000 chia hết cho 7, B là tập các số nguyên không lớn hơn 1000 chia hết cho 11. Khi đó tập số nguyên không lớn hơn 1000 hoặc chia hết cho 7 hoặc chia hết cho 11 là $N(A \cup B)$. Theo công thức 1 ta có:

$$N(A \cup B) = N(A) + N(B) - N(A \cap B) = \lfloor 1000/7 \rfloor + \lfloor 1000/11 \rfloor - \lfloor 1000/7.11 \rfloor$$
$$= 142 + 90 - 12 = 220.$$

Trước khi đưa ra công thức tổng quát cho n tập hợp hữu hạn. Chúng ta đưa ra công thức tính số phần tử của hợp 3 tập A, B, C.

Ta nhận thấy $N(A) + N(B) + N(C)$ đếm một lần những phần tử chỉ thuộc một trong ba tập hợp. Như vậy, số phần tử của $A \cap B$, $A \cap C$, $B \cap C$ được đếm hai lần và bằng $N(A \cap B)$, $N(A \cap C)$, $N(B \cap C)$, được đếm ba lần là những phần tử thuộc $A \cap B \cap C$. Như vậy, biểu thức:

$N(A \cup B \cup C) - N(A \cap B) - N(A \cap C) - N(B \cap C)$ chỉ đếm các phần tử chỉ thuộc một trong ba tập hợp và loại bỏ đi những phần tử được đếm hai lần. Như vậy, số phần tử được đếm ba lần chưa được đếm, nên ta phải cộng thêm với giao của cả ba tập hợp. Từ đó ta có công thức đối với 3 tập không rời nhau:

$$N(A \cup B \cup C) = N(A) + N(B) + N(C) - N(A \cap B) - N(A \cap C) - N(B \cap C) + N(A \cap B \cap C)$$

Định lý. Nguyên lý bù trừ. Giả sử $A_1, A_2, . ., A_m$ là những tập hữu hạn. Khi đó

$$N(A_1 \cup A_2 \cup . . . \cup A_m) = N_1 - N_2 + . . + (-1)^{m-1} N_m, \quad (2)$$

trong đó N_k là tổng phần tử của tất cả các giao của k tập lấy từ m tập đã cho. (nói riêng $N_1 = N(A_1) + N(A_2) + . . + N(A_m)$, $N_m = N(A_1 \cap A_2 \cap . . . \cap A_m)$. Nói cách khác:

$$N(A_1 \cup A_2 \cup ... A_n) = \sum_{1 \le i \le n} N(A_i) - \sum_{1 \le i, j < n} N(A_i \cap A_j) + \sum_{1 \le i < j < k \le n} N(A_i \cap A_j \cap A_k - ... + (-1)^{n+1} N(A_1 \cap A_2 \cap .. \cap A_n)$$

Định lý được chứng minh bằng cách chỉ ra mỗi phần tử của hợp n tập hợp được đếm đúng một lần. Bạn đọc có thể tham khảo cách chứng minh trong tài liệu [1].

Ví dụ 3. Tìm công thức tính số phần tử của 4 tập hợp.

Giải : Từ nguyên lý bù trừ ta có

$N(A_1 \cup A_2 \cup A_3 \cup A_4) = N(A_1) + N(A_2) + N(A_3) + N(A_4) - N(A_1 \cap A_2) - N(A_1 \cap A_3) - N(A_1 \cap A_4) - N(A_2 \cap A_3) - N(A_2 \cap A_4) - N(A_3 \cap A_4) + N(A_1 \cap A_2 \cap A_3) + N(A_1 \cap A_2 \cap A_4) + N(A_1 \cap A_3 \cap A_4) + N(A_2 \cap A_3 \cap A_4) - N(A_1 \cap A_2 \cap A_3 \cap A_4)$.

Ví dụ 4. Hỏi trong tập X = { 1, 2, . ., 10000} có bao nhiêu số không chia hết cho bất cứ số nào trong các số 3, 4, 7.

Giải: Gọi A là tập các số nhỏ hơn 10000 chia hết cho 3, B là tập các số nhỏ hơn 10000 chia hết cho 4, C là tập các số nhỏ hơn 10000 chia hết cho 7. Theo nguyên lý bù trừ ta có:

$N(A \cup B \cup C) = N(A)+N(B) + N(C) – N(A \cap B – N(A \cap C) – N(B \cap C) + N(A \cap B \cap C)$

trong đó :

$N(A) + N(B) + N (C) = [10\ 000/3] + [10\ 000/4] + [10\ 000/7]$

$= 3333 + 2500 + 1428 = 7261$

$N(A \cap B) = N(A) + N(B) – N(A \cap B) = 3333 + 2500 – [10000/3x4] = 833$

$N(A \cap C) = N(A) + N(C) – N(A \cap C) = 3333 + 1428 – [10000/3x7] = 476$

$N(B \cap C) = N(B) + N(C) – N(B \cap C) = 2500 + 1428 – [10000/4x7] = 357$

$N(A \cap B) + N(A \cap C) + N(B \cap C) = 833 + 476 + 357 = 1666$

$N(A \cap B \cap C) = [10000/3x4x7] = 119.$

=>Số các số nhỏ hơn 10000 cần đếm là :

$1000 - N(A \cup B \cup C) = 7261 – 1666 + 119 = 4286.$

Ví dụ 5. Có bao nhiêu xâu nhị phân độ dài 10 bắt đầu bởi 00 hoặc kết thúc bởi 11.

Giải : Gọi A là số xâu nhị phân độ dài 10 bắt đầu bởi 00 , B là số xâu nhị phân độ dài 10 kết thúc bởi 11. Dễ ràng nhận thấy, $N(A) = N(B) = 256$, $N(A \cap B) = 2^6 = 64$. Theo nguyên lý bù trừ ta có:

$N(A \cup B) = N(A) + N(B) – N(A \cap B)$

$= 256 + 256 – 64 = 448.$

Ví dụ 6. Bài toán bỏ thư. Có n lá thư và n phong bì ghi sẵn địa chỉ. Bỏ ngẫu nhiên các lá thư vào các phong bì. Hỏi xác suất để xảy ra không một là thư nào bỏ đúng địa chỉ là bao nhiêu?

Giải: Có tất cả n! cách bỏ thư. Vấn đề đặt ra là đếm số cách bỏ thư sao cho không lá thư nào đúng địa chỉ. Gọi X là tập hợp tất cả các cách bỏ thư và A_k là tính chất lá thư k bỏ đúng địa chỉ. Khi đó theo nguyên lý bù trừ ta có:

$$\overline{N} = N - N_1 + N_2 - ... + (-1)^n N_n$$

Trong đó . là số cần tìm, N = n!, N_k là số tất cả các cách bỏ thư sao cho có k lá thư đúng địa chỉ. Nhận xét rằng, N_k là mọi cách lấy k lá thư từ n lá, với mỗi cách lấy k lá thư, có (n-k)! cách bỏ để k lá thư này đúng địa chỉ, từ đó ta nhận được.

$$N_k = C(n,k)(n-k)! = \frac{n!}{k!} \text{ và } \overline{N} = n!(1 - \frac{1}{1!} + \frac{1}{2!} - \cdots + \frac{(-1)^n}{n!}$$

Từ đó ta có xác xuất cần tìm là:

$$1-\frac{1}{1!}+\frac{1}{2!}-\cdots+\frac{(-1)^n}{n!}=e^{-1}$$

Số được tính như trên được gọi là số mất thứ tự và được ký hiệu là D_n. Dưới đây là một vài giá trị của D_n, sự tăng nhanh của D_n một lần nữa cho ta thấy rõ sự bùng nổ tổ hợp.

N	2	3	4	5	6	7	8	9	10	11
D_n	1	2	9	44	265	1845	14833	133496	1334961	4890741

2.3. Đếm các hoán vị và tổ hợp

2.3.1. Chỉnh hợp lặp

Định nghĩa 1. Một chỉnh hợp lặp chập k của n phần tử là bộ có thứ tự gồm k thành phần lấy từ n phần tử của tập đã cho.

Như vậy, một chỉnh hợp lặp chập k của n phần tử có thể xem là phần tử của tích đề các A^k với A là tập đã cho. Theo nguyên lý nhân, số các tất cả các chỉnh hợp lặp chập k của n sẽ là n^k.

Ví dụ 1. Tính số hàm từ tập có k phần tử vào tập có n phần tử.

Giải: Biểu diễn mỗi hàm bằng một bộ k thành phần, trong đó thành phần thứ i là ảnh của phần tử thứ i (1<=i<=k). Mỗi thành phần được lấy ra từ một trong n giá trị. Từ đó suy ra số hàm là số bộ k thành phần lấy từ n thành phần bằng n^k.

Ví dụ 2. Từ bảng chữ cái tiếng Anh có thể tạo ra được bao nhiêu xâu có độ dài n.

Giải : Bảng chữ cái tiếng Anh gồm 26 kí tự ['A'..'Z'], số các xâu có độ dài n được chọn từ 26 chữ cái chính là chỉnh hợp lặp n của 26 phần tử và bằng 26^n.

Ví dụ 3. Tính xác xuất lấy ra liên tiếp được 3 quả bóng đỏ ra khỏi bình kín chứa 5 quả đỏ, 7 quả xanh nếu sau mỗi lần lấy một quả bóng ra lại bỏ nó trở lại bình.

Giải: Số kết cục có lợi để ta lấy ra liên tiếp 3 quả bóng đỏ là 5^3 vì có 5 quả đỏ ta phải lấy 3 quả (chú ý vì có hoàn lại). Toàn bộ kết cục có thể để lấy ra ba quả bóng bất kỳ trong 12 quả bóng là 12^3. Như vậy, xác suất để có thể lấy ra 3 quả bóng đỏ liên tiếp là $5^3/12^3$.

2.3.2. Chỉnh hợp không lặp

Định nghĩa 2. Chỉnh hợp không lặp chập k của n phần tử là bộ có thứ tự gồm k thành phần lấy ra từ n phần tử đã cho. Các phần tử không được lặp lại.

Để xây dựng một chỉnh hợp không lặp, ta xây dựng từ thành phần đầu tiên. Thành phần này có n khả năng chọn. Mỗi thành phần tiếp theo những khả năng chọn giảm đi 1 (vì không được lấy lặp lại). Tới thành phần thứ k có n-k + 1 khả năng chọn. Theo nguyên lý nhân ta có số chỉnh hợp lặp k của tập hợp n phần tử ký hiệu là P(n, k) được tính theo công thức:

$$P(n,k)=n(n-1)..(n-k+1)=\frac{n!}{(n-k)!}$$

Ví dụ 1. Tìm số hàm đơn ánh có thể xây dựng được từ tập k phần tử sang tập n phần tử.

Giải: Số hàm đơn ánh từ tập k phần tử sang tập n phần tử chính là P(n,k).

Ví dụ 2. Giả sử có tám vận động viên chạy thi. Người về nhất sẽ được nhận huân chương vàng, người về nhì nhận huân chương bạc, người về ba nhận huy chương đồng. Hỏi có bao nhiêu cách trao huy chương nếu tất cả các kết cục đều có thể xảy ra.

Giải: Số cách trao huy chương chính là số chỉnh hợp chập 3 của tập hợp 8 phần tử. Vì thế có P(8,3) = 8.7.6 = 336 cách trao huy chương.

Ví dụ 3. Có bao nhiêu cách chọn 4 cầu thủ khác nhau trong đội bóng gồm 10 cầu thủ để tham gia các trận đấu đơn.

Giải : Có P(10,4) = 10.9.8.7 = 5040 cách chọn.

2.3.3. Hoán vị

Định nghĩa 3. Ta gọi các hoán vị của n phần tử là một cách xếp có thứ tự các phần tử đó. Số các hoán vị của tập n phần tử có thể coi là trường hợp riêng của chỉnh hợp không lặp với k = n.

Ta cũng có thể đồng nhất một hoán vị với một song ánh từ tập n phần tử lên chính nó. Như vậy, số hoán vị của tập gồm n phần tử là P(n, n) = n!.

Ví dụ 1. Có 6 người xếp thành hàng để chụp ảnh. Hỏi có thể bố trí chụp được bao nhiêu kiểu khác nhau.

Giải: Mỗi kiểu ảnh là một hoán vị của 6 người. Do đó có 6! = 720 kiểu ảnh khác nhau có thể chụp.

Ví dụ 2. Cần bố trí thực hiện n chương trình trên một máy tính. Hỏi có bao nhiêu cách bố trí khác nhau.

Giải: Số chương trình được đánh số từ 1, 2, . ., n. Như vậy, số chương trình cần thực hiện trên một máy tính là số hoán vị của 1, 2, .., n.

Ví dụ 3. Một thương nhân đi bán hàng tại tám thành phố. Chị ta có thể bắt đầu hành trình của mình tại một thành phố nào đó nhưng phải qua 7 thành phố kia theo bất kỳ thứ tự nào mà chị ta muốn. Hỏi có bao nhiêu lộ trình khác nhau mà chị ta có thể đi.

Giải: Vì thành phố xuất phát đã được xác định. Do vậy thương nhân có thể chọn tuỳ ý 7 thành phố còn lại để hành trình. Như vậy, tất cả số hành trình của thương nhân có thể đi qua là 7! = 5040 cách.

2.3.4. Tổ hợp

Định nghĩa 4. Một tổ hợp chập k của n phần tử là một bộ không kể thứ tự gồm k thành phần khác nhau lấy từ n phần tử đã cho. Nói cách khác, ta có thể coi một tổ hợp chập k của n phần tử là một tập con k phần tử lấy trong n phần tử. Số tổ hợp chập k của n phần tử kí hiệu là C(n,k).

Ta có thể tính được trực tiếp số các tổ hợp chập k của tập n phần tử thông qua chỉnh hợp không lặp của k phần tử.

Xét tập hợp tất cả các chỉnh hợp không lặp chập k của n phần tử. Sắp xếp chúng thành những lớp sao cho hai chỉnh hợp thuộc cùng một lớp chỉ khác nhau về thứ tự. Rõ ràng mỗi lớp như vậy là một tổ hợp chập k của n phần tử(P(n,k)). Số chỉnh hợp trong mỗi lớp đều bằng nhau và bằng k! (số hoán vị k phần tử: P(k,k)). Số các lớp bằng số tổ hợp chập k của n (P(n,k)). Từ đó ta có:

$$P(n,k) = C(n,k).P(k,k) \Rightarrow C(n,k) = \frac{P(n,k)}{k!} = \frac{n!}{k!(n-k)!} \quad (1)$$

Ví dụ 1. Cho S = { a, b, c, d } tìm C(4,2).

Giải. Rõ ràng C(4,2) = 6 tương ứng với 6 tập con {a, b}, {a, c}, {a, d}, {b,c}, {b, d} {c,d}.

Ví dụ 2. Có n đội bóng thi đấu vòng tròn. Hỏi phải tổ chức bao nhiêu trận đấu.

Giải: Cứ hai đội bóng thì có một trận. Từ đó suy ra số trận đấu sẽ bằng số cách chọn 2 trong n đội, nghĩa là bằng C(n, 2) = n! / 2!(n-2)! = n(n-1)/2 trận đấu.

Ví dụ 3. Chứng minh

a) C(n,k) = C(n, n-k) (2)
b) C(n, 0) = C(n,n)= 1 (3)
c) C(n,k) = C(n-1,k-1) + C(n-1,k) (4)

Giải a. C(n,n-k) = n!/(n-k)! (n-n+k)! = n!/k!(n-k)! = C(n,k).

Hoặc C(n, k) = n!/k!(n-k)! = n!/ (n-k)! (n-(n-k))! = C(n, n-k);

b) Chú ý 0!=1 => b hiển nhiên đúng

c) C(n,k) = C(n-1,k-1) + C(n-1,k)

$$C(n-1,k-1) + C(n-1,k) = \frac{(n-1)!}{(k-1)!(n-1-k+1)!} + \frac{(n-1)!}{k!(n-k-1)!}$$

$$= \frac{(n-1)!}{(k-1)!(n-k-1)!}\left(\frac{1}{n-k} + \frac{1}{k}\right) = \frac{(n-1)!.n}{(k-1)!k(n-k-1)!(n-k)}$$

$$= \frac{n!}{k!(n-k)!} = C(n.k)$$

Từ những tính chất trên, ta có thể tính tất cả các hệ số tổ hợp chỉ bằng phép cộng. Các hệ số này được tính và viết lần lượt theo dòng, trên mỗi dòng ta tính và thực hiện theo cột. Bảng có dạng tam giác chính là tam giác Pascal.

Các hệ số tổ hợp có liên quan chặt chẽ tới việc khai triển luỹ thừa của một nhị thức. Thực vậy, trong tích

$(x+y)^n = (x+y)(x+y)\ldots(x+y)$ hệ số của $x^k y^{k-n}$ sẽ là số cách chọn k phần tử (x+y) mà từ đó lấy ra x và đồng thời (n-k) nhân tử còn lại lấy ra y, nghĩa là:

$$(x+y)^n = C(n,0)x^n + C(n,1)x^{n-1}y + \ldots + C(n,n-1)xy^{n-1} + C(n,0)y^n = \sum_{k=0}^{n} C(n,k)x^{n-k}y^k$$

(5)

Công thức (5) còn được gọi là khai triển nhị thức Newton, các hệ số tổ hợp còn được gọi là hệ số nhị thức. Chẳng hạn luỹ thừa bậc 8 của nhị thức $(x+y)^8$ được khai triển như sau:

$$(x+y)^8 = x^8 + 8x^7y + 28x^6y^2 + 56x^5y^3 + 70x^4y^4 + 56x^3y^5 + 28x^2y^6 + 8xy^7 + y^8$$

Trong trường hợp y=1, tức khai triển $(x+1)^n$ ta có:

$$(x+1)^n = C(n,0)x^n + C(n,1)x^{n-1} + ... + C(n,n-1)x + C(n,n)$$

Hoặc đẳng thức sau sẽ được rút ra từ khai triển nhị thức Newton:

$$2^n = (1+1)^n = C(n,0) + C(n,1) + ... + C(n,n-1) + C(n,n)$$

Có thể nói rất nhiều đẳng thức về hệ số tổ hợp sẽ được suy ra. Như tính các tập lẻ, đạo hàm . . .

CHƯƠNG 3. CÁC MÔ HÌNH THUẬT TOÁN CƠ BẢN

Nội dung chính của chương trình bày một số lược đồ thuật toán cơ bản dùng để giải lớp các bài toán liệt kê, bài toán đếm, và bài toán tối ưu và bài toán tồn tại. Mỗi lược đồ thuật toán giải quyết một lớp các bài toán thỏa mãn một số tính chất nào đó. Đây là những lược đồ thuật toán quan trọng nhằm giúp người học vận dụng nó trong khi giải quyết các vấn đề trong tin học. Các lược đồ thuật toán được trình bày trong chương này bao gồm: thuật toán sinh, thuật toán đệ qui, thuật toán quay lui, thuật toán tham lam, thuật toán nhánh cận, thuật toán qui hoạch động.

3.1. Mô hình thuật toán sinh (Generative Algorithm)

Mô hình thuật toán sinh được dùng để giải lớp các bài toán liệt kê, bài toán đếm, bài toán tối ưu, bài toán tồn tại thỏa mãn hai điều kiện:

- **Điều kiện 1:** *Có thể xác định được một thứ tự trên tập các cấu hình cần liệt kê của bài toán. Biết cấu hình đầu tiên, biết cấu hình cuối cùng.*
- **Điều kiện 2:** *Từ một cấu hình chưa phải cuối cùng, ta xây dựng được thuật toán sinh ra cấu hình đứng ngay sau nó.*

Mô hình thuật toán sinh được biểu diễn thành hai bước: bước khởi tạo và bước lặp. Tại bước khởi tạo, cấu hình đầu tiên của bài toán sẽ được thiết lập. Điều này bao giờ cũng thực hiện được theo giả thiết của bài toán. Tại bước lặp, quá trình lặp được thực hiện khi gặp phải cấu hình cuối cùng. Điều kiện lặp của bài toán bao giờ cũng tồn tại theo giả thiết của bài toán. Hai chỉ thị cần thực hiện trong thân vòng lặp là đưa ra cấu hình hiện tại và sinh ra cấu hình kế tiếp. Mô hình sinh kế tiếp được thực hiện tùy thuộc vào mỗi bài toán cụ thể. Tổng quát, mô hình thuật toán sinh được thể hiện như dưới đây.

```
Thuật toán Generation;
begin
    Bước1 (Khởi tạo):
        <Thiết lập cấu hình đầu tiên>;
    Bước 2 (Bước lặp):
        while (<Lặp khi cấu hình chưa phải cuối cùng>) do
            <Đưa ra cấu hình hiện tại>;
            <Sinh ra cấu hình kế tiếp>;
    endwhile;
End.
```

Ví dụ 3.1. Vector $X = (x_1, x_2, .., x_n)$, trong đó $x_i = 0, 1$ được gọi là một xâu nhị phân có độ dài *n*. Hãy liệt kê các xâu nhị phân có độ dài *n*. Ví dụ với n=4, ta sẽ liệt kê được 24 xâu nhị phân độ dài 4 như trong Bảng 2.1.

Bảng 2.1. Các xâu nhị phân độ dài 4

STT	$X=(x_1, x_2, x_3, x_4)$	STT	$X=(x_1, x_2, x_3, x_4)$

0	0 0 0 0	**8**	1 0 0 0
1	0 0 0 1	**9**	1 0 0 1
2	0 0 1 0	**10**	1 0 1 0
3	0 0 1 1	**11**	1 0 1 1
4	0 1 0 0	**12**	1 1 0 0
5	0 1 0 1	**13**	1 1 0 1
6	0 1 1 0	**14**	1 1 1 0
7	0 1 1 1	**15**	1 1 1 1

Lời giải:

Điều kiện 1: Gọi thứ tự của xâu nhị phân $X=(x_1, x_2,.., x_n)$ là $f(X)$. Trong đó, $f(X)= k$ là số chuyển đổi xâu nhị X thành số ở hệ cơ số 10. Ví dụ, xâu X = (1, 0, 1, 1) được chuyển thành số hệ cơ số 10 là 11 thì ta nói xâu X có thứ tự 11. Với cách quan niệm này, xâu đứng sau xâu có thứ tự 11 là 12 chính là xâu đứng ngay sau xâu X = (1, 0, 1, 1). Xâu đầu tiên có thứ tự là *0* ứng với xâu có *n* số *0*. Xâu cuối cùng có thứ tự là 2^n-1 ứng với xâu có *n* số *1*. Như vậy, điều kiện 1 của thuật toán sinh đã được thỏa mãn.

Điều kiện 2: Về nguyên tắc ta có thể lấy $k = f(X)$ là thứ tự của một xâu bất kỳ theo nguyên tắc ở trên, sau đó lấy thứ tự của xâu kế tiếp là $(k + 1)$ và chuyển đổi $(k+1)$ thành số ở hệ cơ số 10 ta sẽ được xâu nhị phân tiếp theo. Xâu cuối cùng sẽ là xâu có n số 1 ứng với thứ tự $k = 2^n-1$. Với cách làm này, ta có thể coi mỗi xâu nhị phân là một số, mỗi thành phần của xâu là một bít và chỉ cần cài đặt thuật toán chuyển đổi cơ số ở hệ 10 thành số ở hệ nhị phân. Ta có thể xây dựng thuật toán tổng quát hơn bằng cách xem mỗi xâu nhị phân là một mảng các phần tử có giá trị 0 hoặc 1. Sau đó, duyệt từ vị trí bên phải nhất của xâu nếu gặp số 1 ta chuyển thành 0 và gặp số 0 đầu tiên ta chuyển thành 1. Ví dụ với xâu X = (0, 1, 1, 1) được chuyển thành xâu X= (1, 0, 0, 0), xâu X = (1,0,0,0) được chuyển thành xâu X =(1, 0, 0, 1). Lời giải và thuật toán sinh xâu nhị phân kế tiếp được thể hiện trong chương trình dưới đây. Trong đó, thuật toán sinh xâu nhị phân kế tiếp từ một xâu nhị phân bất kỳ là hàm *Next_Bits_String*().

```
#include <iostream>
#include <iomanip>
#define MAX 100
using namespace std;
int X[MAX], n, dem = 0; //sử dụng các biến toàn cục X[], n, OK, dem
bool OK =true;
void Init(void){ //khởi tạo xâu nhị phân đầu tiên
        cout<<"Nhập n="; cin>>n;
        for(int i = 1; i<=n; i++) //thiết lập xâu với n số 0
                X[i]=0;
}
void Result(void){ //đưa ra xâu nhị phân hiện tại
```

```
        cout<<"\n Xâu thứ "<<++dem<<":";
        for(int i=1; i<=n; i++)
           cout<<X[i]<<setw(3);
}
void Next_Bits_String(void){ //thuật toán sinh xâu nhị phân kế tiếp
        int i=n;
        while(i>0 && X[i]){ //duyệt từ vị trí bên phải nhất
                X[i]=0; //nếu gặp X[i] = 1 ta chuyển thành 0
                i--; //lùi lại vị trí sau
        }
        if (i>0)  X[i]=1; //gặp X[i] =0 đầu tiên ta chuyển thành 1
        else OK = false; //kết thúc khi gặp xâu có n số 1
}
int main(void){ //đây là thuật toán sinh
        Init(); //thiết lập cấu hình đầu tiên
        while(OK){//lặp khi chưa phải cấu hình cuối cùng
                Result(); //đưa ra cấu hình hiện tại
                Next_Bits_String(); //sinh ra cấu hình kế tiếp
        }
}
```

Ví dụ 3.2. Liệt kê các tập con k phần tử của 1, 2, .., n.

Lời giải. Mỗi tập con k phần tử của 1, 2, .., N là một tổ hợp chập K của 1, 2,.., N. Ví dụ với n=5, k= 3 ta sẽ có C(n,k) tập con trong Bảng 2.2.

Điều kiện 1. Ta gọi tập con X =(x_1,..x_K) là đứng trước tập con Y =(y_1, y_2,..y_K) nếu tìm được chỉ số t sao cho $x_1 = y_1$, $x_2 = y_2$,.., $x_{t-1} = y_{t-1}$, $x_t < y_t$. Ví dụ tập con X = (*1, 2, 3*) đứng trước tập con Y =(*1, 2, 4*) vì ta tìm được t=3 thỏa mãn $x_1 = y_1$, $x_2 = y_2$, $x_3 < y_3$. Tập con đầu tiên là X = (1, *2,.., k*), tập con cuối cùng là (*n–k+1,.., N*). Như vậy điều kiện 1 của thuật toán sinh được thỏa mãn.

Điều kiện 2. Để ý rằng, tập con cuối cùng (*n-k+1,…, n*) luôn thỏa mãn đẳng thức $X[i] = n – k + i$. Ví dụ tập con cuối cùng $X[]$ = (3, 4, 5) ta đều có: $X[1] = 3 = 5 – 3 + 1$; $X[2] = 4 = 5 – 3 + 2$; $X[3] = 5 = 5 – 3 + 3$. Để tìm tập con kế tiếp từ tập con bất kỳ ta chỉ cần duyệt từ phải qua trái tập con $X[]$ = (x_1, x_2, .., x_k) để xác định chỉ số i thỏa mãn điều kiện $X[i] \neq n – k + i$. Ví dụ với $X[]$ = (*1, 4, 5*), ta xác định được $i=1$ vì $X[3] = 5 = 5-3+3$, $X[2] = 4 = 5-3+2$, và $X[1] = 1 \neq 5-3+1$. Sau khi xác định được chỉ số i, tập con mới sẽ được sinh là $Y[]$ = (y_1,.., y_i, …, y_k) ra thỏa mãn điều kiện: $y_1 = x_1$, $y_2 = x_2$,.., $y_{i-1} = x_{i-1}$, $y_i = x_i+1$, và $y_j = x_t + j – i$ với ($j = j+1$, …, k).

Bảng 2.2. Tập con 3 phần tử của 1, 2, 3, 4, 5

STT	Tập con
1	1 2 3

2	1 2 4
3	1 2 5
4	1 3 4
5	1 3 5
6	1 4 5
7	2 3 4
8	2 3 5
9	2 4 5
10	3 4 5

Chương trình cài đặt thuật toán sinh tập con k phần tử được thể hiện như dưới đây. Trong đó, thuật toán sinh tổ hợp kế tiếp có tên là Next_Combination().

```
#include <iostream>
#include <iomanip>
#define MAX 100
int X[MAX], n, k, dem=0;
bool OK = true;
using namespace std;
void Init(void){ //thiết lập tập con đầu tiên
        cout<<"\n Nhập n, k:"; cin>>n>>k;
        for(int i=1; i<=k; i++)  //tập con đầu tiên là 1, 2, .., k
                X[i] = i;
}



void Result(void){ //đưa ra tập con hiện tại
        cout<<"\n Kết quả "<<++dem<<":";
                for(int i=1; i<=k; i++) //đưa ra X[] =( x1, x2, .., xk)
                cout<<X[i]<<setw(3);
}
void Next_Combination(void){ //sinh tập con k phần tử từ tập con bất kỳ
        int i = k; //duyệt từ vị trí bên phải nhất của tập con
        while(i>0 && X[i]== n-k+i) //tìm i sao cho xi ≠ n-k+i
                i--;
        if (i>0){//nếu chưa phải là tập con cuối cùng
                X[i]= X[i]+1; //thay đổi giá trị tại vị  trí i: xi = xi +1;
                for(int j=i+1; j<=k; j++) //các vị trí j từ i+1,.., k
                        X[j] = X[i] + j - i; // được thay đổi là xj = xi +j - i;
        }
```

```
        else  //nếu là tập con cuối cùng
                OK = false; //ta kết thúc duyệt
}
int main(void){
        Init(); //khởi tạo cấu hình đầu tiên
        while(OK){ //lặp trong khi cấu hình chưa phải cuối cùng
                Result(); //đưa ra cấu hình hiện tại
                Next_Combination(); //sinh ra cấu hình kế tiếp
        }
}
```

Ví dụ 3.3. Liệt kê các hoán vị của 1, 2, .., n.

Lời giải. Mỗi hoán vị của 1, 2, .., N là một cách xếp có tính đến thứ tự của 1, 2,..,N. Số các hoán vị là N!. Ví dụ với N =3 ta có 6 hoán vị dưới đây.

Bảng 2.3. Hoán vị của 1, 2, 3.

STT	Hoán vị
1	1 2 3
2	1 3 2
3	2 1 3
4	2 3 1
5	3 1 2
6	3 2 1

Điều kiện 1. Có thể xác định được nhiều trật tự khác nhau trên các hoán vị. Tuy nhiên, thứ tự đơn giản nhất có thể được xác định như sau. Hoán vị $X =(x_1, x_2,.., x_n)$ được gọi là đứng sau hoán vị $Y = (y_1, y_2,..,y_n)$ nếu tồn tại chỉ số k sao cho $x_1 = y_1$, $x_2 = y_2$,…, $x_{k-1} =y_{k-1}$, $x_k<y_k$. Ví dụ hoán vị X = (1, 2, 3) được gọi là đứng sau hoán vị Y =(1, 3, 2) vì tồn tại k =2 để $x_1 = y_1$, và $x_2<y_2$. Hoán vị đầu tiên là X[] = (1, 2, …, n), hoán vị cuối cùng là X[] = (n, n-1, …, 1).

Điều kiện 2. Được thể hiện thông qua hàm Next_Permutation() như chương trình dưới đây.

```
#include <iostream>
#include <iomanip>
#define MAX 100
int X[MAX], n,  dem=0;
bool OK = true;
using namespace std;
void Init(void){ //thiết lập hoán vị đầu tiên
        cout<<"\n Nhap n:"; cin>>n;
        for(int i=1; i<=n; i++) //thiết lập X[] = (1, 2, ..,n)
                X[i] = i;
```

```
}
void Result(void){ //đưa ra hoán vị hiện tại
        cout<<"\n Kết quả "<<++dem<<":";
        for(int i=1; i<=n; i++)
           cout<<X[i]<<setw(3);
}
void Next_Permutation(void){ //sinh ra hoán vị kế tiếp
        int j = n-1; //xuất phát từ vị trí j = n-1
        while(j>0 && X[j]>X[j+1]) //tìm chỉ số j sao cho X[j] < X[j+1]
                j--;
        if ( j > 0){ // nếu chưa phải hoán vị cuối cùng
                int k = n; //xuất phát từ vị trí k = n
                while(X[j]>X[k]) //tìm chỉ số k sao cho X[j] < X[k]
                        k--;
                int t = X[j]; X[j] = X[k]; X[k]=t; //đổi chỗ X[j] cho X[k]
                int r = j+1, s = n;
                while (r<=s){ //lật ngược lại đoạn từ j+1,..,n
                        t=X[r]; X[r]=X[s]; X[s]=t;
                        r++; s--;
                }
        }
        else  //nếu là cấu hình cuối cùng
                OK = false; //ta kết thúc duyệt
}
int main(void){ //đây là thuật toán sinh
        Init(); //thiết lập cấu hình đầu tiên
        while(OK){ //lặp trong khi cấu hình chưa phải cuối cùng
                Result(); //đưa ra cấu hình hiện tại
                Next_Permutation(); //sinh ra cấu hình kế tiếp
        }
}
```

3.2. Mô hình thuật toán đệ qui (Recursion Algorithm)

Một đối tượng được định nghĩa trực tiếp hoặc gián tiếp thông qua chính nó được gọi là phép định nghĩa bằng đệ qui. Thuật toán giải bài toán *P* một cách trực tiếp hoặc gián tiếp thông qua bài toán *P*' giống như *P* được gọi là thuật toán đệ qui giải bài toán *P*. Một hàm được gọi là đệ qui nếu nó được gọi trực tiếp hoặc gián tiếp đến chính nó.

Tổng quát, một bài toán có thể giải được bằng đệ qui nếu nó thỏa mãn hai điều kiện:

- **Phân tích được**: *Có thể giải được bài toán P bằng bài toán P' giống như P. Bài tóa P' và chỉ khác P ở dữ liệu đầu vào. Việc giải bài toán P' cũng được thực hiện theo cách phân tích giống như P.*
- **Điều kiện dừng**: *Dãy các bài toán P' giống như P là hữu hạn và sẽ dừng tại một bài toán xác định nào đó.*

Thuật toán đệ qui tổng quát có thể được mô tả như sau:

Thuật toán Recursion (P) {

1. *Nếu P thỏa mãn điều kiện dừng:*
   <Giải P với điều kiện dừng>;
2. Nếu *P* không thỏa mãn điều kiện dừng:
   <Giải P' giống như P:Recursion(P')>;

}

Ví dụ 3.4. Tìm tổng của n số tự nhiên đầu tiên bằng phương pháp đệ qui.

Lời giải. Gọi S_n là tổng của n số tự nhiên. Khi đó:

- **Bước phân tích**: dễ dàng phận thấy tổng n số tự nhiên $S_n = n + S_{n-1}$, với $n \geq 1$.
- **Điều kiện dừng**: $S_0 = 0$ nếu n = 0;

Từ đó ta có lời giải của bài toán như sau:

```
int     Tong (int  n ) {
        if (n ==0 ) return(0); //Điều kiện dừng
        else return(n + Tong(n-1)); //Điều kiện phân tích được
}
```

Chẳng hạn ta cần tìm tổng của 5 số tự nhiên đầu tiên, khi đó:

```
S  = Tong(5)
   = 5 + Tong(4)
   = 5 + 4 + Tong(3)
   = 5 + 4 + 3 + Tong(2)
   = 5 + 4 + 3 + 2+ Tong(1)
   = 5 + 4 + 3 + 2 + 1 + Tong(0)
   = 5 + 4 + 3 + 2 + 1 + 0
   = 15
```

Ví dụ 3.5. Tìm n!.

Lời giải. Gọi Sn là n!. Khi đó:

- **Bước phân tích**: Sn = n*(n-1)! nếu n>0;
- **Điều kiện dừng**: s0=1 nếu n=0.

Từ đó ta có lời giải của bài toán như sau:

```
long    Giaithua (int  n ) {
        if (n ==0 ) return(1); //Điều kiện dừng
        else return(n *Giaithua(n-1)); //Điều kiện phân tích được
}
```

Ví dụ 3.6. Tìm ước số chung lớn nhất của a và b bằng phương pháp đệ qui.
Lời giải. Gọi d =USCLN(a,b). Khi đó:

- **Bước phân tích**: nếu b≠0 thì d = USCLN(a, b) = USCLN(b, r), trong đó a =b, b = r = a mod b .
- **Điều kiện dừng**: nếu b = 0 thì a là ước số chung lớn nhất của a và b.

Từ đó ta có lời giải của bài toán như sau:

```
int     USCLN (int a, int b ) {
        if (a ==b ) return(a); //Điều kiện dừng
        else { //Điều kiện phân tích được
                int r = a % b; a = b; b = r;
                return(USCLN(a, b)); //giải bài toán USCLN(a, b)
        }
}
```

3.3. Mô hình thuật toán quay lui (Back-track Algorithm)

Giả sử ta cần xác định bộ $X = (x_1, x_2,..,x_n)$ thỏa mãn một số ràng buộc nào đó. Ứng với mỗi thành phần x_i ta có n_i khả năng cần lựa chọn. Ứng với mỗi khả năng $j \in n_i$ dành cho thành phần x_i ta cần thực hiện:

- *Kiểm tra xem khả năng j có được chấp thuận cho thành phần x_i hay không? Nếu khả năng j được chấp thuận thì ta xác định thành phần x_i theo khả năng j. Nếu i là thành phần cuối cùng (i=n) ta ghi nhận nghiệm của bài toán. Nếu i chưa phải cuối cùng ta xác định thành phần thứ i +1.*
- *Nếu không có khả năng j nào được chấp thuận cho thành phần x_i thì ta quay lại bước trước đó (i-1) để thử lại các khả năng còn lại.*

Thuật toán quay lui được mô tả như sau:

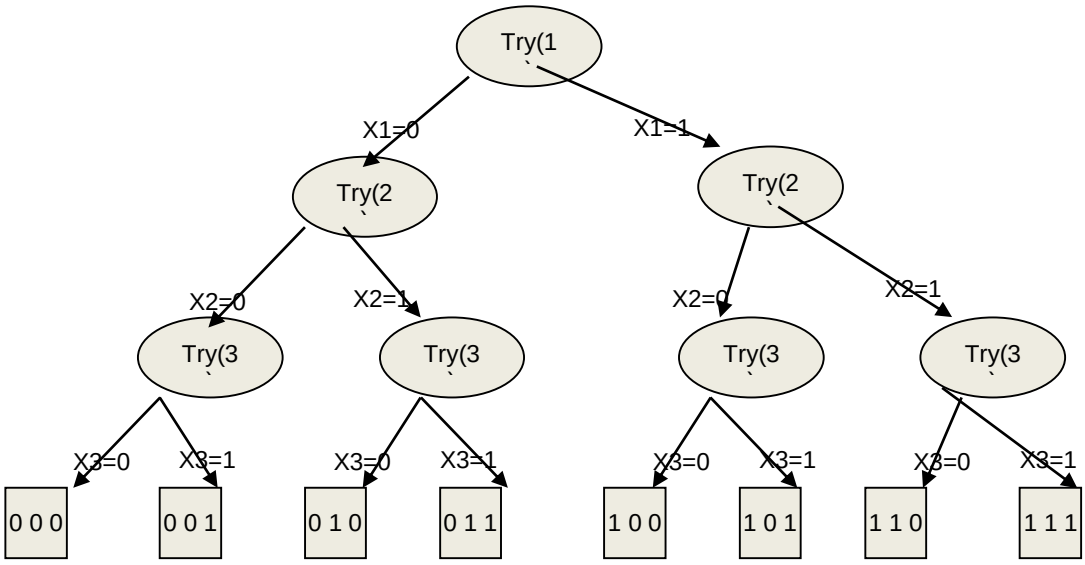
```
        Thuật toán Back-Track ( int i ) {
                for ( j =<Khả năng 1>; j <=ni;  j++ ){
                        if (<chấp thuận khả năng j>) {
                                X[i] = <khả năng j>;
                                if ( i ==n) Result();
                                else Back-Track(i+1);
                        }
        }
```

Ví dụ 2.7. Duyệt các xâu nhị phân có độ dài n.
Lời giải. Xâu nhị phân $X = (x_1, x_2,..,x_n)| x_i = 0, 1$. Mỗi $x_i \in X$ có hai lựa chọn $x_i=0, 1$. Cả hai giá trị này đều được chấp thuận mà không cần có thêm bất kỳ điều kiện gì. Thuật toán được mô tả như sau:

```
                void Try ( int i ) {
```

```
        for (int j =0; j<=1; j++){
            X[i] = j;
            if ( i ==n) Result();
            else Try (i+1);
        }
    }
```

Khi đó, việc duyệt các xâu nhị phân có độ dài n ta chỉ cần gọi đến thủ tục Try(1). Cây quay lui được mô tả như Hình 2.1 dưới đây.



Hình 3.1. *Duyệt các xâu nhị phân độ dài 3*

Chương trình duyệt các xâu nhị phân có độ dài n bằng thuật toán quay lui được thể hiện như dưới đây.

```
#include <iostream>
#include <iomanip>
#define MAX 100
using namespace std;
int X[MAX], n, dem=0;
void Init(){ //thiết lập độ dài xâu nhị phân
    cout<<"\n Nhập n="; cin>>n;
}
void Result(void){ //In ra xâu nhị phân X[] = x1, x2,.., xn
    cout<<"\n Kết quả "<<++dem<<":";
    for(int i =1; i<=n; i++)
        cout<<X[i]<<setw(3);
    }
void Try(int i){ //thuật toán quay lui
    for (int j=0; j<=1; j++){ //duyệt các khả năng j dành cho xi
        X[i]=j; //thiết lập thành phần xi là j
```

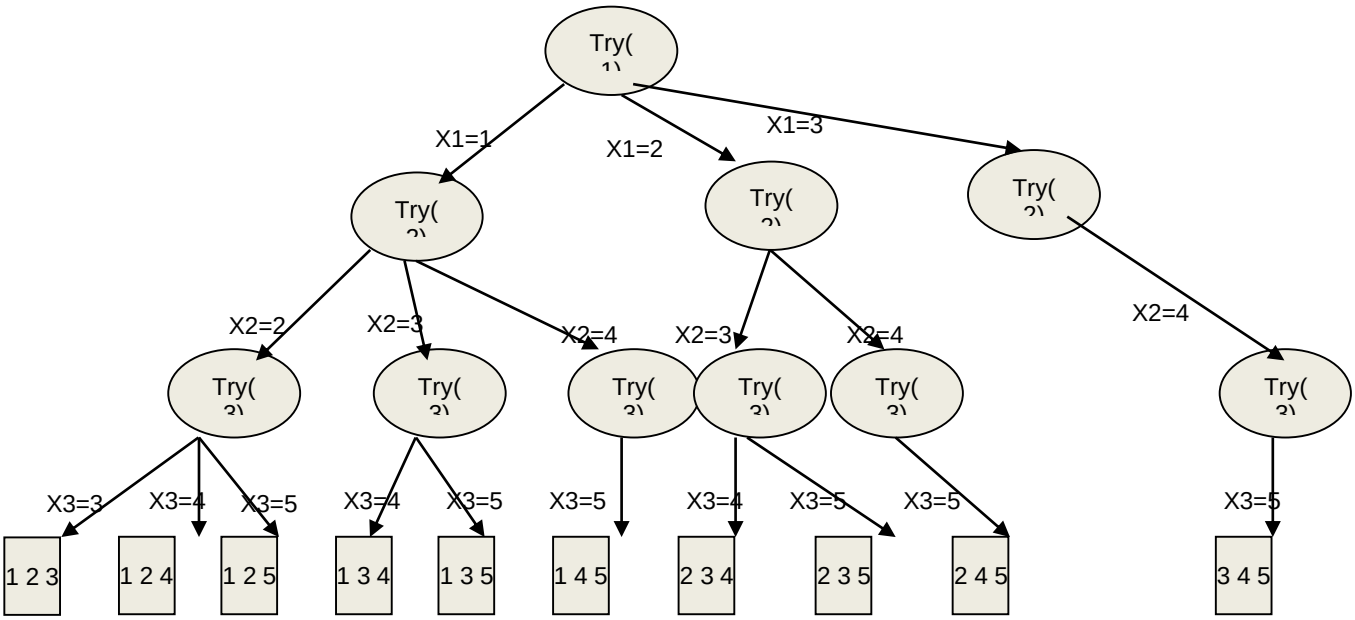
```
            if(i==n) //nếu i là thành phần cuối cùng
                Result(); // ta đưa ra kết quả
            else //trong trường hợp khác
                Try(i+1); //ta xác định tiếp thành phần x_{i+1}
        }
    }
    int main(void){     Init(); Try(1);}
```

Ví dụ 3.8. Duyệt các tập con K phần tử của 1, 2, .., N.

Lời giải. Mỗi tập con K phần tử $X = (x_1, x_2,..,x_K)$ là bộ không tính đến thứ tự K phần tử của 1, 2, .., N. Mỗi $x_i \in X$ có N-K+i lựa chọn. Các giá trị này đều được chấp thuận mà không cần có thêm bất kỳ điều kiện gì. Thuật toán được mô tả như sau:

```
    void Try ( int i ) {
        for (int j =X[i-1]+1; j<=N-K+ i; j++){
            X[i] = j;
            if ( i ==K) Result();
            else Try (i+1);
        }
    }
```

Khi đó, việc duyệt các tập con K phần tử của 1, 2, .., N ta chỉ cần gọi đến thủ tục Try(1). Cây quay lui được mô tả như hình dưới đây.



Hình 3.2. *Duyệt các tập con 3 phần tử của 1, 2, 3, 4, 5.*

Chương trình liệt kê các tập con k phần tử của 1, 2, ..,n được thể hiện như sau.

```
#include <iostream>
#include <iomanip>
#define MAX 100
using namespace std;
int X[MAX], n, k, dem=0;
void Init(){//thiết lập giá trị cho n, k
        cout<<"\n Nhập n, k: "; cin>>n>>k;
}
void Result(void){   cout<<"\n Kết quả "<<++dem<<":";//đưa ra kết quả
        for(int i =1; i<=k; i++)    cout<<X[i]<<setw(3);
}

void Try(int i){//thuật toán quay lui
        for (int j=X[i-1]+1; j<=n-k+i; j++){ //duyệt trên tập khả năng dành cho xi
                X[i]=j; //thiết lập thành phần xi là j
                if(i==k) //nếu xi đã là thành phần cuối
                        Result(); //ta đưa ra kết quả
                else //trong trường hợp khác
                        Try(i+1); //ta đi xác định thành phần thứ xi+1
        }
}

int main(void){
        Init();
        X[0] =0 ;
        Try(1);
}
```

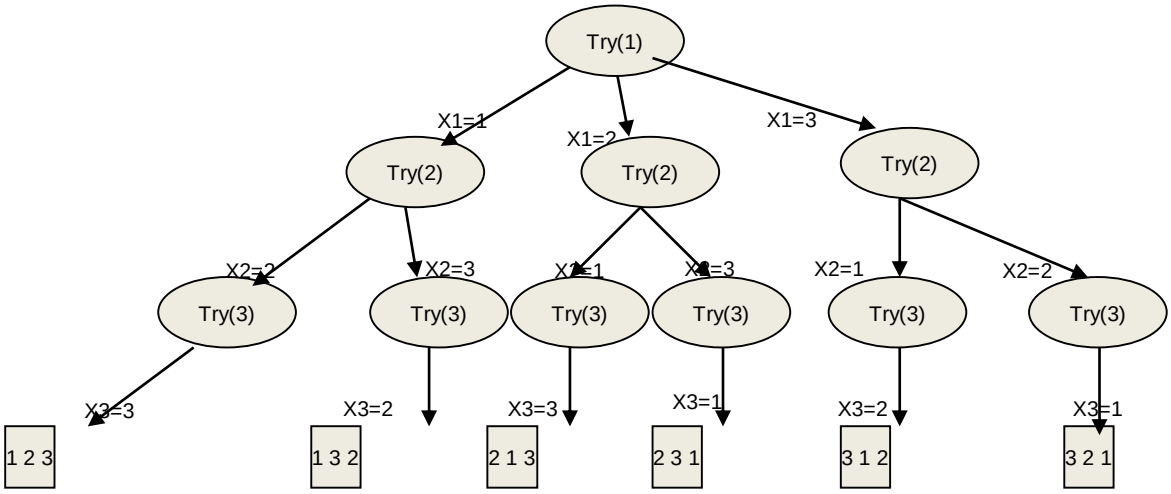
Ví dụ 3.9. Duyệt các hoán vị của 1, 2, .., N.

Lời giải. Mỗi hoán vị $X = (x_1, x_2,..,x_K)$ là bộ có tính đến thứ tự của 1, 2, .., N. Mỗi $x_i \in X$ có N lựa chọn. Khi $x_i = j$ được lựa chọn thì giá trị này sẽ không được chấp thuận cho các thành phần còn lại. Để ghi nhận điều này, ta sử dụng mảng chuaxet[] gồm N phần

tử. Nếu chuaxet[i] = True điều đó có nghĩa giá trị i được chấp thuận và chuaxet[i] = False tương ứng với giá trị i không được phép sử dụng. Thuật toán được mô tả như sau:

```
void Try ( int i ) {
        for (int j =1; j<=N; j++){
                if (chuaxet[j] ) {
                        X[i] = j;chuaxet[j] = False;
                        if ( i ==N) Result();
                        else Try (i+1);
                        Chuaxet[j] = True;
                }
        }
}
```

Khi đó, việc duyệt các hoán vị của 1, 2, .., N ta chỉ cần gọi đến thủ tục Try(1). Cây quay lui được mô tả như hình dưới đây.



Hình 2.3. Duyệt các hoán vị của 1, 2, 3.

Chương trình liệt kê tất cả các hoán vị của 1, 2, .., n được thể hiện như sau:

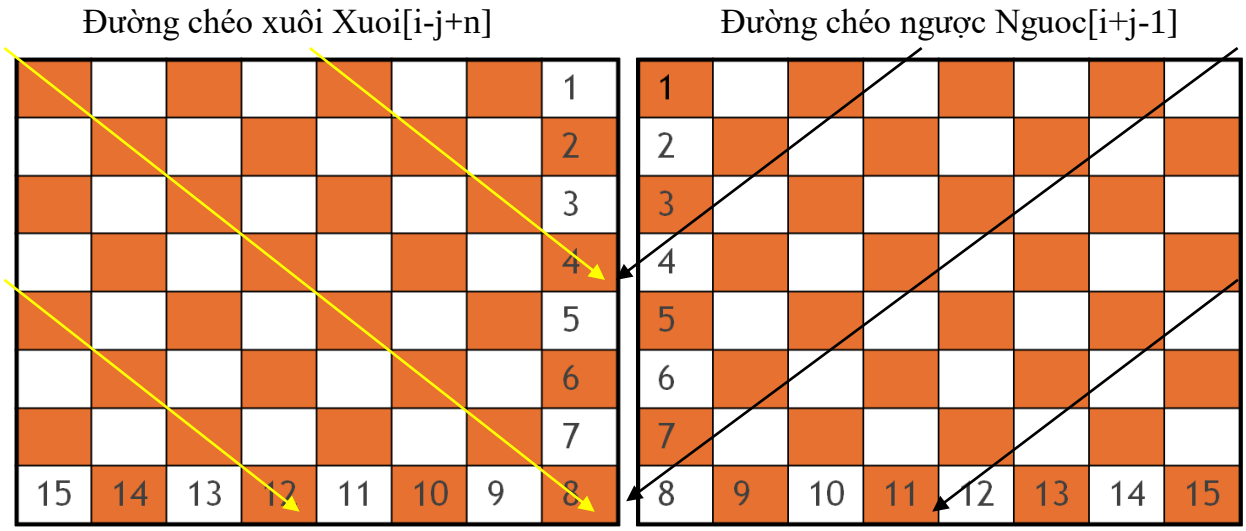
```
#include <iostream>
#include <iomanip>
#define MAX 100
using namespace std;
int X[MAX], n, dem=0;
bool  chuaxet[MAX];
void Init(){//thiết lập giá trị cho n
        cout<<"\n Nhập n="; cin>>n;
        for(int i=1; i<=n; i++) //thiết lập giá trị cho mảng chuaxet[]
                chuaxet[i]=true;
}
void Result(void){ //Đưa ra hoán vị hiện tại
        cout<<"\n Kết quả "<<++dem<<":";
```

```
        for(int i =1; i<=n; i++) cout<<X[i]<<setw(3);
}
void Try(int i){ //thuật toán quay lui duyệt các hoán vị của 1, 2, .., n.
        for (int j=1; j<=n; j++){ //duyệt các khả năng j cho thành phần xi
                if(chuaxet[j]){ //nếu khả năng j đúng chưa được dùng đến
                        X[i]=j; //thiết lập  thành phần xi là j
                        chuaxet[j]=false; //thiết lập chuaxet[j] đã được dùng
                        if(i==n)  //nếu xi đã là thành phần cuối cùng
                                Result();//ta đưa ra kết quả
                        else ///trong trường hợp khác
                                Try(i+1); //ta xác định tiếp thành phần thứ i+1
                        chuaxet[j]=true; //nhớ hoàn trả lại giá trị cho chuaxet[j]
                        }
        }
}
int main(void){
        Init(); Try(1);
}
```

Ví dụ 3.10. Bài toán N quân hậu. Trên bàn cờ kích cỡ N×N, hãy đặt N quân hậu mỗi quân trên 1 hàng sao cho tất cả các quân hậu đều không ăn được lẫn nhau.

Lời giải. Gọi X =(x_1, x_2,..,x_n) là một hoán vị của 1, 2, .., n.. Khi đó, x_i = j được hiểu là quân hậu hàng thứ i đặt ở cột j. Để các quân hậu khác không thể ăn được, quân hậu thứ i cần không được lấy trùng với bất kỳ cột nào, không được cùng đường chéo xuôi, không được cùng trên đường chéo ngược. Ta có n cột Cot = (c_1,..c_n), có Xuoi[2*n-1] đường chéo xuôi, Nguoc[2*n-1] đường chéo ngược. Quân hậu ở hàng *i* được đặt vào cột *j* nếu A[j] = True (chưa có quân hậu nào án ngữ cột j), Xuoi[i-j+n] = True (chưa có quân hậu

nào án ngữ đường chéo i-j+n), Nguoc[i + j -1] = True (chưa có quân hậu nào án ngữ đường chéo ngược i + j-1).



Hình 3.4. *Mô tả các đường chéo, xuôi đường chép ngược*

Thuật toán quay lui giải bài toán n quân hậu được mô tả như dưới đây.

```
void Try (int    i){
        for(int j=1; j<=n; j++){
                if( Cot[j] && Xuoi[ i – j + n ] && Nguoc[i + j -1]){
                        X[i] =j;  Cot[j]=FALSE;
                        Xuoi[ i - j + n]=FALSE;
                        Nguoc[ i + j - 1]=FALSE;
                        if(i==n) Result();
                        else Try(i+1);
                        Cot[j] = TRUE;
                        Xuoi[ i - j + n] = TRUE;
                        Nguoc[ i + j - 1] = TRUE;
                }
        }
}
```

3.4. Mô hình thuật toán tham lam (Greedy Algorithm)

Thuật toán tham lam (Greedy Algorithm) xây dựng một chiến lược "*tham từng miếng*", miếng dễ tham nhất thường là một giải pháp tối ưu cục bộ nhưng lại luôn có mặt trong cấu trúc tối ưu toàn cục. Tiếp tục phát triển chiến lược "*tham từng miếng*" cho các bước tiếp theo để đạt được kết quả tối ưu toàn cục. Tóm lại, thuật toán tham lam dùng để giải lớp các bai toán tối ưu thỏa mãn hai điều kiện:

- Ở mỗi bước ta luôn tạo ra một lựa chọn tốt nhất tại thời điểm đó.
- Việc liên kết lại kết quả mỗi bước sẽ cho ta kết quả tối ưu toàn cục.

Tổng quát, thuật toán Greedy bao gồm 5 thành phần chính:

1) Một tập các ứng viên (*candidate members*) mà giải pháp có thể tham lam.

2) Một hàm lựa chọn (*selection fuction*) để chọn ứng viên tốt nhất cho giải pháp tham lam cục bộ.
3) Một hàm thực thi (*feasibility function*) được sử dụng để quyết định xem một ứng viên có được dùng để xây dựng lời giải hay không?
4) Một hàm mục tiêu (objective function) dùng để xác định giá trị của lời giải hoặc một phần của lời giải.
5) Một hàm giải pháp (solution function) dùng để xác định khi nào giải pháp hoàn chỉnh.

Khi sử dụng giải pháp tham lam, hai thành phần quyết định nhất tới quyết định tham lam đó là:

Lựa chọn tính chất để tham. Khi chưa tìm được lời giải tối ưu toàn cục nhưng ta biết chắc chắn một giải pháp tối ưu cục bộ dựa vào tính chất cụ thể của mỗi bài toán. Bài toán con tiếp theo cũng thực hiện với tính chất như vậy cho đến khi đạt được giải pháp tối ưu toàn cục.

Lựa chọn cấu trúc con tối ưu. Một bài toán được gọi là "có cấu trúc tối ưu" nếu một lời giải tối ưu của bài toán con nằm trong lời giải tối ưu của bài toán lớn hơn. Điều này cho phép ta xác định từng cấu trúc con tối ưu cho mỗi bài toán con, sau đó phát triển các bài toán con cho đến bài toán con cuối cùng.

Ví dụ 2.11. Bài toán lựa chọn hành động (*Activity Selection Problem*). Lựa chọn hành động là bài toán tối ưu tổ hợp điển hình quan tâm đến việc lựa chọn các hành động không mâu thuẫn nhau trong các khung thời gian cho trước. Bài toán được phát biểu như sau:

Cho tập gồm n hành động, mỗi hành động được biểu diễn như bộ đôi thời gian bắt đầu s_i và thời gian kết thúc f_i (i=1, 2, .., n). Bài toán đặt ra là hãy lựa chọn nhiều nhất các hành động có thể thực hiện bởi một máy hoặc một người mà không xảy ra mâu thuẫn. Giả sử mỗi hành động chỉ thực hiện đơn lẻ tại một thời điểm.

Input:

- Số lượng hành động: 6
- Thời gian bắt đầu Start []= { 1, 3, 0, 5, 8, 5}
- Thời gian kết thúc Finish[]= { 2, 4, 6, 7, 9, 9}

Output: Số lượng lớn nhất các hành động có thể thực hiện bởi một người.

OPT[] = {1, 2, 4, 5 }

Lời giải. Rõ ràng, hệ sẽ xảy ra mâu thuẫn khi tồn tại hai hành động kế tiếp nhau *i* và *i+1* có thời gian kết thúc của hành động *i* lớn hơn thời gian bắt đầu của hành động *i+1*. Nói cách khác hệ mâu thuẫn khi Finish[i]>Start[i+1] (1<i<n). Vấn đề còn lại là làm thế nào ta có thể lựa chọn được tập lớn nhất các hành động không tồn tại mâu thuẫn. Để giải quyết điều này, ta có thể sử dụng thuật toán sinh hoặc thuật toán quay lui thể duyệt trên tập tất cả các tập con các hành động, sau đó lựa chọn tập con có số lượng hành động lớn nhất và không xảy ra mâu thuẫn. Tuy nhiên lời giải này cho ta độ phức tạp hàm mũ (2^n) nên lời giải này thực tế không giải được bằng máy tính.

Để khắc phục điều này ta có thể xây dựng thuật toán tham lam một cách hiệu quả dựa vào những nhận xét trực quan như sau:

- Trường hợp xấu nhất xảy ra khi hệ chỉ có thể đáp ứng được nhiều nhất là một hành động, hễ cứ thêm một hành động hệ sẽ xảy ra mâu thuẫn. Rõ ràng, lựa chọn tối ưu nhất trong trường hợp này là chọn hành động kết thúc với thời gian sớm nhất.
- Nếu hệ có thể đáp ứng nhiều hơn một hành động, thì lựa chọn tốt tiếp theo chính là kết nạp hành động có thời gian bắt đầu lớn hơn thời gian kết thúc của hành động trước đó và có thời gian kết thúc nhỏ nhất trong số các hành động còn lại. Cứ tiếp tục làm như vậy ta sẽ có được giải pháp tối ưu toàn cục.

Thuật toán tham lam dùng để giải bài toán Activity Selectio được thể hiện như sau:

Algorithm Greedy-Activities- Selection(N, S[], F[]):
Input:
- N là số lượng hành động (công việc).
- S[] thời gian bắt đầu mỗi hành động.
- F[] thời gian kết thúc mỗi hành động.

Ouput:
- Danh sách thực thi nhiều nhất các hành động.

Actions:

Bước 1 (sắp xếp). Sắp xếp các hành động theo thứ tự tăng dần của thời gian kết thúc.

Bước 2 (Khởi tạo) Lựa chọn hành động đầu tiên làm phương án tối ưu (OPT=1). N = N\ {i};

Bước 3 (Lặp).

```
for each activities j∈N do {
        if ( S[j] >= F[i] ) {
                OPT = OPT∪j; i = j; N = N\{i}
        }
}
```

Bước 4 (Trả lại kết quả).

Return (OPT)

EndActions.

Hình 3.5. Thuật toán tham lam giải bài toán Activity Selection

Kiểm nghiệm thuật toán:

Ví dụ. Thuật toán Greedy-ActivitiesN-Selection(int N, int S[], int F[]):
Input:

- Số lượng hành động n = 8.
- Thời gian bắt đầu S[] = {1, 3, 0, 5, 8, 5, 9, 14}.
- Thời gian kết thúc F[] = {2, 4, 6, 7, 9, 9, 12, 18}.

Output:

- OPT = { Tập nhiều nhất các hành động có thể thực hiện bởi một máy}.

Phương pháp tiến hành::

Bước	i= ? j =?	(S[j] >=F[i]) ? i=?	OPT=?
			OPT = OPT∪{1}.
1	i=1; j=2	(3>=2): Yes; i=2.	OPT = OPT∪{2}.
2	i=2; j=3;	(0>=4): No; i=2.	OPT = OPT∪ϕ.
3	i=2; j=4;	(5>=4): Yes; i=4.	OPT = OPT∪{4}.
4	i=4; j=5;	(8>=7): Yes; i=5.	OPT = OPT∪{5}.
5	i=5; j=6;	(5>=9): No; i=5.	OPT = OPT∪ϕ.
6	i=5; j=7;	(9>=9): Yes; i=7.	OPT = OPT∪{7}.
7	i=7; j=8;	(14>=12): Yes; i=8.	OPT = OPT∪{8}.
OPT = { 1, 2, 4, 5, 7, 8}			

Cài đặt thuật toán: thuật toán được cài đặt với khuôn dạng dữ liệu vào trong file data.in và kết quả ra trong file ketqua.out như sau:

File data.in:

- Dòng đầu tiên ghi lại số tự nhiên N là số lượng hành động.
- N dòng kế tiếp, mỗi dòng ghi lại bộ đôi Start[i], Finish[i] tương ứng với thời gian bắt đầu và thời gian kết thúc mỗi hành động. Hai số được viết cách nhau một vài khoảng trống.

File ketqua.out:

- Dòng đầu ghi lại số lượng lớn nhất các hành động.
- Dòng kế tiếp ghi lại các hành động được lựa chọn. Các hành động được viết cách nhau một vài khoảng trống.

Ví dụ về file data.in và ketqua.out:

data.in	ketqua.out
6	4
1 2	1 2 4 5
3 4	
0 6	
5 7	
8 9	
5 9	

Chương trình giải bài toán Activity Selection như dưới đây:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#define MAX 1000
using namespace std;
int Start[MAX], Finish[MAX], n, XOPT[MAX], dem=0;
//đọc dữ liêu từ file data.in
void Read_Data(){
	ifstream fp("data.in"); fp>>n;
	for (int i=1; i<=n; i++){
		fp>>Start[i];
		fp>>Finish[i];
		XOPT[i]=false;
	}
	fp.close();
}
//Sắp xếp tăng dần theo thời gian kết thúc các hành động
void Sapxep(void) {
	for(int i=1; i<=n-1; i++){
		for(int j=i+1; j<=n; j++){
			if(Finish[i]> Finish[j]){
				int t = Finish[i]; Finish[i]=Finish[j];Finish[i]=t;
				t = Start[i];Start[i]=Start[j];Start[i]=t;
			}
		}
	}
}
//đưa ra kết quả tối ưu
void Result(void){
	ofstream fp("ketqua.out");
	fp<<dem<<endl;
	for(int i=1; i<=n; i++){
		if(XOPT[i])
			fp<<i<<setw(3);
	}
}
void Greedy_Solution(void){ //Thuật toán tham lam
	Read_Data();//đọc dữ liêu từ file data.in
	Sapxep();//Bước 1: Sắp xếp
	int i =1; XOPT[i]=true; dem=1; //Bước 2. Khởi tạo
```

```
        for(int j=2; j<=n; j++){//Bước 3: lặp
            if(Finish[i]<=Start[j]){
                dem++; i = j; XOPT[i]=true;

            }
        }
        Result();//Bước 4. Trả lại kết quả
}
int main(void){
        Greedy_Solution();
}
```

Ví dụ 3.12. *Sắp đặt lại các kỹ tự giống nhau trong xâu ký tự.* Cho xâu ký tự s[] có độ dài n và số tự nhiên d. Hãy sắp đặt lại các ký tự trong xâu s[] sao cho các ký tự giống nhau đều cách nhau một khoảng là d. Nếu bài toán có nhiều nghiệm, hãy đưa ra một cách sắp đặt đầu tiên tìm được. Nếu bài toán không có lời giải hãy đưa ra thông báo "Vô nghiệm".

Ví dụ.

Input:

- Xâu ký tự S[] ="ABB";
- Khoảng cách d = 2.

Output: BAB

Input:

- Xâu ký tự S[] ="AAA";
- Khoảng cách d = 2.

Output: Vô nghiệm.

Input:

- Xâu ký tự S[] ="GEEKSFORGEEKS";
- Khoảng cách d = 3.

Output: EGKEGKESFESFO

Lời giải. Dễ dàng nhận thấy, trường hợp xấu nhất xảy ra khi ta không có phương án sắp đặt lại các ký tự giống nhau trong xâu s[] cách nhau một khoảng d. Trường hợp này dễ dàng đoán nhận được bằng cách chọn ký tự $x \in s$ có số lần xuất hiện nhiều lần nhất trong s[], sau đó dãn các ký tự x cách nhau một khoảng d bắt đầu tại vị trí $i = 0$. Nếu (i+k*d)≥n thì ta có kết luận ngay bài toán vô nghiệm (k số lần xuất hiện ký tự x). Trong trường hợp, (i+k*d)<n thì ký tự x sắp đặt được với khoảng cách d ta chỉ cần đặt k ký tự x, mỗi ký tự cách nhau một khoảng là d. Quá trình sắp đặt được tiến hành với ký tự xuất hiện nhiều lần nhất còn lại cho đến ký tự cuối cùng. Thuật toán tham lam dùng giải quyết bài toán được thể hiện trong Hình 2.6 dưới đây.

Thuật toán Greedy-Arrang-String (S[], d):
Input:
- Xâu ký tự S[].
- Khoảng cách giữa các ký tự d.

Output: Xâu ký tự được sắp đặt lại thỏa mãn yêu cầu bài toán.
Formats : Greedy-Arrang-String(S, d, KQ);
Actions:

Bước 1. Tìm Freq[] là số lần xuất hiện mỗi ký tự trong xâu.
Bước 2. Sắp xếp theo thứ tự giảm dần theo số xuất hiện ký tự.
Bước 3. (Lặp).

```
i = 0; k =<Số lượng ký tự trong Freq[]>;
While ( (i < k ) {
        p = Max(Freq); //Chọn ký tự xuất hiện nhiều lần nhất.
        For ( t = 0; t<p; t++ ) // điền các ký tự i, i+d, i+2d, ..,i + pd
                if (i+(t*d)>n ) { < Không có lời giải>; return;>
                KQ[i + (t*d)]  = Freq[i].kytu;
        }
        i++;
}
```

Bước 4(Trả lại kết quả): Return(KQ);
EndActions.

Hình 3.6. Thuật toán tham lam giải quyết bài toán.

Thử nghiệm thuật toán:

Greedy-Arrang-String (S[], d):Input : S[] = "GEEKSFORGEEKS; d= 3.

Bước 1. Tìm tập ký tự và số lần xuất hiện mỗi ký tự.

Freq[]	
G	2
E	4
K	2
S	2
F	1
O	1
R	1

Sắp xếp theo thứ tự giảm dần của số lần xuất hiện.

Bước 2. Sắp xếp theo thứ tự giảm dần số lần xuất hiện.

Freq[]	
E	4
G	2
K	2
S	2
F	1
O	1
R	1

Bước 3. Lặp.

	KQ[I + p*d]												
I =0	E			E			E			E			
i=1	E	G		E	G		E			E			
i=2	E	G	K	E	G	K	E			E			
i=3	E	G	K	E	G	K	E	S		E	S		
i=4	E	G	K	E	G	K	E	S	F	E	S		
i=5	E	G	K	E	G	K	E	S	F	E	S	O	
i=6	E	G	K	E	G	K	E	S	F	E	S	O	R

3.5. Mô hình thuật toán chia và trị (Devide and Conquer Algorithm)

Thuật toán chia để trị (Devide and Conquer) dùng để giải lớp các bài toán có thể thực hiện được thông qua ba bước:

- **Bước chia (Devide)**. Chia bài toán lớn thành những bài toán con có cùng kiểu với bài toán lớn.
- **Bước trị (Conquer)**. Giải các bài toán con đã chia ở bước trước đó. Thông thường các bài toán con chỉ khác nhau về dữ liệu vào nên ta có thể thực hiện bằng một thủ tục đệ qui.

- **Bước tổng hợp (Combine)**. Tổng hợp lại kết quả của các bài toán con để nhận được kết quả của bài toán lớn.

Ví dụ 3.13. Nâng *x* lên lũy thừa *n*.

Lời giải.

Bước chia. Ta có $x^n = \begin{cases} x^{n/2}.x^{n/2} & nếu\ n\ chẵn \\ x.x^{n/2}.x^{n/2} & nếu\ x\ lẻ \end{cases}$.

Bước trị. Tính toán $x^{n/2}$ trong trường hợp x chẵn, $x.x^{n/2}$ trong trường hợp *x* lẻ.

Bước tổng hợp. Kết quả chính là $x^{n/2}.x^{n/2}$ trong trường hợp x chẵn và $x.x^{n/2}.x^{n/2}$ trong trường hợp x lẻ.

Chương trình nâng x lên lũy thừa n bằng kỹ thuật chia và trị được thể hiện như sau:

```
#include <iostream>
#include <iomanip>
using namespace std;
int power(int x, unsigned int n){
        if( n == 0)
                return 1;
        else if (n%2 == 0)
                return power(x, n/2)*power(x, n/2);
        else
                return x*power(x, n/2)*power(x, n/2);
}
int main(){
        int x = 2;
        unsigned int n = 5;
        cout<<power(x, n)<<setw(3);
}
```

Thử nghiệm thuật toán: tính S = power(2, 5)

```
S       = power(2, 5)
        = 2* power(2, 2)* power(2, 2)
        = 2* power(2, 1)* power(2, 1)* power(2, 1)* power(2, 1)
        = 2* 2* power(0, 1)*2* power(2, 0)* 2*power(2, 0)* 2*power(2, 0)
        = 2*2*2*2*2
        =32
```

Thuật toán trên có độ phức tạp là O(n). Thuật toán trình bày dưới đâu có độ phức tạp O(log(n)).

```
#include <iostream>
#include <iomanip>
using namespace std;
int power(int x, unsigned int n) {
        int temp;
```

```
        if( n == 0)
                return 1;
        temp = power(x, n/2);
        if (n%2 == 0)
                return temp*temp;
        else
                return x*temp*temp;
}
int main(){
        int x = 2;
        unsigned int n = 5;
        cout<<power(x, n)<<setw(3);
}
```

3.6. Mô hình thuật toán nhánh cận (Branch and Bound Algorithm)

Thuật toán nhánh cận thường được dùng trong việc giải quyết các bài toán tối ưu tổ hợp. Bài toán được phát biểu dưới dạng sau:

Tìm min { $f(X) : X \in D$ }, với $D = \{ X = (x_1, x_2, .., x_n) \in A_1 \times A_2 \times A_n$: *X thỏa mãn tính chất P* }.

- $X \in D$ được gọi là một phương án của bài toán.
- Hàm $f(X)$ được gọi là hàm mục tiêu của bài toán.
- Miền D được gọi là tập phương án của bài toán.

Để giải quyết bài toán trên, ta có thể dùng thuật toán quay lui duyệt các phần tử $X \in D$, phần tử X* làm cho F(X*) đạt giá trị nhỏ nhất (lớn nhất) là phương án tối ưu của bài toán. Thuật toán nhánh cận có thể giải được bài toán đặt ra nếu ta xây dựng được một hàm g xác định trên tất cả phương án bộ phận cấp k của bài toán sao cho:

$$g(a_1, a_2, .., a_k) \leq \min \{ f(X): X \in D, x_i = a_i, i = 1, 2, .., k \}$$

Nói cách khác, giá trị của hàm g tại phương án bộ phận cấp k là $g(a_1, a_2, .., a_k)$ không vượt quá giá trị nhỏ nhất của hàm mục tiêu trên tập con các phương án.

$$D(a_1, a_2, .., a_k) = \{ X \in D: x_i = a_i, i = 1, 2, .., k \}$$

Giá trị của hàm $g(a_1, a_2, .., a_k)$ là cận dưới của hàm mục tiêu trên tập $D(a_1, a_2, .., a_k)$. Hàm g được gọi là hàm cận dưới, $g(a_1, a_2, .., a_k)$ gọi là cận dưới của tập $D(a_1, a_2, .., a_k)$.

Giả sử ta đã có hàm g. Để giảm bớt khối lượng duyệt trên tập phương án, bằng thuật toán quay lui ta xác định được X^* là phương án làm cho hàm mục tiêu có giá trị nhỏ nhất trong số các phương án tìm được $f^* = f(X^*)$. Ta gọi X^* là phương án tốt nhất hiện có, f^* là kỷ lục hiện tại.

Nếu

$$f^* < g(a_1, a_2, .., a_k)$$

thì

$$f^* < g(a_1, a_2, .., a_k) \le \min \{ f(X): X \in D, x_i = a_i, i=1, 2, .., k \}.$$

Điều này có nghĩa tập $D(a_1, a_2, .., a_k)$ chắc chắn không chứa phương án tối ưu. Trong trường hợp này ta không cần phải triển khai phương án bộ phận $(a_1, a_2, .., a_k)$. Tập $D(a_1, a_2, .. a_k)$ cũng bị loại bỏ khỏi quá trình duyệt. Nhờ đó, số các phương án cần duyệt nhỏ đi trong quá trình tìm kiếm. Thuật toán nhánh cận tổng quát được mô tả chi tiết trong Hình 2.7.

```
Thuật toán Branch_And_Bound (k) {
        for a_k ∈ A_k do {
                if (<chấp nhận a_k>){
                        x_k = a_k;
                        if ( k==n )
                                <Cập nhật kỷ lục>;
                        else if ( g(a_1, a_2,..,a_k) ≤ f*)
                                Branch_And_Bound (k+1)    ;
                }
        }
}
```

Hình 3.7. Thuật toán Branch-And-Bound

3.7. Mô hình thuật toán qui hoạch động (Dynamic Programming Algorithm)

Phương pháp qui hoạch động dùng để giải lớp các bài toán thỏa mãn những điều kiện sau:

- Bài toán lớn cần giải có thể phân rã được thành nhiều bài toán con. Trong đó, sự phối hợp lời giải của các bài toán con cho ta lời giải của bài toán lớn. Bài toán con có lời giải đơn giản được gọi là cơ sở của qui hoạch động. Công thức phối hợp nghiệm của các bài toán con để có nghiệm của bài toán lớn được gọi là công thức truy hồi của qui hoạch động.
- Phải có đủ không gian vật lý lưu trữ lời giải các bài toán con (Bảng phương án của qui hoạch động). Vì qui hoạch động đi giải quyết tất cả các bài toán con, do vậy nếu ta không lưu trữ được lời giải các bài toán con thì không thể phối hợp được lời giải giữa các bài toán con.
- Quá trình giải quyết từ bài toán cơ sở (bài toán con) để tìm ra lời giải bài toán lớn phải được thực hiện sau hữu hạn bước dựa trên bảng phương án của qui hoạch động.

Ví dụ 3.15. Tìm số các cách chia số tự nhiên n thành tổng các số tự nhiên nhỏ hơn n. Các cách chia là hoán vị của nhau chỉ được tính là một cách.

Lời giải. Gọi F [m, v] là số cách phân tích số v thành tổng các số nguyên dương nhỏ hơn hoặc bằng m. Khi đó, số các cách chia số v thành tổng các số nhỏ hơn hoặc bằng m được chia thành hai loại:

- **Loại 1**: Số các cách phân tích số v thành tổng các số nguyên dương không chứa số m. Điều này có nghĩa số các cách phân tích Loại 1 là số các cách chia số v thành tổng các số nguyên dương nhỏ hơn hoặc bằng m-1. Như vậy số Loại 1 chính là F[m-1,v].
- **Loại 2**: Số các cách phân tích số v thành tổng các số nguyên dương chứa ít nhất một số m. Nếu ta xem sự xuất hiện của một hay nhiều số m trong phép phân tích v chỉ là 1, thì theo nguyên lý nhân số lượng các cách phân tích loại này chính là số cách phân tích số v thành tổng các số nhỏ hơn m-v hay F[m, v-m].

Trong trường hợp m> v thì F[m,v-m] = 0 nên ta chỉ có các cách phân tích loại 1. Trong trường hợp m≤v thì ta có cả hai cách phân tích loại 1 và loại 2. Vì vậy:

- F[m, v] = F[m-1, v] nếu m>v.
- F[m, v] = F[m-1, v] + F[m, v-m] nếu m ≤v.

Tổng quát.

- **Bước cơ sở**: F[0,0] = 1; F[0,v] = 0 với v>0.
- **Tính toán giá trị bảng phương án dựa vào công thức truy hồi**:
  - F[m, v] = F[m-1, v] nếu v > m;
  - F[m, v] = F[m-1, v] + F[m, v-m] nếu m≥v.
- **Lưu vết**. tìm F[5, 5] = F[4, 5] + F[5, 0] = 6 + 1 = 7.

F[m, v]

	0	1	2	3	4	5
0	1	0	0	0	0	0
1	1	1	1	1	1	1
2	1	1	2	2	3	3
3	1	1	2	3	4	5
4	1	1	2	3	5	6
5	1	1	2	3	5	7

m

Ví dụ 3.16. Giải bài toán cái túi bằng qui hoạch động.

Lời giải. Gọi A =(a_1, a_2, …, a_n), C =(c_1, c_2, …, c_n) là vector trọng lượng và giá trị sử dụng các đồ vật. Gọi F[i, w] là giá trị lớn nhất bằng cách chọn các đồ vật {1, 2, ..,i} với giới hạn trọng lượng w. Khi đó, giá trị lớn nhất khi chọn n đồ vật với trọng lượng b là F[n,m], trong đó m là giới hạn trọng lượng túi.

Bước cơ sở: F[0, w] = 0 vì giá trị lớn nhất khi chọn 0 gói với giới hạn trọng lượng w là 0.

Bước truy hồi: với mọi i>0 và trọng lượng w, khi đó việc chọn tối các gói từ {1, 2, .., i} sẽ có hai khả năng: chọn được đồ vật i và không chọn được đồ vật i. Giá trị lớn nhất của một trong hai cách chọn này chính là F[i, w].

- Nếu không chọn được đồ vật i thì giá trị lớn nhất ta chỉ có thể chọn từ i-1 đồ vật với trọng lượng w. Như vậy, F[i, w] = F[i-1, w].
- Nếu chọn được đồ vật i thì F[i, w] chính là giá trị sử dụng đồ vật i là c_i cộng thêm với giá trị lớn nhất được chọn từ các đồ vật {1, 2, .., i-1} với giới hạn trọng lượng là w-a_i. Nói cách khác F[i, w] = c_i + F[i-1, w-a_i].

Xây dựng bảng phương án: Bảng phương án gồm n+1 dòng và m+1 cột (m là giới hạn trọng lượng túi). Dòng đầu tiên ghi toàn bộ số 0 đó là lời giải bài toán cơ sở. Sử dụng công thức truy hồi tính dòng 1 dựa vào dòng 0, tính dòng 2 dựa vào các hàng còn lại cho đến khi nhận được đầy đủ n +1 dòng.

Truy vết: F[n, m] trong bảng phương án chính là giá trị lớn nhất chọn n đồ vật với giới hạn trọng lượng m. Nếu F[n,m] = F[n-1, m] thì phương án tối ưu không chọn đồ vật n và ta truy tiếp F[n-1, m]. Nếu F[n,m] ≠ F[n-1, m] thì phương án tối ưu có đồ vật n và ta truy tiếp F[n-1, m-a_n]. Truy tiếp đến hàng thứ 0 của bảng phương án ta nhận được phương án tối ưu.

CHƯƠNG 4. LÝ THUYẾT ĐỒ THỊ

Đồ thị là một cấu trúc dữ liệu rời rạc bao gồm các đỉnh và các cạnh nối các cặp đỉnh này. Với quan niệm như trên, một mạng máy tính được xem như một đồ thị có mỗi đỉnh là một máy tính, có các cạnh là một liên kết giữa các máy tính khác nhau trong mạng. Các mạch điện, các hệ thống giao thông, các mạng xã hội đều được xem xét như một đồ thị. Có thể nói đồ thị được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau của khoa học máy tính. Nội dung chính của chương này đề cập đến phương pháp biểu diễn và các thuật toán trên đồ thị.

4.1. Định nghĩa và khái niệm

Ta có thể phân chia đồ thị thành hai loại: đồ thị vô hướng (directed graph) và đồ thị có hướng (directed graph). Mỗi loại đồ thị lại được chia thành 3 loại: đơn đồ thị, đa đồ thị và giả đồ thị. Mỗi loại đồ thị có các thuật ngữ chung và những khái niệm riêng. Dưới đây là một số thuật ngữ cơ bản trên các loại đồ thị.

4.1.1. Một số thuật ngữ cơ bản trên đồ thị

Định nghĩa. Bộ đô $G =<V, E>$, trong đó $V = \{1, 2 , .., n\}$ là tập hợp hữu hạn được gọi là tập đỉnh, E là tập có thứ tự hoặc không có thứ tự các cặp đỉnh trong V được gọi là tập cạnh.

Đồ thị vô hướng. Đồ thị G=<V,E> được gọi là đồ thị vô hướng nếu các cạnh thuộc E là các cặp không tính đến thứ tự các đỉnh trong V.

Đơn đồ thị vô hướng. Đồ thị G=<V,E> được gọi là đơn đồ thị vô hướng nếu G là đồ thị vô hướng và giữa hai đỉnh bất kỳ thuộc V có nhiều nhất một cạnh nối.

Đa đồ thị vô hướng. Đồ thị G=<V,E> được gọi là đơn đồ thị vô hướng nếu là đồ thị vô hướng và tồn tại một cặp đỉnh trong V có nhiều hơn một cạnh nối. Cạnh $e_1 \in E$, $e_2 \in E$ được gọi là cạnh bội nếu chúng cùng chung cặp đỉnh.

Giả đồ thị vô hướng. Đồ thị G = <V, E> bao gồm V là tập đỉnh, E là họ các cặp không có thứ tự gồm hai phần tử (hai phần tử không nhất thiết phải khác nhau) trong V được gọi là các cạnh. Cạnh e được gọi là khuyên nếu có dạng e =(u, u), trong đó u là đỉnh nào đó thuộc V.

Đơn đồ thị có hướng. Đồ thị G = <V, E> bao gồm V là tập các đỉnh, E là tập các cặp có thứ tự gồm hai phần tử của V gọi là các cung. Giữa hai đỉnh bất kỳ của G tồn tại nhiều nhất một cung.

Đa đồ thị có hướng. Đồ thị G = <V, E> bao gồm V là tập đỉnh, E là cặp có thứ tự gồm hai phần tử của V được gọi là các cung. Hai cung e_1, e_2 tương ứng với cùng một cặp đỉnh được gọi là cung lặp.

Giả đồ thị có hướng. Đa đồ thị G = <V, E>, trong đó V là tập đỉnh, E là tập các cặp không có thứ tự gồm hai phần tử (hai phần tử không nhất thiết phải khác nhau)

trong V được gọi là các cung. Cung e được gọi là khuyên nếu có dạng e =(u, u), trong đó u là đỉnh nào đó thuộc V.

4.1.2. Một số thuật ngữ trên đồ thị vô hướng

Đỉnh kề. Hai đỉnh *u* và *v* của đồ thị vô hướng $G = <V, E>$ được gọi là kề nhau nếu (u,v) là cạnh thuộc đồ thị *G*. Nếu $e = (u, v)$ là cạnh của đồ thị *G* thì ta nói cạnh này liên thuộc với hai đỉnh *u* và *v*, hoặc ta nói cạnh *e* nối đỉnh *u* với đỉnh *v*, đồng thời các đỉnh *u* và *v* sẽ được gọi là đỉnh đầu của cạnh (u,v).

Bậc của đỉnh. Ta gọi bậc của đỉnh *v* trong đồ thị vô hướng là số cạnh liên thuộc với nó và ký hiệu là $deg(v)$. Đỉnh có bậc là 0 được gọi là đỉnh cô lập. Đỉnh có đỉnh bậc 1 được gọi là đỉnh treo.

Đường đi, chu trình. Đường đi độ dài *n* từ đỉnh *u* đến đỉnh *v* trên đồ thị vô hướng $G=<V,E>$ là dãy $x_0, x_1, \ldots, x_{n-1}, x_n$, trong đó *n* là số nguyên dương, $x_0=u$, $x_n=v$, $(x_i, x_{i+1}) \in E$, $i = 0, 1, 2, \ldots, n-1$. Đường đi có đỉnh đầu trùng với đỉnh cuối gọi là chu trình.

Tính liên thông. Đồ thị vô hướng được gọi là liên thông nếu luôn tìm được đường đi giữa hai đỉnh bất kỳ của nó.

Thành phần liên thông. Đồ thị vô hướng liên thông thì số thành phần liên thông là 1. Đồ thị vô hướng không liên thông thì số liên thông của đồ thị là số các đồ thị con của nó liên thông.

Đỉnh trụ. Đỉnh $u \in V$ được gọi là đỉnh trụ nếu loại bỏ u cùng với các cạnh nối với u làm tăng thành phần liên thông của đồ thị.

Cạnh cầu. Cạnh $(u,v) \in E$ được gọi là cầu nếu loại bỏ (u,v) làm tăng thành phần liên thông của đồ thị.

Đỉnh rẽ nhánh. Đỉnh s được gọi là đỉnh rẽ nhánh (đỉnh thắt) của cặp đỉnh u, v nếu mọi đường đi từ u đến v đều qua s.

4.1.3. Một số thuật ngữ trên đồ thị có hướng

Đỉnh kề. Nếu $e=(u,v)$ là cung của đồ thị có hướng *G* thì ta nói hai đỉnh *u* và *v* là kề nhau, và nói cung (u, v) nối đỉnh *u* với đỉnh *v*, hoặc nói cung này đi ra khỏi đỉnh *u* và đi vào đỉnh *v*. Đỉnh *u* được gọi là đỉnh đầu, đỉnh *v* được gọi là đỉnh cuối của cung (u,v).

Bán bậc của đỉnh. Ta gọi bán bậc ra của đỉnh *v* trên đồ thị có hướng là số cung của đồ thị đi ra khỏi *v* và ký hiệu là $deg^+(v)$. Ta gọi bán bậc vào của đỉnh *v* trên đồ thị có hướng là số cung của đồ thị đi vào *v* và ký hiệu là $deg^-(v)$.

Đường đi. Đường đi độ dài *n* từ đỉnh *u* đến đỉnh *v* trong đồ thị có hướng $G=<V,A>$ là dãy $x_0, x_1, \ldots, x_n$, trong đó, *n* là số nguyên dương, $u = x_0$, $v = x_n$, $(x_i, x_{i+1}) \in E$. Đường đi như trên có thể biểu diễn thành dãy các cung : $(x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n)$. Đỉnh *u* được gọi là đỉnh đầu, đỉnh *v* được gọi là đỉnh cuối của đường đi. Đường đi có đỉnh đầu trùng với đỉnh cuối $(u=v)$ được gọi là một chu trình. Đường đi hay chu trình được gọi là đơn nếu như không có hai cạnh nào lặp lại.

Liên thông mạnh. Đồ thị có hướng G=<V,E> được gọi là liên thông mạnh nếu giữa hai đỉnh bất kỳ u∈V, v∈V đều có đường đi từ u đến v.

Liên thông yếu. Ta gọi đồ thị vô hướng tương ứng với đồ thị có hướng G=<V,E> là đồ thị tạo bởi G và bỏ hướng của các cạnh trong G. Khi đó, đồ thị có hướng G=<V,E> được gọi là liên thông yếu nếu đồ thị vô hướng tương ứng với nó là liên thông.

Thành phần liên thông mạnh. Đồ thị con có hướng H =<V1, E1> được gọi là một thành phần liên thông mạnh của đồ thị có hướng G=<V,E> nếu V1⊆V, E1⊆E và H liên thông mạnh.

4.1.4. Một số loại đồ thị đặc biệt

Dưới đây là một số dang đơn đồ thị vô hướng đặc biệt có nhiều ứng dụng khác nhau của thực tế.

Đồ thị đầy đủ. Đồ thị đầy đủ n đỉnh, ký hiệu là Kn, là đơn đồ thị vô hướng mà giữa hai đỉnh bất kỳ của nó đều có cạnh nối. Ví dụ đồ thị K3, K4, K5 trong Hình 4.1a.

Đồ thị vòng. Đồ thị vòng Cn (n≥3) có các cạnh (1,2), (2,3),..,(n-1,n), (n,1). Ví dụ đồ thị C3, C4, C5 trong Hình 4.1.b.

Đồ thị bánh xe. Đồ thị bánh xe Wn thu được bằng cách bổ sung một đỉnh nối với tất cả các đỉnh của Cn. Ví dụ đồ thị W3, W4, W5 trong Hình 4.1.c.

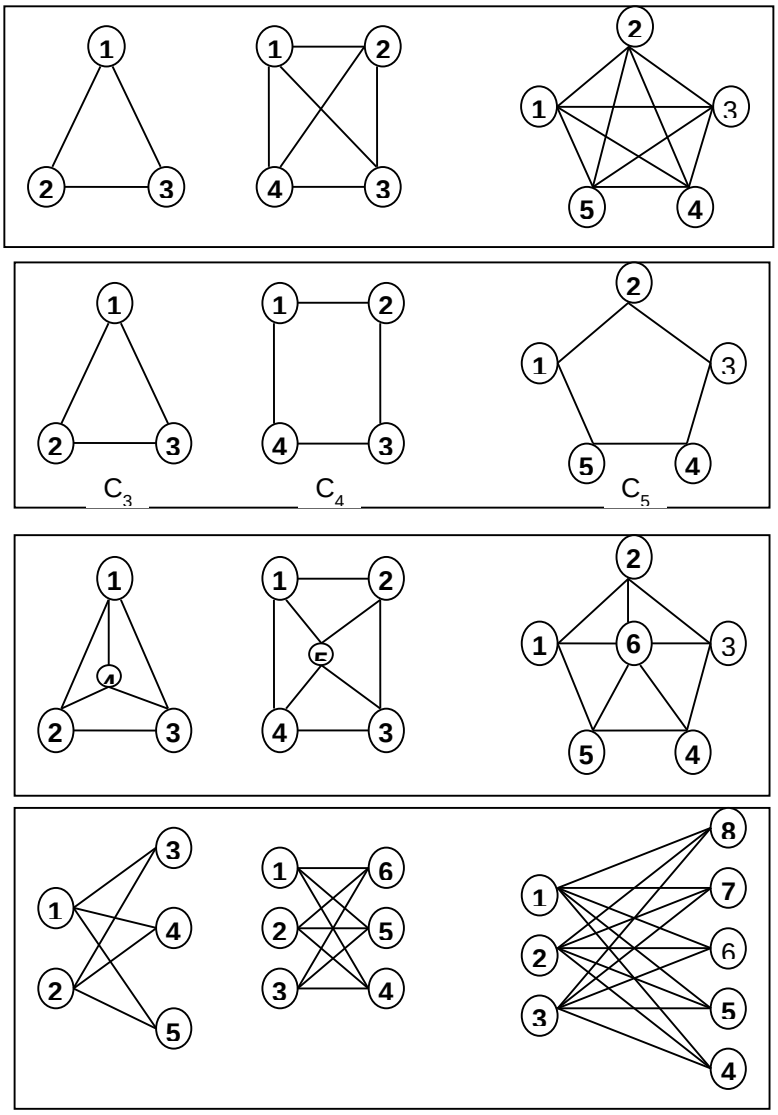
Đồ thị hai phía. Đồ thị G =<V,E> được gọi là đồ thị hai phía nếu tập đỉnh V của nó có thể phân hoạch thành hai tập X và Y sao cho mỗi cạnh của đồ thị chỉ có dạng (*x*, *y*), trong đó $x \in X$ và $y \in Y$. Ví dụ đồ thị $K_{2,3}$, K_{33}, $K_{3,5}$ trong Hình 4.1.d.

Hình 4.1a

Hình 4.1b

Hình 4.1c

Hình 4.1d



Hình 4.1. Một số dạng đồ thị đặc biệt.

4.2. Biểu diễn đồ thị

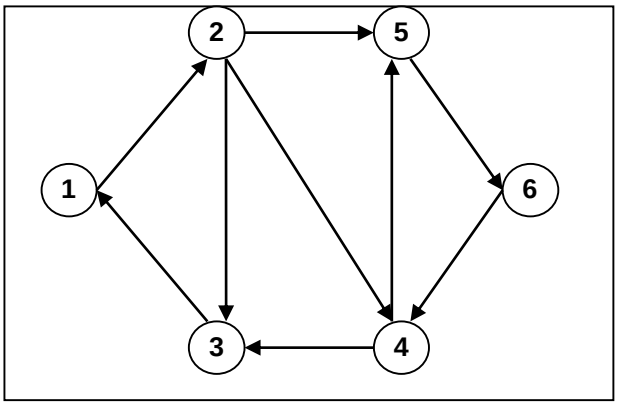
Để lưu trữ, xử lý hiệu quả ta cần có phương pháp biểu diễn đồ thị trên máy tính. Ba phương pháp biểu diễn thông dụng thường được ứng dụng trên đồ thị đó là: ma trận kề, danh sách cạnh và danh sách kề. Phương pháp biểu diễn cụ thể được thể hiện như dưới đây.

4.2.1. Biểu diễn bằng ma trận kề

Phương pháp biểu diễn đồ thị bằng ma trận kề là phép làm tương ứng đồ thị G = <V,E> với một ma trận vuông cấp n. Các phần tử của ma trận kề được xác định như dưới đây.

$$a_{uv} = \begin{cases} 1 & nếu\ (u,v) \in E \\ 0 & otherwise \end{cases}$$

Ví dụ với đồ thị cho bởi Hình 5.2 sẽ cho ta biểu diễn ma trận kề như sau:



0	1	0	0	0	0
0	0	1	1	1	0
1	0	0	0	0	0
0	0	1	0	1	0
0	0	0	0	0	1

Hình 4.2. Biểu diễn ma trận kề của đồ thị.

Tính chất của ma trận kề:

- Ma trận kề biểu diễn đồ thị vô hướng G = <V,E> là ma trận đối xứng.
- Ma trận kề biểu diễn đồ thị có hướng G =<V,E> thường là không đối xứng.
- Tổng các phần tử của ma trận kề biểu diễn đồ thị vô hướng G = <V,E> bằng *2.m*, trong đó *m* là số cạnh của đồ thị.
- Tổng các phần tử của ma trận kề biểu diễn đồ thị có hướng G = <V,E> bằng đúng *m*, trong đó *m* là số cạnh của đồ thị.
- Tổng các phần tử của hàng *u* hoặc cột *u* của ma trận kề biểu diễn đồ thị vô hướng G= <V,E> là bậc của đỉnh u (deg(u)).
- Tổng các phần tử của hàng *u* của ma trận kề biểu diễn đồ thị có hướng G= <V,E> là bán bậc ra của đỉnh u ($deg^+(u)$).
- Tổng các phần tử của cột *u* của ma trận kề biểu diễn đồ thị có hướng G= <V,E> là bán bậc vào của đỉnh u ($deg^-(u)$).

Ưu điểm của ma trận kề:

- Đơn giản dễ cài đặt trên máy tính bằng cách sử dụng một mảng hai chiều.
- Dễ dàng kiểm tra được hai đỉnh *u, v* có kề với nhau hay không bằng đúng một phép so sánh (a[u][v]≠0).

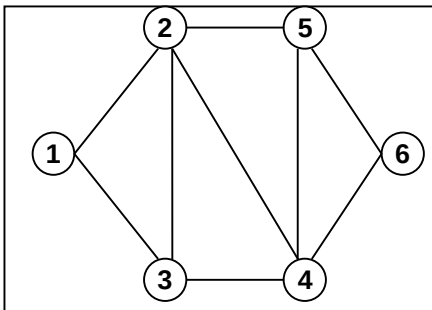
Nhược điểm của ma trận kề:

- Lãng phí bộ nhớ: bất kể số cạnh nhiều hay ít ta cần n^2 đơn vị bộ nhớ để biểu diễn.
- Không thể biểu diễn được với các đồ thị có số đỉnh lớn .

- Để xem xét đỉnh đỉnh u có những đỉnh kề nào cần mất n phép so sánh kể cả đỉnh u là đỉnh cô lập hoặc đỉnh treo.

4.2.2. Biểu diễn đồ thị bằng danh sách cạnh

Phương pháp biểu diễn đồ thị G =<V,E> bằng cách liệt kê tất cả các cạnh của nó được gọi là phương pháp biểu diễn bằng danh sách cạnh. Đối với đồ thị có hướng ta liệt kê các cung tương ứng. Đối với đồ thị vô hướng ta chỉ cần liệt kê các cạnh (u,v)∈E mà không cần liệt kê các cạnh (v,u)∈E. Ví dụ về biểu diễn đồ thị bằng danh sách cạnh được cho trong Hình 5.3.



Đỉnh đầu	Đỉnh cuối
1	2
1	3
2	3
2	4
2	5
3	4
4	5
4	6

Hình 4.3. Biểu diễn đồ thị bằng danh sách cạnh

Tính chất của danh sách cạnh:

- Đỉnh đầu luôn nhỏ hơn đỉnh cuối của mỗi cạnh đối với đồ thị vô hướng.
- Đỉnh đầu không phải lúc nào cũng nhỏ hơn đỉnh cuối của mỗi cạnh đối với đồ thị có hướng.
- Số các số có giá trị u thuộc cả tập đỉnh đầu và tập đỉnh cuối các cạnh là bậc của đỉnh u đối với đồ thị vô hướng.
- Số các số có giá trị u thuộc cả tập đỉnh đầu các cạnh là bán bậc ra của đỉnh u đối với đồ thị có hướng.
- Số các số có giá trị u thuộc cả tập đỉnh cuối là bán bậc vào của đỉnh u đối với đồ thị có hướng.

Ưu điểm của danh sách cạnh:

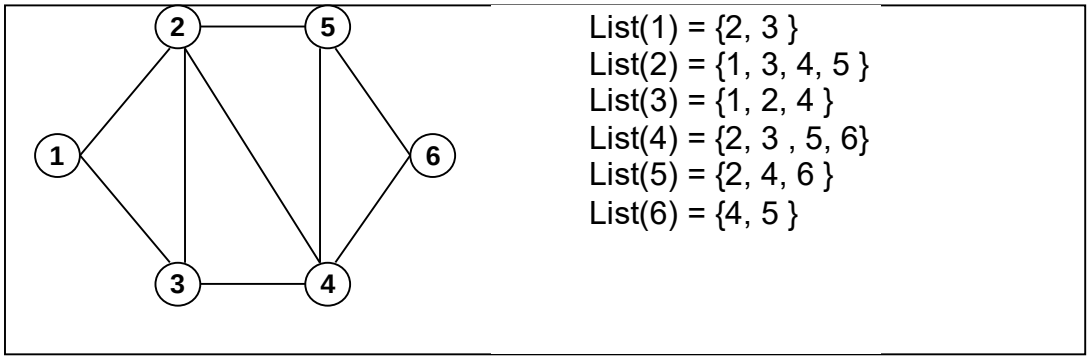
- Trong trường hợp đồ thị thưa (m<6n), biểu diễn bằng danh sách cạnh tiết kiệm được không gian nhớ.
- Thuận lợi cho một số thuật toán chỉ quan tâm đến các cạnh của đồ thị.

Nhược điểm của danh sách cạnh:

- Khi cần duyệt các đỉnh kề với đỉnh u bắt buộc phải duyệt tất cả các cạnh của đồ thị. Điều này làm cho thuật toán có chi phí tính toán cao.

4.2.3. Biểu diễn đồ thị bằng danh sách kề

Ta định nghĩa $List(u) = \{v \in V: (u, v) \in E\}$ là danh sách các đỉnh kề với đỉnh u. Biểu diễn đồ thị bằng danh sách kề là phương pháp liệt kê tập đỉnh kề của mỗi đỉnh. Ví dụ về biểu diễn đồ thị bằng danh sách kề được cho trong Hình 5.4.



Hình 4.4. Biểu diễn đồ thị bằng danh sách kề.

Tính chất của danh sách kề:

- Lực lượng tập đỉnh kề của đỉnh *u* là bậc của đỉnh *u* đối với đồ thị vô hướng (deg(*u*)=|List(*u*)|.
- Lực lượng tập đỉnh kề của đỉnh *u* là bán bậc ra của đỉnh *u* đối với đồ thị có hướng ($deg^+(u)$=|List(*u*)|.
- Số các số có giá trị *u* thuộc tất cả các danh sách kề là bán bậc vào của đỉnh *u* đối với đồ thị có hướng.

Ưu điểm của danh sách kề:

- Dễ dàng duyệt tất cả các đỉnh của một danh sách kề.
- Dễ dàng duyệt các cạnh của đồ thị trong mỗi danh sách kề.
- Tối ưu việc cài đặt một số giải thuật trên đồ thị.

Nhược điểm của danh sách kề:

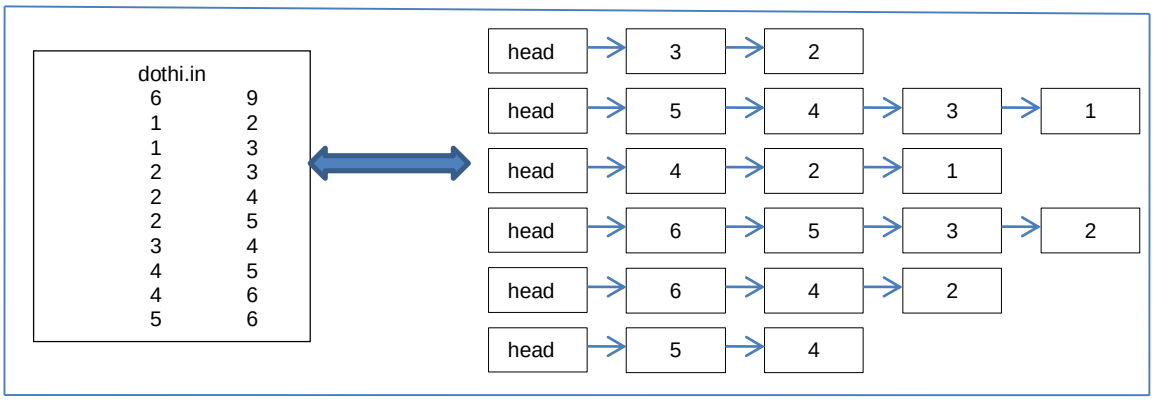
- Khó khăn cho người học có kỹ năng lập trình yếu vì khi biểu diễn đồ thị ta phải dùng một mảng, mỗi phần tử của nó là một danh sách liên kết.

4.2.4. Biểu diễn đồ thị bằng danh sách kề dựa vào danh sách liên kết

Như đã đề cập ở trên, để biểu diễn đồ thị G =<V,E> bằng danh sách kề ta cần phải tạo nên một mảng, mỗi phần từ của mảng là một danh sách liên kết. Mỗi danh danh sách liên kết lưu trữ danh sách kề của đỉnh tương ứng. Chương trình dưới đây cài đặt phương pháp biểu diễn đồ thị bằng danh sách kề với khuôn dạng dữ liệu trong file dothi.in theo khuôn dạng:

- Dòng đầu tiên ghi lại hai số N, M tương ứng với số đỉnh và số cạnh của đồ thị. Hai số được viết cách nhau một vài khoảng trống.
- M dòng kế tiếp mỗi dòng ghi lại một cạnh của đồ thị. Đỉnh đầu và đỉnh cuối mỗi được viết cách nhau một vài khoảng rỗng.

Ví dụ với đồ thị trong Hình 5.4 sẽ cho ta file dothi.in như dưới đây:



4.3. Các thuật toán tìm kiếm trên đồ thị

Có nhiều thuật toán trên đồ thị được xây dựng để duyệt tất cả các đỉnh của đồ thị sao cho mỗi đỉnh được viếng thăm đúng một lần. Những thuật toán như vậy được gọi là thuật toán tìm kiếm trên đồ thị. Chúng ta cũng sẽ làm quen với hai thuật toán tìm kiếm cơ bản, đó là duyệt theo chiều sâu DFS (Depth First Search) và duyệt theo chiều rộng BFS (Breath First Search). Trên cơ sở của hai phép duyệt cơ bản, ta có thể áp dụng chúng để giải quyết một số bài toán quan trọng của lý thuyết đồ thị.

4.3.1. Thuật toán tìm kiếm theo chiều sâu (Depth First Search)

Tư tưởng cơ bản của thuật toán tìm kiếm theo chiều sâu là bắt đầu tại một đỉnh v_0 nào đó, chọn một đỉnh u bất kỳ kề với v_0 và lấy nó làm đỉnh duyệt tiếp theo. Cách duyệt tiếp theo được thực hiện tương tự như đối với đỉnh v_0 với đỉnh bắt đầu là u.

Để kiểm tra việc duyệt mỗi đỉnh đúng một lần, chúng ta sử dụng một mảng chuaxet[] gồm n phần tử (tương ứng với n đỉnh), nếu đỉnh thứ u đã được duyệt, phần tử tương ứng trong mảng chuaxet[u] có giá trị FALSE. Ngược lại, nếu đỉnh chưa được duyệt, phần tử tương ứng trong mảng có giá trị TRUE.

Biểu diễn thuật toán DFS(u):

Thuật toán DFS (u): *//u là đỉnh bắt đầu duyệt*
Begin
    <Thăm đỉnh u>;*//duyệt đỉnh u*
    chuaxet[u] := FALSE;*//xác nhận đỉnh u đã duyệt*
    for each v ∈ke(u) do *//lấy mỗi đỉnh v∈Ke(u).*
        if (chuaxet[v]) then *//vếu đỉnh v chưa duyệt*
            DFS(v); *//duyệt theo chiều sâu bắt đầu từ đỉnh v*
        EndIf;
    EndFor;
End.

Thuật toán DFS(u) có thể khử đệ qui bằng cách sử dụng ngăn xếp như Hình 5.4.

```
Thuật toán DFS(u):
Begin
    Bước 1 (Khởi tạo):
        stack = ∅; //Khởi tạo stack là ∅
        Push(stack, u); //Đưa đỉnh u vào ngăn xếp
        <Thăm đỉnh u>; //Duyệt đỉnh u
        chuaxet[u] = False; //Xác nhận đỉnh u đã duyệt
    Bước 2 (Lặp) :
        while ( stack ≠ ∅ ) do
            s = Pop(stack); //Loại đỉnh ở đầu ngăn xếp
            for each t∈ Ke(s) do //Lấy mỗi đỉnh t∈Ke(s)
                if ( chuaxet[t] ) then //Nếu t đúng là chưa duyệt
                    <Thăm đỉnh t>; // Duyệt đỉnh t
                    chuaxet[t] = False; // Xác nhận đỉnh t đã duyệt
```

Hình 5.6. Thuật toán DFS(u) dựa vào ngăn xếp.

Độ phức tạp thuật toán DFS:

Độ phức tạp thuật toán DFS(u) phụ thuộc vào phương pháp biểu diễn đồ thị. Độ phức tạp thuật toán DFS(u) theo các dạng biểu diễn đồ thị như sau:

- Độ phức tạp thuật toán là $O(n^2)$ trong trường hợp đồ thị biểu diễn dưới dạng ma trận kề, với n là số đỉnh của đồ thị.
- Độ phức tạp thuật toán là O(n.m) trong trường hợp đồ thị biểu diễn dưới dạng danh sách cạnh, với n là số đỉnh của đồ thị, m là số cạnh của đồ thị.
- Độ phức tạp thuật toán là O(max(n, m)) trong trường hợp đồ thị biểu diễn dưới dạng danh sách kề, với n là số đỉnh của đồ thị, m là số cạnh của đồ thị.

Bạn đọc tự chứng minh hoặc có thể tham khảo trong các tài liệu [1, 2, 3].

4.3.2. Thuật toán tìm kiếm theo chiều rộng (Breadth First Search)

Để ý rằng, với thuật toán tìm kiếm theo chiều sâu, đỉnh thăm càng muộn sẽ trở thành đỉnh sớm được duyệt xong. Đó là kết quả tất yếu vì các đỉnh thăm được nạp vào stack trong thủ tục đệ qui. Khác với thuật toán tìm kiếm theo chiều sâu, thuật toán tìm kiếm theo chiều rộng thay thế việc sử dụng stack bằng hàng đợi (queue). Trong thủ tục này, đỉnh được nạp vào hàng đợi đầu tiên là u, các đỉnh kề với u là (v_1, v_2, . . ., v_k)

được nạp vào hàng đợi nếu như nó chưa được xét đến. Quá trình duyệt tiếp theo được bắt đầu từ các đỉnh còn có mặt trong hàng đợi.

Để ghi nhận trạng thái duyệt các đỉnh của đồ thị, ta cũng vẫn sử dụng mảng chuaxet[] gồm n phần tử thiết lập giá trị ban đầu là TRUE. Nếu đỉnh u của đồ thị đã được duyệt, giá trị chuaxet[u] sẽ nhận giá trị FALSE. Thuật toán dừng khi hàng đợi rỗng. Hình 5.7. dưới đây mô tả chi tiết thuật toán BFS(u).

Biểu diễn thuật toán:

Thuật toán BFS(u):

Bước 1(Khởi tạo):

Queue = ∅; //*tạo lập hàng đợi rỗng*

Push(Queue,u); //*đưa u vào hàng đợi*

chuaxet[u] = False; //*ghi nhận đỉnh u đã xét*

Bước 2 (Lặp):

while (Queue ≠ ∅) do //*lặp đến khi hàng đợi rỗng*

s = Pop(Queue); //*đưa s ra khỏi hàng đợi*

<*Thăm đỉnh s*>;

for each t∈Ke(s) do //*duyệt trên danh sách ke(s)*

if (chuaxet[t]) then //*nếu t chưa xét*

Hình 5.7. Thuật toán BFS(u).

Độ phức tạp tính toán:

Độ phức tạp thuật toán BFS(u) phụ thuộc vào phương pháp biểu diễn đồ thị. Độ phức tạp thuật toán BFS(u) theo các dạng biểu diễn đồ thị như sau:

- Độ phức tạp thuật toán là $O(n^2)$ trong trường hợp đồ thị biểu diễn dưới dạng ma trận kề, với n là số đỉnh của đồ thị.
- Độ phức tạp thuật toán là O(n.m) trong trường hợp đồ thị biểu diễn dưới dạng danh sách cạnh, với n là số đỉnh của đồ thị, m là số cạnh của đồ thị.
- Độ phức tạp thuật toán là O(max(n, m)) trong trường hợp đồ thị biểu diễn dưới dạng danh sách kề, với n là số đỉnh của đồ thị, m là số cạnh của đồ thị.

Bạn đọc tự chứng minh hoặc có thể tham khảo trong các tài liệu [1, 2, 3].

4.3.3. Ứng dụng của thuật toán DFS và BFS

Có rất nhiều ứng dụng khác nhau của thuật toán DFS và BFS trên đồ thị. Trong khuôn khổ của tài liệu này, ta chỉ xem xét đến một vài ứng dụng cơ bản. Những ứng dụng cụ thể hơn bạn đọc có thể tìm thấy rất nhiều trong các tài liệu khác nhau. Những ứng dụng cơ bản của thuật toán DFS và BFS được đề cập bao gồm:

- Duyệt tất cả các đỉnh của đồ thị.
- Duyệt tất cả các thành phần liên thông của đồ thị.
- Tìm đường đi từ đỉnh s đến đỉnh t trên đồ thị.
- Duyệt các đỉnh trụ trên đồ thị vô hướng.
- Duyệt các cạnh cầu trên đồ thị vô hướng.
- Định chiều đồ thị vô hướng.
- Duyệt các đỉnh rẽ nhánh của cặp đỉnh s, t.
- Xác định tính liên thông mạnh trên đồ thị có hướng.
- Xây dựng cây khung trên đồ thị vô hướng.
- Tìm chu trình trên đồ thị vô hướng, có hướng.
- Định chiều đồ thị…

4.4. Đồ thị Euler

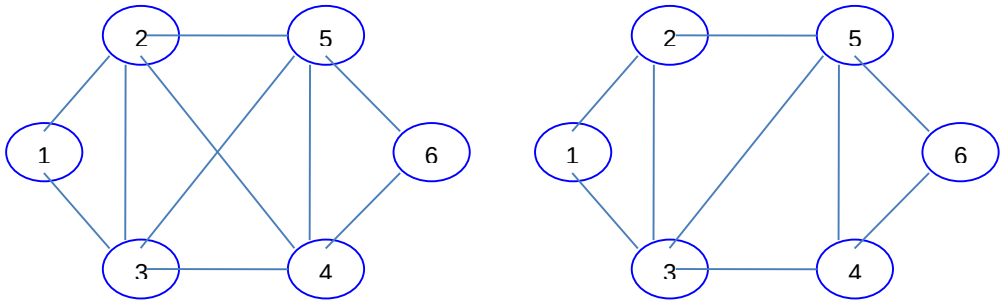
Đồ thị Euler là một dạng đồ thị có nhiều ứng thực tế được định nghĩa như sau:

Định nghĩa.

- Chu trình đơn trong đồ thị G =<V,E> đi qua tất cả các cạnh của đồ thị được gọi là chu trình Euler.
- Đường đi đơn trong G =<V,E> đi qua tất cả các cạnh của đồ thị được gọi là đường đi Euler.
- Đồ thị được G =<V,E> có chu trình Euler được gọi là đồ thị Euler.
- Đồ thị được G =<V,E> có đường đi Euler được gọi là đồ thị nửa Euler.

Ví dụ đồ thị trong Hình 5.8 :

- Đồ thị G1 là Euler vì G1 có chu trình Euler: 1-2-3-4-5-2-4-6-5-3-1.
- Đồ thị G2 là nửa Euler nhưng không là Euler vì G2 có đường đi Euler: 2-1-3-2-5-3-4-6-5-4.



Hình 5.8. Ví dụ về đồ thị Euler, nửa Euler.

Định lý:

- Đồ thị vô hướng liên thông G =<V, E> là Euler khi và chỉ khi tất cả các đỉnh của V đều có bậc chẵn.

- Đồ thị vô hướng liên thông G =<V, E> là nửa Euler nhưng không là Euler khi và chỉ khi G có đúng hai đỉnh bậc lẻ. Đường đi Euler xuất phát tại một đỉnh bậc lẻ và kết thúc tại đỉnh bậc lẻ còn lại.
- Đồ thị có hướng liên thông yếu G =<V, E> là Euler khi và chỉ khi mọi đỉnh của G đều có bán bậc ra bằng bán đỉnh bậc vào của đỉnh.
- Đồ thị có hướng liên thông yếu G =<V, E> là nửa Euler nhưng không là Euler khi và chỉ khi tồn tại đúng hai đỉnh u, v sao cho bán bậc ra của u trừ bán đỉnh bậc vào của u bằng bán đỉnh bậc vào của v trừ bán đỉnh bậc ra của v và bằng 1. Các đỉnh khác u, v còn lại có bán bậc ra bằng bán bậc vào. Đường đi Euler trên đồ thị sẽ xuất phát tại u và kết thúc tại v.

Dựa vào điều kiện cần và đủ để đồ thị G =<V,E> là Euler hay nửa Euler, ta xây dựng thuật toán kiểm tra và tìm một đường đi, hoặc chu trình Euler trên đồ thị vô hướng và có hướng. Điểm khác biệt giữa thuật toán kiểm tra G=<V,E> là Euler hay nửa Euler chỉ là việc xem xét bậc của đỉnh hay bán bậc của đỉnh. Điểm khác biệt giữa thuật toán tìm đường đi và chu trình Euler chỉ là việc xem xét đỉnh nào là đỉnh bắt đầu xây dựng đường đi hay chu trình. Để thấy rõ được sự khác biệt này, ta xem xét từng trường hợp cho đồ thị vô hướng và đồ thị có hướng.

4.4.1. Thuật toán tìm một chu trình Euler trên đồ thị vô hướng

Để xác định đồ thị vô hướng G =<V,E> có phải là đồ thị Euler hay không ta chỉ cần dựa vào tính chất của ma trận kề, danh sách cạnh, danh sách kề biểu diễn đồ thị. Thuật toán được thể hiện trong Hình 5.9. Thuật toán tìm một chu trình Euler trên đồ thị vô hướng liên thông được thể hiện trong Hình 5.10.

a) Biểu diễn thuật toán

```
Thuật toán Check_Euler(G=<V, E>):
begin
    for each u∈V do //duyệt trên tập các đỉnh trong V
        if ( deg(u) mode 2 ) //nếu bậc của đỉnh là lẻ
            return false; //trả lại giá trị false
```

Hình 4.9. Thuật toán kiểm tra đồ thị vô hướng có là Euler?

Thuật toán Euler-Cycle(u):

Bước 1 (Khởi tạo) :

stack = ∅ ; //*khởi tạo một stack bắt đầu là* ∅

CE = ∅ ; //*khởi tạo mảng CE bắt đầu là* ∅

Push (stack, u) ; //*đưa đỉnh u vào ngăn xếp*

Bước 2 (Lặp):

while (stack≠∅) do { //*lặp cho đến khi stack rỗng*

s = get(stack); //*lấy s đỉnh ở đầu ngăn xếp*

if (Ke(s) ≠ ∅) then { //*nếu danh sách Ke*(s) *chưa rỗng*

t =< *Đỉnh đầu tiên trong Ke*(s)>;

Push(stack, t)' //*đưa t vào stack*

E = E \ (s,t); // *loại bỏ cạnh* (s,t)

}

Hình 4.10. Thuật toán tìm một chu trình Euler trên đồ thị.

b) **Độ phức tạp thuật toán**:

Bạn đọc tự tìm hiểu và chứng minh độ phức tạp thuật toán trong các tài liệu tham khảo liên quan.

4.4.2. Thuật toán tìm một chu trình Euler trên đồ thị có hướng

Tìm chu trình Euler trên đồ thị vô hướng và đồ thị có hướng khác biệt nhau duy nhất ở việc kiểm tra đồ thị có hướng là Euler hay không? Để kiểm tra đồ thị có hướng G=<V,E> là Euler ta chỉ cần kiểm duyệt bán bậc ra và bán bậc vào của đỉnh. Nếu tất cả các đỉnh đều có bán đỉnh bậc ra và bán đỉnh bậc vào thì ta kết luận đồ thị là Euler. Nếu tồn tại một đỉnh có bán bậc ra khác bán bậc vào ta kết luận đồ thị không là Euler. Thuật toán kiểm tra đồ thị có hướng G=<V,E> là đồ thị Euler được mô tả trong Hình 5.11. Thuật toán tìm một chu trình Euler trên đồ thị có hướng bắt đầu tại đỉnh u được giữ nguyên trong Hình 4.10.

a) Biểu diễn thuật toán

```
Thuật toán Check_Euler(G=<V, E>):
begin
    for each u∈V do //duyệt trên tập các đỉnh trong V
        if ( deg+(u) !=deg-(u)) //nếu bán bậc ra khác bán bậc vào của u
            return false; //trả lại giá trị false
```

Hình 4.11. Thuật toán kiểm tra đồ thị vô hướng có là Euler?

b) **Độ phức tạp thuật toán**:

Bạn đọc tự tìm hiểu và chứng minh độ phức tạp thuật toán trong các tài liệu tham khảo liên quan.

4.4.3. Thuật toán tìm một đường đi Euler trên đồ thị vô hướng

Tìm một đường đi Euler cũng giống như tìm một chu trình Euler trên đồ thị vô hướng. Điểm khác biệt duy nhất giữa hai thuật toán này là đỉnh xuất phát để tìm đường đi hay chu trình Euler. Đối với đồ thị Euler ta có thể xây dựng chu trình Euler tại bất kỳ đỉnh nào thuộc tập đỉnh. Đối với đồ thị là nửa Euler nhưng không là Euler, đỉnh để xây dựng đường đi Euler là một trong hai đỉnh có bậc lẻ. Thuật toán kiểm tra một đồ thị vô hướng liên thông là nửa Euler nhưng không là Euler được thể hiện như Hình 5.12.

a) Biểu diễn thuật toán

```
Thuật toán Check_Semi_Euler(G=<V, E>):
Begin
    int   so_dinh_le=0;
    for each u∈V do //duyệt trên tập các đỉnh trong V
        if ( deg (u) %2!=0) //bậc của đỉnh u lẻ
            so_dinh_le = so_dinh_le +1; //tăng số đỉnh bậc lẻ
        endif;
    endfor;
    if (so_dinh_le==2) //nếu số đỉnh bậc lẻ là 2
```

Hình 4.12. Thuật toán kiểm tra đồ thị vô hướng liên thông là nửa Euler

b) Độ phức tạp thuật toán

Bạn đọc tự tìm hiểu và chứng minh độ phức tạp thuật toán trong các tài liệu tham khảo liên quan.

4.4.4. Thuật toán tìm một đường đi Euler trên đồ thị có hướng

Tìm một đường đi Euler cũng giống như tìm một chu trình Euler trên đồ thị có hướng. Điểm khác biệt duy nhất giữa hai thuật toán này là đỉnh xuất phát để tìm đường đi hay chu trình Euler. Đối với đồ thị Euler ta có thể xây dựng chu trình Euler tại bất kỳ đỉnh nào thuộc tập đỉnh. Đối với đồ thị là nửa Euler nhưng không là Euler, đỉnh để xây dựng đường đi Euler là m đỉnh u có deg+(u)-deg-(u) =1. Thuật toán kiểm tra một đồ thị có hướng liên thông yếu là nửa Euler nhưng không là Euler được thể hiện như Hình 5.13.

a) Biểu diễn thuật toán

Thuật toán Check_Semi_Euler(G=<V, E>):

Begin

int s, t, dem1 = 0, dem2 = 0;

for each u∈V do //*duyệt trên tập các đỉnh trong V*

if ($deg^+(u)- deg^-(u)==1$){ //*tìm hiệu bán bậc của đỉnh u*

dem1++; s =u; //*ghi nhận đỉnh u có* $deg^+(u)- deg^-(u)==1$

endif;

else if ($deg^-(u)- deg^+(u)==1$){

dem2++; t =u; //*ghi nhận đỉnh u có* $deg^-(u)- deg^+(u)==1$

endesle;

Hình 4.13. Thuật toán kiểm tra đồ thị có hướng là nửa Euler.

b) Độ phức tạp thuật toán

Bạn đọc tự tìm hiểu và chứng minh độ phức tạp thuật toán trong các tài liệu tham khảo liên quan.

4.5. Bài toán xây dựng cây khung của đồ thị

Trong mục này ta đề cập đến một loại đồ thị đơn giản nhất đó là cây. Cây được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau của tin học như tổ chức các thư mục, lưu trữ dữ liệu, biểu diễn tính toán, biểu diễn quyết định và tổ chức truyền tin. Ta có thể tiếp cận cây bằng lý thuyết đồ thị như dưới đây.

Định nghĩa 1. Ta gọi cây là đồ thị vô hướng liên thông không có chu trình. Đồ thị không liên thông được gọi là rừng. Như vậy, rừng là đồ thị mà mỗi thành phần liên thông của nó là một cây.

Định lý 1. Giả sử T= <V, E> là đồ thị vô hướng n đỉnh. Khi đó những khẳng định sau là tương đương

a) T là một cây.
b) T không có chu trình và có n-1 cạnh.
c) T liên thông và có đúng n-1 cạnh.
d) T liên thông và mỗi cạnh của nó đều là cầu.
e) Giữa hai đỉnh bất kỳ của T được nối với nhau bởi đúng một đường đi đơn.
f) T không chứa chu trình nhưng hễ cứ thêm vào nó một cạnh ta thu được đúng một chu trình.

Định nghĩa 2. Cho G =<V, E> là đồ thị vô hướng liên thông. Ta gọi đồ thị con H=<V,T> là một cây khung của G nếu H là một cây và T⊆E.

Tiếp cận cây bằng lý thuyết đồ thị, người ta qua tâm đến hai bài toán cơ bản về cây:

Bài toán 1. Cho đồ thị vô hướng G =<V,E>. Hãy xây dựng một cây khung của đồ thị bắt đầu tại đỉnh u∈V.

Bài toán 2. Cho đồ thị vô hướng G =<V,E> có trọng số. Hãy xây dựng cây khung có độ dài nhỏ nhất.

Bài toán 1 được giải quyết bằng các thuật toán tìm kiếm cơ bàn: tìm kiếm theo chiều rộng (BFS) hoặc thuật toán tìm kiếm theo chiều sâu (DFS). Bài toán 2 được giải quyết bằng thuật toán Kruskal hoặc PRIM. Dưới đây là nội dung các thuật toán.

4.5.1. Xây dựng cây khung của đồ thị bằng thuật toán DFS

Khi ta thực hiện thủ tục tìm kiếm theo chiều sâu bắt đầu tại đỉnh u∈V, ta nhận được tập đỉnh v∈V có cùng thành phần liên thông với u. Nếu DFS(u) = V thì ta kết luận đồ thị liên thông và phép duyệt DFS(u) đi qua đúng n-1 cạnh. Nếu Nếu DFS(u) ≠ V thì ta kết luận đồ thị không liên thông. Chính vì vậy, ta có thể sử dụng phép duyệt DFS(u) để xây dựng cây khung của đồ thị. Trong mỗi bước của thuật toán DFS, xuất phát tại đỉnh u ta sẽ thăm được đỉnh v và ta kết nạp cạnh (u,v) vào tập cạnh của cây khung. Các bước tiếp theo được tiến hành tại đỉnh v cho đến khi kết thúc thuật toán DFS. Thuật toán được xây dựng dựa vào ngăn xếp được mô tả chi tiết trong Hình 4.14.

a) Biểu diễn thuật toán

```
Thuật toán Tree-DFS(u):
Begin
    Bước 1 (Khởi tạo):
        T = ∅; //tập cạnh cây khung ban đầu.
        stack = ∅; //thiết lập stack rỗng;
        Push(stack, u); //đưa u vào stack;
        chuaxet[u] = False;//bật trạng thái đã xét của đỉnh u
    Bước 2 (Lặp):
        while (stack≠∅ ) do { //lặp cho đến khi stack rỗng
            s = Pop(stack); //lấy s ra khỏi stack
            for each t∈Ke(s) do { //lặp trên danh sách Ke(s)
                if (chuaxet[t] ) then { //nếu đỉnh t chuaxet
                    Push(stack, s);// đưa s vào stack trước
                    Push(stack, s);// đưa t vào stack sau
                    T = T ∪ (s,t); //kết nạp (s,t) vào cây khung
```

Hình 4.14. Xây dựng cây khung của đồ thị bằng thuật toán DFS.

b) Độ phức tạp thuật toán

Độ phức tạp thuật toán Tree-DFS(u) đúng bằng độ phức tạp thuật toán DFS(u).

4.5.2. Xây dựng cây khung của đồ thị bằng thuật toán BFS

Để tìm một cây khung trên đồ thị vô hướng liên thông ta có thể sử dụng kỹ thuật tìm kiếm theo chiều rộng. Giả sử ta cần xây dựng một cây bao trùm xuất phát tại đỉnh *u* nào đó. Trong cả hai trường hợp, mỗi khi ta đến được đỉnh *v* tức (*chuaxet[v] = False*) từ đỉnh u thì cạnh (*u,v*) được kết nạp vào cây khung.

a) Biểu diễn thuật toán

Thuật toán Tree-BFS(u):

Begin

Bước 1 (Khởi tạo):

T = ∅; //*tập cạnh cây khung ban đầu.*

Queue = ∅; //*thiết lập hàng đợi ban đầu;*

Push(Queue, u); //*đưa u vào hàng đợi;*

chuaxet[u] = False;//*bật trạng thái đã xét của đỉnh u*

Bước 2 (Lặp):

while (Queue≠∅) do { //*lặp cho đến khi hàng đợi rỗng*

s = Pop(Queue); //*lấy s ra khỏi hàng đợi*

for each t∈Ke(s) do { //*lặp trên danh sách Ke(s)*

if (chuaxet[t]) then { //*nếu đỉnh t chuaxet*

Push(Queue, t);// *đưa t vào hàng đợi*

Hình 4.15. Thuật toán Tree-BFS

b) Độ phức tạp thuật toán

Độ phức tạp thuật toán Tree-BFS(u) đúng bằng độ phức tạp thuật toán BFS(u).

4.5.3. Xây dựng cây khung nhỏ nhất của đồ thị bằng thuật toán Kruskal

Bài toán tìm cây khung nhỏ nhất là một trong những bài toán tối ưu trên đồ thị có ứng dụng trong nhiều lĩnh vực khác nhau của thực tế. Bài toán được phát biểu như dưới sau. Cho $G=<V, E>$ là đồ thị vô hướng liên thông với tập đỉnh $V = \{1, 2, \ldots, n\}$ và tập cạnh E gồm m cạnh. Mỗi cạnh e của đồ thị được gán với một số không âm $c(e)$ được gọi là độ dài cạnh. Giả sử $H=<V, T>$ là một cây khung của đồ thị G. Ta gọi độ dài $c(H)$ của cây khung H là tổng độ dài các cạnh: $c(H) = \sum_{e \in T} c(e)$. Bài toán được đặt ra là, trong số các cây khung của đồ thị hãy tìm cây khung có độ dài nhỏ nhất của đồ thị.

a) Biểu diễn thuật toán

Thuật toán Kruskal xây dựng cây khung nhỏ nhất cho đồ thị vô hướng liên thông có trọng số được thực hiện thep mô hình tham lam trong Hình 4.16.

Thuật toán Kruskal:

Begin

Bước 1 (Khởi tạo):

T = ∅; //*Khởi tạo tập cạnh cây khung là ∅*

d(H) = 0; //*Khởi tạo độ dài nhỏ nhất cây khung là 0*

Bước 2 (Sắp xếp):

<Sắp xếp các cạnh của đồ thị theo thứ tự giảm dần của trọng số>;

Bước 3 (Lặp):

while (|T<n-1| && E≠∅) do { // *Lặp nếu E≠∅ và |T|<n-1*

e = <Cạnh có độ dài nhỏ nhất>;

E = E \ {e}; //*Loại cạnh e ra khỏi đồ thị*

if (T∪{e} không tạo nên chu trình) then {

T = ∪{e}; // *Kết nạp e vào tập cạnh cây khung*

Hình 4.16. Thuật toán Kruskal tìm cây khung nhỏ nhất

b) Độ phức tạp thuật toán

Độ phức tạp thuật toán là O(E.log(V)), vơi E là số cạnh và V là số đỉnh của đồ thị.

Bạn đọc tự tìm hiểu và chứng minh trong các tài liệu tham khảo liên quan.

4.5.4. Xây dựng cây khung nhỏ nhất của đồ thị bằng thuật toán PRIM

Thuật toán Kruskal làm việc kém hiệu quả đối với những đồ thị có số cạnh khoảng m=n(n-1)/2. Trong những tình huống như vậy, thuật toán Prim tỏ ra hiệu quả hơn. Thuật toán Prim còn được mang tên là người láng giềng gần nhất. Trong thuật toán này, bắt đầu tại một đỉnh tuỳ ý *s* của đồ thị, nối *s* với đỉnh *y* sao cho trọng số cạnh *graph[s, y]* là nhỏ nhất. Tiếp theo, từ đỉnh *s* hoặc *y* tìm cạnh có độ dài nhỏ nhất, điều này dẫn đến đỉnh thứ ba *z* và ta thu được cây bộ phận gồm *3* đỉnh *2* cạnh. Quá trình được tiếp tục cho tới khi ta nhận được cây gồm *n-1* cạnh, đó chính là cây bao trùm nhỏ nhất cần tìm. Thuật toán Prim được mô tả trong Hình 5.17.

a) Biểu diễn thuật toán

Thuật toán PRIM (s):

Begin:

Bước 1 (Khởi tạo):

V_H = {s}; //Tập đỉnh cây khung thiết lập ban đầu là s

V = V\{s}; //Tập đỉnh V được bớt đi s

T = ∅; //Tập cạnh cây khung thiết lập ban đầu là ∅

d(H) = 0; //Độ dài cây khung được thiết lập là 0

Bước 2 (Lặp):

while (V≠ ∅) do {

e = <u, v>: cạnh có độ dài nhỏ nhất thỏa mãn u∈V, v∈V_H;

d(H) = d(H) + d(e); // Thiết lập đồ dài cây khung nhỏ nhất

T = T ∪ {e}; //Kết nạp e vào cây khung

V = V \{u}; // Tập đỉnh V bớt đi đỉnh u

Hình 4.16. Thuật toán PRIM

b) Độ phức tạp thuật toán

Độ phức tạp thuật toán là $O(V^2)$, với V là số đỉnh của đồ thị. Bạn đọc tự tìm hiểu phương pháp chứng minh độ phức tạp thuật toán trong các tài liệu liên quan.

4.6. Bài toán tìm đường đi ngắn nhất

Xét đồ thị $G=<V, E>$; trong đó $|V| = n$, $|E| = m$. Với mỗi cạnh (u, v)∈E, ta đặt tương ứng với nó một số thực A[u][v] được gọi là trọng số của cạnh. Ta sẽ đặt $A[u,v]=\infty$ nếu (u, v)∉E. Nếu dãy $v_0, v_1, \ldots, v_k$ là một đường đi trên G thì $\sum_{i=1}^{p} A[v_{i-1}, v_i]$ được gọi là độ dài của đường đi.

Bài toán tìm đường đi ngắn nhất trên đồ thị dưới dạng tổng quát có thể được phát biểu dưới dạng sau: tìm đường đi ngắn nhất từ một đỉnh xuất phát $s \in V$ (đỉnh nguồn) đến đỉnh cuối $t \in V$ (đỉnh đích). Đường đi như vậy được gọi là đường đi ngắn nhất từ *s* đến *t*, độ dài của đường đi *d(s,t)* được gọi là khoảng cách ngắn nhất từ *s* đến *t* (trong trường hợp tổng quát *d(s,t)* có thể âm). Nếu như không tồn tại đường đi từ *s* đến *t* thì độ dài đường đi $d(s,t)=\infty$. Dưới đây là một số thể hiện cụ thể của bài toán.

Trường hợp 1. Nếu *s* cố định và *t* thay đổi, khi đó bài toán được phát biểu dưới dạng tìm đường đi ngắn nhất từ *s* đến tất cả các đỉnh còn lại trên đồ thị. Đối với đồ thị có trọng số không âm, bài toán luôn có lời giải bằng thuật toán Dijkstra. Đối với đồ thị

có trọng số âm nhưng không tồn tại chu trình âm, bài toán có lời giải bằng thuật toán Bellman-Ford. Trong trường hợp đồ thị có chu trình âm, bài toán không có lời giải.

Trường hợp 2. Nếu s thay đổi và t cũng thay đổi, khi đó bài toán được phát biểu dưới dạng tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị. Bài toán luôn có lời giải trên đồ thị không có chu trình âm. Đối với đồ thị có trọng số không âm, bài toán được giải quyết bằng cách thực hiện lặp lại n lần thuật toán Dijkstra. Đối với đồ thị không có chu trình âm, bài toán có thể giải quyết bằng thuật toán Floyd-Warshall.

4.6.1. Thuật toán Dijkstra

Thuật toán tìm đường đi ngắn nhất từ đỉnh *s* đến các đỉnh còn lại được Dijkstra đề nghị áp dụng cho trường hợp đồ thị có hướng với trọng số không âm. Thuật toán được thực hiện trên cơ sở gán tạm thời cho các đỉnh. Nhãn của mỗi đỉnh cho biết cận trên của độ dài đường đi ngắn nhất tới đỉnh đó. Các nhãn này sẽ được biến đổi (tính lại) nhờ một thủ tục lặp, mà ở mỗi bước lặp một số đỉnh sẽ có nhãn không thay đổi, nhãn đó chính là độ dài đường đi ngắn nhất từ s đến đỉnh đó. Thuật toán Dijkstra tìm đường đi ngắn nhất từ s đến tất cả các đỉnh còn lại của đồ thị được mô tả chi tiết trong Hình 5.17.

a) Biểu diễn thuật toán

Thuật toán Dijkstra (s): //s∈ V là một đỉnh bất kỳ của G = <V,E>

Begin

Bước 1 (Khởi tạo):

d[s]=0; //*Gán nhãn của đỉnh s là 0*

T = V\{s}; // *T là tập đỉnh có nhãn tạm thời*

for each v∈ V do { //*Sử dụng s gán nhãn cho các đỉnh còn lại*

d[v] = A[s,v];

truoc[v]=s;

endfor;

Bước 2 (Lặp):

while (T≠ ∅) do {

Tìm đỉnh u∈T sao cho d[u] = min { d[z] | z∈T};

T= T\{u}*; //cố định nhãn đỉnh u*

for each v∈ T do { //Sử dụng u, gán nhãn lại cho các đỉnh

Hình 4.17. Thuật toán Dijkstra

b) Độ phức tạp thuật toán

Độ phức tạp thuật toán là $O(V^2)$, trong đó V là số đỉnh của đồ thị. Bạn đọc tự tìm hiểu và chứng minh độ phức tạp thuật toán Dijkstra trong các tài liệu liên quan.

4.6.2. Thuật toán Bellman-Ford

Thuật toán Bellman-Ford dùng để tìm đường đi ngắn nhất trên đồ thị không có chu trình âm. Do vậy, trong khi thực hiện thuật toán Bellman-Ford ta cần kiểm tra đồ thị có chu trình âm hay không. Trong trường hợp đồ thị có chu trình âm, bài toán sẽ không có lời giải. Thuật toán được thực hiện theo k = n - 2 vòng lặp trên tập đỉnh hoặc tập cạnh tùy thuộc vào dạng biểu diễn của đồ thị. Nếu đồ thị được biểu diễn dưới dạng ma trận kề, độ phức tạp thuật toán là $O(V^3)$, với V là số đỉnh của đồ thị. Trong trường hợp đồ thị được biểu diễn dưới dạng danh sách cạnh, độ phức tạp thuật toán là O(V.E), với V là số đỉnh của đồ thị, E là số cạnh của đồ thị. Thuật toán được mô tả chi tiết trong Hình 5.18 cho đồ thị biểu diễn dưới dạng ma trận kề.

a) **Biểu diễn thuật toán**

Thuật toán Bellman-Ford (s): *//s∈ V là đỉnh bất kỳ của đồ thị*

Begin:

Bước 1 (Khởi tạo):

for v∈V do { *//Sử dụng s gán nhãn cho các đỉnh v∈V*

D[v] = A[s][v];

Truoc[v] = s;

}

Bước 2 (Lặp) :

D[s] = 0; K=1;

while (K<=N-2) { //N-2 vòng lặp

for v∈V\{s} do { *//Lấy mỗi đỉnh v∈V\s*

for u∈V do { *//Gán nhãn cho v*

if (D[v] > D[u] + A[u][v]) {

Hình 4.18. Thuật toán Bellman-Ford

b) Độ phức tạp thuật toán

Độ phức tạp thuật toán là O(VE), trong đó V, E là số đỉnh và số cạnh của đồ thị. Bạn đọc tự tìm hiểu và chứng minh độ phức tạp thuật toán Bellman-Ford trong các tài liệu liên quan.

4.6.3. Thuật toán Floyd-Warshall

Để tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị, chúng ta có thể sử dụng *V* lần thuật toán *Ford_Bellman* hoặc *Dijkstra* (trong trường hợp trọng số không âm). Tuy nhiên, trong cả hai thuật toán được sử dụng đều có độ phức tạp tính toán lớn (chí ít là $O(V^3)$). Trong trường hợp tổng quát, người ta thường dùng thuật toán *Floyd-Warshall*. Thuật toán Floy được mô tả chi tiết trong Hình 5.19.

a) Biểu diễn thuật toán

Thuật toán Floyd-Warshall:
Begin:

Bước 1 (Khởi tạo):

```
for (i=1; i≤ n; i++) {
    for (j =1; j≤ n; j++) {
        d[i,j] = a[i, j];
        p[i,j] = i;
    }
}
```

Bước 2 (lặp) :

```
for (k=1; k≤ n; k++) {
    for (i=1; i≤ n; i++){
        for (j =1; j≤ n; j++) {
            if (d[i,j] > d[i, k] + d[k, j]) {
                d[i, j] = d[i, k] + d[k, j];
```

Hình 4.19. Thuật toán Floyd-Warshall.

b) Độ phức tạp thuật toán

Độ phức tạp thuật toán là $O(V^3)$, trong đó V là số đỉnh của đồ thị. Bạn đọc tự tìm hiểu và chứng minh độ phức tạp thuật toán Floyd-Warshall trong các tài liệu liên quan.

CHƯƠNG 5. CÁC CẤU TRÚC DỮ LIỆU CƠ BẢN

Nội dung của chương này trình bày ba kiểu dữ liệu trừu tượng quan trọng đó là danh sách liên kết, ngăn xếp và hàng đợi. Mỗi kiểu dữ liệu trừu tượng được xây dựng giải quyết lớp các vấn đề cụ thể của khoa học máy tính. Đối với người học, mỗi cấu trúc dữ liệu trừu tượng cần làm chủ được bốn điểm quan trọng sau:

- Định nghĩa cấu trúc dữ liệu ADTs.
- Biểu diễn cấu trúc dữ liệu ADTs.
- Thao tác (phép toán) trên cấu trúc dữ liệu ADTs.
- Ứng dụng của cấu trúc dữ liệu ADTs.

5.1. Danh sách liên kết đơn (Single Linked List)

Như ta đã biết mảng (*array*) là tập có thứ tự các phần tử có cùng chung một kiểu dữ liệu và được tổ chức liên tục nhau trong bộ nhớ. Ưu điểm lớn nhất của mảng là đơn giản và sử lý nhanh nhờ cơ chế tru cập phần tử trực tiếp vào các phần tử của mảng. Hạn chế lớn nhất của mảng là số lượng phần tử không thay đổi gây nên hiện tượng thừa bộ nhớ trong mọt số trường hợp và thiếu bộ nhớ trong một số trường hợp khác. Đối với một số bài toán có dữ liệu lớn, nhiều khi ta không đủ không gian nhớ tự do liên tục để cấp phát cho mảng. Để khắc phục hạn chế này ta có thể xây dựng kiểu dữ liệu danh sách liên kết đơn được định nghĩa, biểu diễn và thao tác như dưới đây.

5.1.1. Định nghĩa danh sách liên kết đơn

Tập hợp các node thông tin được tổ chức rời rạc trong bộ nhớ. Trong đó, mỗi node gồm coa hai thành phần:

- Thành phần dữ liệu (*data*): dùng để lưu trữ thông tin của node.
- Thành phần con trỏ (*pointer*): dùng để liên kết với node dữ liệu tiếp theo.

5.1.2. Biểu diễn danh sách liên kết đơn

Để biểu diễn danh sách liên kết đơn ta sử dụng phương pháp định nghĩa cấu trúc tự trỏ của các ngôn ngữ lập trình. Giả sử thành phần thông tin của mỗi node được định nghĩa như một cấu trúc Item như sau:

```
struct  Item {
        <Kiểu 1>     <Thành viên 1>;
         <Kiểu 2>     <Thành viên 2>;
         . . . . . ;
         <Kiểu N>     <Thành viên N>;
};
```

Khi đó, danh sách liên kết đơn được định nghĩa như sau:

```
struct node {
        Item    infor; //Thành phần thông tin của node;
        struct  node *next; //thành phần con trỏ của node
} *Start; //Start là một danh sách liên kết đơn
```

Hình 5.1. Biểu diễn danh sách liên kết đơn

5.1.3. Thao tác trên danh sách liên kết đơn

Các thao tác trên danh sách liên kết đơn bao gồm:

- Tạo node rời rạc có giá trị value cho danh sách liên kết đơn
- Thêm một node vào đầu danh sách liên kết đơn.
- Thêm một node vào cuối danh sách liên kết đơn.
- Thêm node vào vị trí xác định trong danh sách liên kết đơn.
- Loại node trong sách liên kết đơn.
- Tìm node trong sách liên kết đơn.
- Sắp xếp node trong danh sách liên kết đơn.
- Sửa đổi nội dung node trong sách liên kết đơn.
- Đảo ngược các node trong danh sách liên kết đơn.
- Duyệt các node của danh sách liên kết đơn.

5.1.4. Ứng dụng của danh sách liên kết đơn

Ta có thể sử dụng danh sách liên đơn để giải quyết những vấn đề sau:

- Sử dụng danh sách liên kết đơn trong việc xây dựng các lược đồ quản lý bộ nhớ.
- Sử dụng danh sách liên kết đơn trong việc xây dựng ngăn xếp.
- Sử dụng danh sách liên kết đơn trong việc xây dựng hàng đợi.
- Sử dụng danh sách liên kết đơn biểu diễn cây.
- Sử dụng danh sách liên kết đơn biểu diễn đồ thị.
- Sử dụng danh sách liên kết đơn trong biểu diễn tính toán.

Ví dụ 3.1. Xây dựng phép cộng giữa hai đa thức.

Input : đa thức $P_n(x)$ bậc n và đa thức $Q_m(x)$ bậc m.

Output: đa thức $R = P_n(x) + Q_m(x)$.

Lời giải. Ta có thể sử dụng danh sách liên kết đơn để biểu diễn và xây dựng thuật toán cộng hai đa thức như sau.

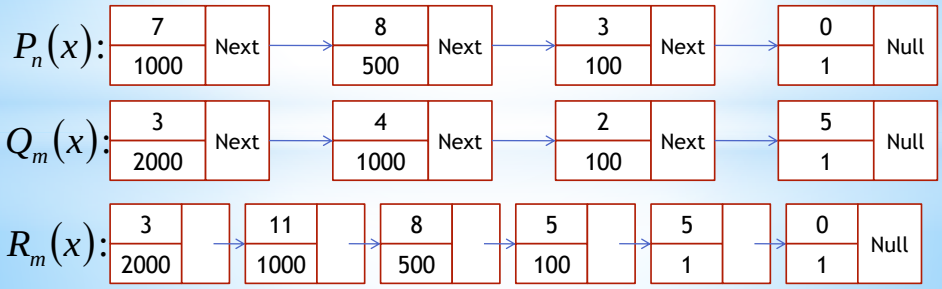
Ví dụ 3.1. Thuật toán cộng hai đa thức $R = P_n(x) + Q_m(x)$.

$$P_n(x) = 7x^{1000} + 8x^{500} + 3x^{100} + 1$$

$$Q_m(x) = 3x^{2000} + 4x^{1000} + 2x^{100} + 5x$$

Biểu diễn mỗi số hạng của đa thức:

```
typedef struct node {
        float hsomueso;
        float ;
        struct node *next;
} *dathuc;
```



Hình 5.2. Biểu diễn đa thức bằng danh sách liên kết đơn

Thuật toán Cong_Dathuc (Dathuc *P, Dathuc *Q):

Bước 1 (Khởi tạo): R = ∅;

Bước 2 (lặp):

```
while ( P≠∅ && Q≠∅ ) {
        if( P→Somu > Q → Somu ) {
                R = R →P; P = P →next;
        }
        else if (P→Somu < Q → Somu ) {
                R = R →Q; Q = Q →next;
        }
        else {   P → heso = P → heso + Q → heso;
                R = R →P; P = P →next; Q = Q →next;
        }
}
```

Bước 3 (Hoàn chỉnh đa thức):

```
if (P ≠∅ ) R = R = R →P;
if (Q ≠∅ ) R = R = R →Q;
```

Bước 4 (Trả lại kết quả):

```
Return (R);
```

Hình 5.3. Thuật toán cộng hai đa thức

5.2. Danh sách liên kết kép (double linked list)

Hạn chế lớn nhất đối với danh sách liên kết đơn là vấn đề tìm kiếm. Phương pháp tìm kiếm duy nhất ta có thể cài đặt được trên danh sách liên kết đơn là tìm kiếm tuyến tính. Từ một node bất kỳ, hoặc ta quay lại từ node đầu tiên hoặc chỉ được phép tìm theo hướng con trỏ next. Để hạn chế điều này ta có thể xây dựng kiểu dữ liệu danh sách liên kết kép như sau.

5.2.1. Định nghĩa

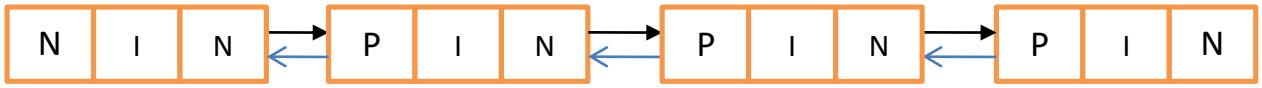
Danh sách liên kết kép là tập các node dữ liệu được tổ chức rời rạc nhau trong bộ nhớ, mỗi node gồm có ba thành phần:

- Thành phần thông tin (infor): dùng để lưu trữ thông tin của node.
- Thành phần liên kết trước (next): dùng để liên kết với node dữ liệu phía trước.
- Thành phần liên kết sau (previous): dùng để liên kết với node dữ liệu phía sau.

5.2.2. Biểu diễn

Sử dụng phương pháp biểu diễn đệ qui của các cấu trúc tự trỏ ta định nghĩa danh sách liên kết kép như sau:

```
struct node { //định nghĩa node
        Item   Infor; //thành phần dữ liệu của node
        struct  node *next; //thành phần con trỏ trước
        struct  node *prev; //thành phần con trỏ sau
}*start; //đây là danh sách liên kết kép
```



Hình 5.4. Biểu diễn danh sách liên kết kép.

5.2.3. Các thao tác trên danh sách liên kết kép

Các thao tác trên danh sách liên kết kép bao gồm:

- Tạo node rời rạc có giá trị value cho danh sách liên kết kép.
- Thêm node vào đầu danh sách liên kết kép.
- Thêm node vào cuối danh sách liên kết kép.
- Thêm node vào vị trí pos trong danh sách liên kết kép.
- Loại node tại vị trí pos của danh sách liên kết kép.
- Sắp xếp nội dung các node của danh sách liên kết kép.
- Tìm kiếm node có giá trị value trên danh sách liên kết kép.

- Duyệt trái danh sách liên kết kép.
- Duyệt phải danh sách liên kết kép.
- Đảo ngược các node trong danh sách liên kết kép.

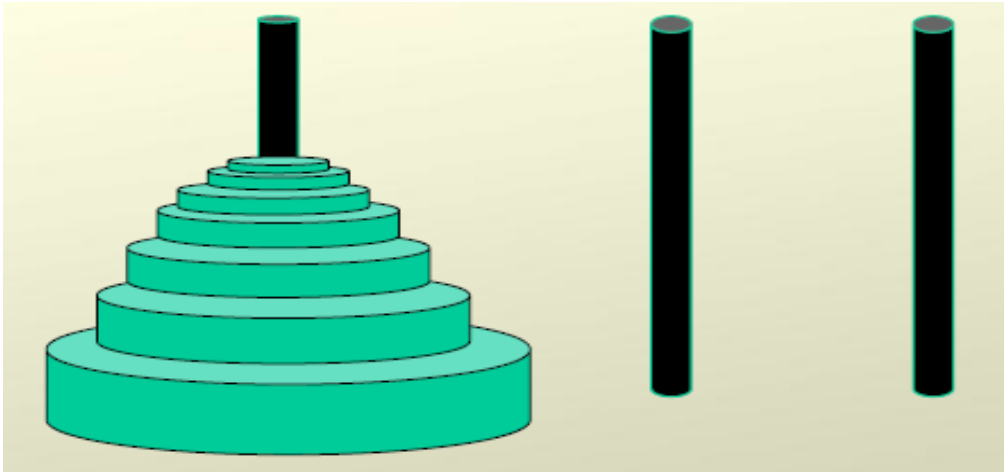
5.3. Ngăn xếp (Stack)

Ngăn xếp được hiểu như một cái ngăn để xếp. Ngăn xếp được ứng dụng trong nhiều lĩnh vực quan trọng khác nhau của khoa học máy tính. Trong mục này, ta sẽ xem xét phương pháp xây dựng kiểu dữ liệu ngăn xếp và ứng dụng của ngăn xếp.

5.3.1. Định nghĩa ngăn xếp

Tập hợp các node thông tin được tổ chức liên tục hoặc rời rạc nhau trong bộ nhớ và thực hiện theo cơ chế vào trước ra sau FILO (First – In – Last – Out).

Ta có thể hình dung stack như chồng đĩa trong bài toán "Tháp Hà Nội". Đĩa có đường kính lớn nhất được xếp trước nhất, đĩa có đường kính nhỏ nhất được xếp sau cùng vào chồng đĩa. Tuy nhiên, khi lấy ra các đĩa có đường kính nhỏ nhất được lấy ra trước nhất, đĩa có đường lớn nhất sẽ được lấy ra sau cùng.



Hình 5.4. Mô tả ngăn xếp

5.3.2. Biểu diễn ngăn xếp

Có hai phương pháp biểu diễn ngăn xếp: biểu diễn liên tục và biểu diễn rời rạc.

- **Biểu diễn liên tục**: các phần tử dữ liệu của ngăn xếp được lưu trữ liên tục nhau trong bộ nhớ.
- **Biểu diễn rời rạc**: các phần tử dữ liệu của ngăn xếp được lưu trữ rời rạc nhau trong bộ nhớ (Danh sách liên kết).

Trong trường hợp biểu diễn liên tục, ta sử dụng mảng để biểu diễn các node dữ liệu của ngăn xếp như sau:

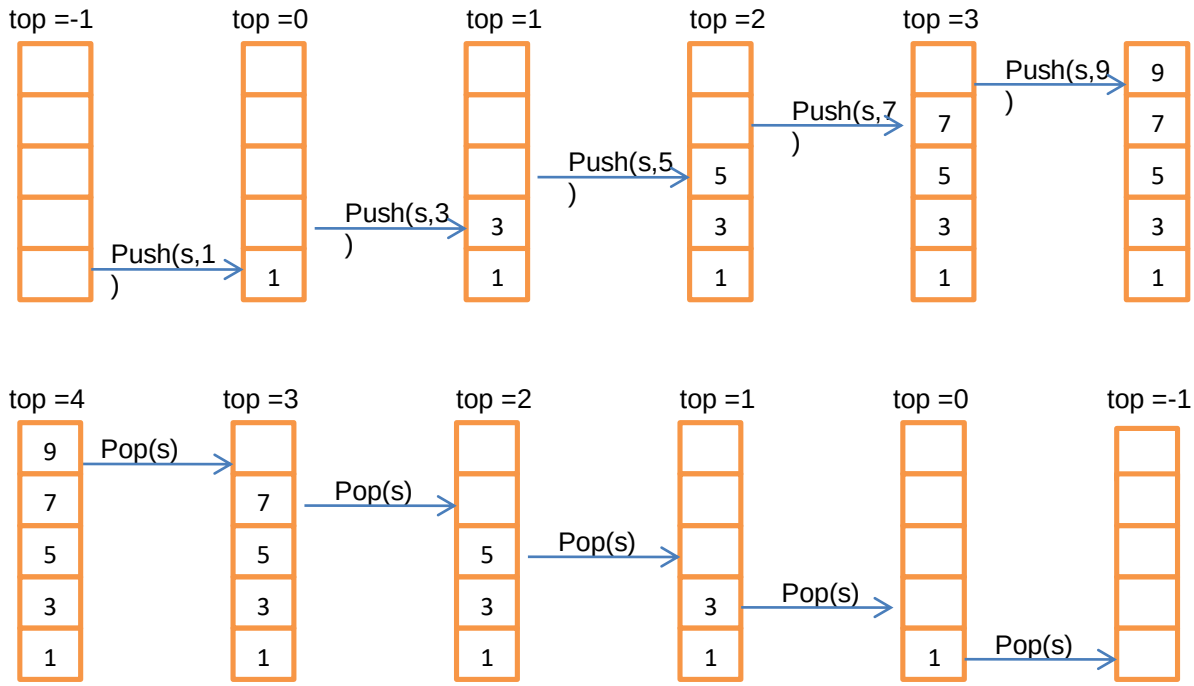
```
typedef struct Stack{
        int     top; //Đỉnh đầu của stack nơi diễn ra mọi thao tác
        int     node[MAX]; //Dữ liệu lưu trữ trong stack gồm MAX phần tử
};
```

Trong trường hợp biểu diễn rời rạc, ta sử dụng danh sách liên kết để biểu diễn rời rạc các node dữ liệu của ngăn xếp như sau:

```
typedef struct node{
        int     infor; //thông tin của node
        struct node *next; //con trỏ trỏ đến node tiếp theo.
      };
```

5.3.3. Các thao tác trên ngăn xếp

Có hai thao tác cơ bản để tạo nên cơ chế vào sau ra trước (LIFO) của stack đó là đưa phần tử vào ngăn xếp (push) và lấy phần tử ra khỏi ngăn xếp. Hai thao tác push và pop đều thực hiện chung tại một vị trí trên ngăn xếp sẽ tạo nên cơ chế LIFO. Hình 3.11 dưới đây sẽ mô tả cơ chế LIFO của ngăn xếp.



Hình 5.5. Cơ chế vào trước ra sau của ngăn xếp

5.3.4. Ứng dụng của ngăn xếp

Ngăn xếp được ứng dụng để giải quyết những vấn đề sau:

- Xây dựng các giải thuật đệ qui: tất cả các giải thuật đệ qui được xây dựng dựa trên cơ chế FIFO của ngăn xếp.

- Khử bỏ các giải thuật đệ qui.
- Biểu diễn tính toán.
- Duyệt cây, duyệt đồ thị…

Ví dụ 3.1. Chuyển đổi biểu thức trung tố về biểu thức hậu tố. Ta vẫn hay làm quen và thực hiện tính toán trên các biểu thức số học trung tố. Ví dụ biểu thức trung tố P = (a+b*c)-(a/b+c). Trong cách viết này các phép toán bao giờ cũng đứng giữa hai toán hạng. Phương pháp tính toán được thực hiện theo thứ tự ưu tiên các phép toán số học. Biểu thức số học hậu tố là phương pháp biểu diễn các phép toán hai ngôi +, -, *, /, ^ đứng sau các toán hạng. Phương pháp biểu diễn được thực hiện theo nguyên tắc như sau:

- Nếu a + b là biểu thức trung tố thì biểu thức hậu tố tương ứng là a b +
- Nếu a - b là biểu thức trung tố thì biểu thức hậu tố tương ứng là a b -
- Nếu a * b là biểu thức trung tố thì biểu thức hậu tố tương ứng là a b *
- Nếu a / b là biểu thức trung tố thì biểu thức hậu tố tương ứng là a b /
- Nếu a ^ b là biểu thức trung tố thì biểu thức hậu tố tương ứng là a b ^ (phép ^ được ký hiệu cho phép lấy lũy thừa)
- Nếu (P) trong đó P là hậu tố thì biểu thức hậu tố tương ứng P.

Biểu thức trung tố	**Biểu thức hậu tố tương đương**
a + b	ab+
a - b	ab-
a * b	ab*
a / b	ab/
a ^ b	ab^
(P)	P

Ví dụ với biểu thức P = (a+b*c)-(a/b+c) sẽ được biến đổi thành biểu thức hậu tố tương đương như dưới đây:

$$\begin{aligned}(a+b*c)-(a/b+c)&=\\&=(a+bc*)-(ab/+c)\\&=(abc*+)-(ab/c+)\\&=abc*+-ab/c+\\&=abc*+ab/c+-\end{aligned}$$

Đối với biểu thức số học hậu tố, không còn các phép toán ‘(‘ , ‘)’, không còn thứ tự ưu tiên các phép toán. Bài toán đặt ra là cho biểu thức trung tố P hãy chuyển đổi P thành biểu diễn hậu tố tương đương với P. Sau khi đã có biểu thức trung tố P, hãy xây dựng thuật toán tính toán giá trị biểu thức hậu tố P. Thuật toán được xây dựng dựa vào ngăn xếp được thể hiện trong Hình 5.6 như dưới đây.

Thuật toán infix-to-postfix (P):

Bước 1 (Khởi tạo)**:**

stack = ∅; *//stack dùng để lưu trữ các phép toán*

Out = ∅;*// out dùng lưu trữ biểu thức hậu tố*

Bước 2 (Lặp) :

For each x∈P do *// duyệt từ trái qua phải biểu thức trung tố P*

2.1. Nếu x =‘ (‘ : Push(stack, x);

2.2. Nếu x là toán hạng: x⇒Out;

2.3. Nếu x ∈ { +, -, *, / ,^}

y = get(stack); *//lấy phép toán ở đầu ngăn xếp*

a) Nếu priority(x)≥priority(y): Push(stack, x);

b) Nếu priority(x)<priority(y):

y = Po(stack); y⇒Out; Push(stack, x);

c) Nếu stack = ∅: Push(stack, x);

2.4. Nếu x =’)’:

y = Pop(stack);

While (y!=‘(’) do

y⇒Out; y = Pop(stack);

EndWhile;

EndFor;

Bước 3(Hoàn chỉnh biểu thức hậu tố)*:*

While (stack ≠ ∅) do

y = Pop(stack); y⇒Out;

EndWhile;

Bước 4(Trả lại kết quả)*:*

Return(Out).

Hình 5.6. Thuật toán chuyển đổi biểu thức trung tố thành biểu thức hậu tố

Kiểm nghiệm thuật toán với P = (a + b*c) – (a/b + c):

x∈P	Bước	Stack	Out
x =‘(‘	2.1	(	∅
x =a	2.2	(	a
x =+	2.3.a	(+	a
x =b	2.2	(+	a b
x =*	2.3.a	(+ *	a b
x = c	2.2	(+ *	a b c
x =‘)‘	2.3	∅	a b c * +
x =-	2.2.c	-	a b c * +
x =‘(‘	2.1	- (	a b c * +
x =a	2.2	- (	a b c * + a
x =/	2.2.a	- (/	a b c * + a
x =b	2.2	- (/	a b c * + a b
x =+	2.3.b	- (+	a b c * + a b/
x =c	2.2	- (+	a b c * + a b/c
x =‘)‘	2.4	∅	a b c * + a b/c + -
P = a b c * + a b/c + -			

Hình 5.7. Kiểm nghiệm thuật toán

Thuật toán tính toán giá trị biểu thức hậu tố:

Bước 1 (Khởi tạo)**:**

stack = ∅;

Bước 2 (Lặp) :

For each x∈P do

2.1. Nếu x là toán hạng:

Push(stack, x);

*2.2. Nếu x ∈ { +, -, *, / }*

a) TH2 = Pop(stack, x);

b) TH1 = Pop(stack, x);

c) KQ = TH1 ⊗ TH2;

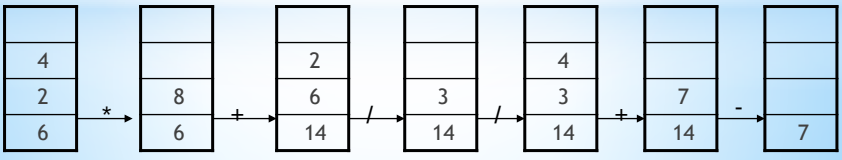
d) Push (stack, KQ);

EndFor;

Bước 4(Trả lại kết quả)*:*

Return(Pop(stack)).

Ví dụ: P = 6 2 4 * + 6 2 / 4 + -



Hình 5.8. Tính toán biểu thức hậu tố

5.4. Hàng đợi (Queue)

Hàng đợi được hiểu là một hàng để đợi. Hàng đợi trong máy tính cũng giống như hàng đợi trong thực tế: hàng đợi mua vé tàu, vé xe, vé máy bay. Hàng đợi ứng dụng trong nhiều lĩnh vực khác nhau của khoa học máy tính. Trong mục này ta sẽ xem xét phương pháp xây dựng hàng đợi cùng với ứng dụng của hàng đợi.

5.4.1. Định nghĩa hàng đợi

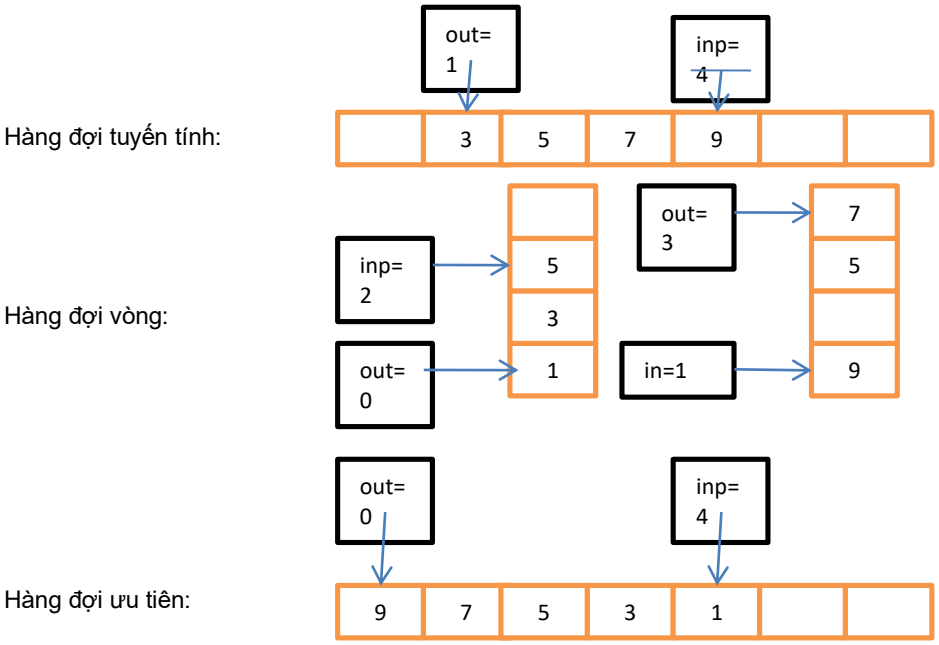
Hàng đợi (queue) là tập các node thông tin được tổ chức liên tục hoặc rời rạc nhau trong bộ nhớ và thực hiện theo cơ chế FIFO (First-In-First-Out).

Hàng đợi tuyến tính (sequntial queue): hàng đợi liên tục được xây dựng theo nguyên tắc có điểm vào (inp) luôn lớn hơn điểm ra (out). Không gian nhớ sẽ không được tái sử dụng sau mỗi phép lấy phần tử ra khỏi hàng đợi.

Hàng đợi vòng (curcular queue): hàng đợi liên tục được xây dựng theo nguyên tắc không gian nhớ sẽ được tái sử dụng sau mỗi phép lấy phần tử ra khỏi hàng đợi.

Hàng đợi ưu tiên (priority queue): hàng đợi được xây dựng theo nguyên tắc phép đưa phần tử vào hàng đợi được xếp ứng với thứ tự ưu tiên của nó.

Hàng đợi hai điểm cuối (double ended queue): hàng đợi được xây dựng theo nguyên tắc phép đưa phần tử vào và lấy phần tử ra khỏi hàng đợi được thực hiện ở hai điểm cuối.



Hình 5.9. Các loại hàng đợi

5.4.2. Biểu diễn hàng đợi

Có hai phương pháp biểu diễn hàng đợi: biểu diễn liên tục và biểu diễn rời rạc. Trong trường hợp biểu diễn liên tục ta sử dụng mảng, trong trường hợp biểu diễn rời rạc ta sử dụng danh sách liên kết.

Biểu diễn liên tục:

```
typedef struct {
        int     inp; // dùng để đưa phần tử vào hàng đợi
        int     out; // dùng để lấy phần tử ra khỏi hàng đợi
        int     node[MAX]; //các node thông tin của hàng đợi
} queue;
```

Biểu diễn rời rạc:

```
struct node { //định nghĩa cấu trúc node
        int infor;//thành phần dữ liệu
        struct node *next; //thành phần liên kết
} *queue;
```

5.4.3. Thao tác trên hàng đợi

Hàng đợi được xây dựng dựa vào hai thao tác cơ bản: đưa phần tử vào hàng đợi (push) và lấy phần tử ra khỏi hàng đợi (pop). Hai thao tác push và pop phối hợp với nhau để tạo nên cơ chế FIFO của hàng đợi.

5.4.4. Ứng dụng của hàng đợi

Hàng đợi được sử dụng trong các ứng dụng sau:

- Dùng để xây dựng các hệ thống lập lịch.
- Dùng để xây dựng các thuật toán duyệt cây.
- Dùng để xây dựng các thuật toán duyệt đồ thị.
- Dùng trong việc biểu diễn tính toán.

Ví dụ 5.2. **Bài toán n-ropes**. Cho n dây với chiều dài khác nhau. Ta cần phải nối các dây lại với nhau thành một dây. Chi phí nối hai dây lại với nhau được tính bằng tổng độ dài hai dây. Nhiệm vụ của bài toán là tìm cách nối các dây lại với nhau thành một dây sao cho chi phí nối các dây lại với nhau là ít nhất.

Input:

- Số lượng dây: 4

- Độ dài dây L[]= { 4, 3, 2, 6}

Output: Chi phí nối dây nhỏ nhất.

OPT = 29

Chi phí nhỏ nhất được thực hiện như sau: lấy dây số 3 nối với dây số 2 để được tập 3 dây với độ dài 4, 5, 6. Lấy dây độ dài 4 nối với dây độ dài 5 ta nhận được tập 2 dây với độ dài 6, 9. Cuối cùng nối hai dây còn lại ta nhận được tập một dây với chi phí là 6+9 =15. Như vậy, tổng chi phí nhỏ nhất của ba lần nối dây là 5 + 9 + 15 = 29.

Ta không thể có cách nối dây khác với chi phí nhỏ hơn 29. Ví dụ lấy dây 1 nối dây 2 ta nhận được 3 dây với độ dài {7, 2, 6}. Lấy dây 3 nối dây 4 ta nhận được tập hai dây với độ dài {7, 8}, nối hai dây cuối cùng ta nhận được 1 dây với độ dài 15. Tuy vậy, tổng chi phí là 7 + 8 + 15 = 30.

Lời giải. Sử dụng thuật toán tham lam dựa vào hàng đợi ưu tiên như trong Hình 3.16.

Thuật toán. *Sử dụng phương pháp tham lam dựa vào hàng đợi ưu tiên*.
Thuật toán Greedy-N-Ropes(int L[], int n):
Input:
- n : số lượng dây.
- L[] : chi phí nối dây.

Output:
- Chi phí nối dây nhỏ nhất.

Actions:

Bước 1 . Tạo pq là hàng đợi ưu tiên lưu trữ độ dài n dây.

Bước 2 (Lặp).

```
OPT = 0; // Chi phí nhỏ nhất.
While (pq.size>1) {
        First = pq.top; pq.pop(); //Lấy và loại phần tử đầu tiên trong pq.
        Second = pq.top; pq.pop(); //Lấy và loại phần tử kế tiếp trong pq.
        OPT = First + Second; //Giá trị nhỏ nhất để nối hai dây
        Pq.push(First + Second); //Đưa lại giá trị First + Second vào pq.
}
```

Bước 3(Trả lại kết quả).

```
Return(OPT);
```

EndActions.

Hình 5.10. Thuật toán tham giải bài toán n-ropes

Kiểm nghiệm Thuật toán Greedy-N-Ropes(int L[], int n):
Input:

- Số lượng dây n = 8.
- Chi phí nối dây L[] = { 9, 7, 12, 8, 6, 5, 14, 4}.

Output:

- Chi phí nối dây nhỏ nhất.

Phương pháp tiến hành::

Bước	Giá trị First, Second	OPT=?	Trạng thái hàng đợi ưu tiên.
		0	4, 5, 6, 7, 8, 9, 12, 14
1	First=4; Second=5	9	6, 7, 8, 9, 9, 12, 14
2	First=6; Second=7	22	8, 9, 9, 12, 13, 14
3	First=8; Second=9	39	9, 12, 13, 14, 17
4	First=9; Second=12	60	13, 14, 17, 21
5	First=13; Second=14	87	17, 21, 27
6	First=17; Second=21	125	27, 38
7	First=27; Second=38	190	65
OPT = 190			

Hình 5.11. Kiểm nghiệm thuật toán tham giải bài toán n-rope

PHỤ LỤC: BÀI TẬP LUYỆN TẬP

BÀI TẬP CHƯƠNG 1. KỸ THUẬT LẬP TRÌNH VỚI NGÔN NGỮ JAVA

1.1. Bài tập lập trình Java cơ bản

BÀI 1. ƯỚC SỐ CHUNG LỚN NHẤT VÀ BỘI SỐ CHUNG NHỎ NHẤT

Viết chương trình tìm ước số chung lớn nhất và bội số chung nhỏ nhất của hai số nguyên dương a,b.

Dữ liệu vào: Dòng đầu ghi số bộ test. Mỗi bộ test ghi trên một dòng 2 số nguyên a và b không quá 9 chữ số.

Kết quả: Mỗi bộ test ghi trên 1 dòng, lần lượt là USCLN, sau đó đến BSCNN.

Ví dụ:

Input	**Output**
2 12 34 1234 5678	2 204 2 3503326

BÀI 2. BẮT ĐẦU VÀ KẾT THÚC

Viết chương trình kiểm tra một số nguyên dương bất kỳ (2 chữ số trở lên, không quá 9 chữ số) có chữ số bắt đầu và kết thúc bằng nhau hay không.

Dữ liệu vào: Dòng đầu tiên ghi số bộ test. Mỗi bộ test viết trên một dòng số nguyên dương tương ứng cần kiểm tra.

Kết quả: Mỗi bộ test viết ra YES hoặc NO, tương ứng với bộ dữ liệu vào

Ví dụ:

Input	**Output**
2	YES

12451 1000012	NO

BÀI 3. PHÉP CỘNG

Cho một phép toán có dạng **a + b = c** với a,b,c chỉ là các số nguyên dương có một chữ số. Hãy kiểm tra xem phép toán đó có đúng hay không.

Dữ liệu vào: Chỉ có một dòng ghi ra phép toán (gồm đúng 9 ký tự)

Kết quả: Ghi ra YES nếu phép toán đó đúng. Ghi ra NO nếu sai.

Ví dụ:

Test 1	**Test 2**
Input 1 + 2 = 3 Output YES	Input 2 + 2 = 5 Output NO

BÀI 4. CHIA HẾT CHO 2

Cho số nguyên dương N.

Nhiệm vụ của bạn là hãy xác định xem có bao nhiêu ước số của N chia hết cho 2?

Dữ liệu vào: Dòng đầu tiên là số lượng bộ test T ($T \leq 100$). Mỗi bộ test gồm một số nguyên N ($1 \leq N \leq 10^9$)

Kết quả: Với mỗi test, in ra đáp án tìm được trên một dòng.

Ví dụ:

Input	**Output**
2 9 8	0 3

BÀI 5. ƯỚC SỐ NGUYÊN TỐ LỚN NHẤT

Cho số nguyên dương N. Hãy đưa ra ước số nguyên tố lớn nhất của N.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào T bộ test. Mỗi bộ test ghi số nguyên dương N.
- T, N thỏa mãn ràng buộc: $1 \leq T \leq 100$; $2 \leq N \leq 10^{10}$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input:	**Output:**
2 315 31	7 31

BÀI 6. KIỂM TRA SỐ FIBONACCI

Cho số nguyên dương n. Hãy kiểm tra xem n có phải là số trong dãy Fibonacci hay không?

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là một số nguyên dương n.
- T, n thỏa mãn ràng buộc :$1 \leq T \leq 100$; $1 \leq n \leq 10^{18}$.

Output:

- Đưa ra "YES" hoặc "NO" tương ứng với n là số Fibonacci hoặc không phải số Fibonacci của mỗi test theo từng dòng.

Ví dụ:

Input	**Output**
2 8	YES NO

15	

BÀI 7. LIỆT KÊ VÀ ĐẾM

Cho một dãy các số nguyên dương không quá 9 chữ số, mỗi số cách nhau vài khoảng trống, có thể xuống dòng. Hãy tìm các số không giảm (các chữ số theo thứ tự từ trái qua phải tạo thành dãy không giảm) và đếm số lần xuất hiện của các số đó.

Dữ liệu vào: Gồm các số nguyên dương không quá 9 chữ số. Không quá 100000 số.

Kết quả Ghi ra các số không giảm kèm theo số lần xuất hiện. Các số được liệt kê theo thứ tự sắp xếp số lần xuất hiện giảm dần. Các số có số lần xuất hiện bằng nhau thì số nào xuất hiện trước in ra trước.

Ví dụ:

Input	**Output**
123 321 23456 123 123 23456 3523 123 321 4567 8988 78 7654 9899 3456 123 678 999 78 3456 987654321 4546 63543 4656 13432 4563 123471 659837 454945 34355 9087 9977 98534 3456 23134	123 5 23456 2 78 2 4567 1 3456 1 678 1 999 1

BÀI 8. SỐ TĂNG GIẢM

Một số được gọi là số tăng giảm nếu số đó có các chữ số thỏa mãn hoặc tăng dần, hoặc giảm dần từ trái qua phải. Hãy đếm các số nguyên tố là số tăng giảm với **số chữ số cho trước**.

Dữ liệu vào: Dòng đầu tiên ghi số bộ test. Mỗi bộ test viết trên một dòng số chữ số tương ứng cần kiểm tra (lớn hơn 1 và nhỏ hơn 10)

Kết quả: Ghi ra số lượng các số thỏa mãn điều kiện.

Input	**Output**
2	20

2 4	50

BÀI 9. PHÂN TÍCH THỪA SỐ NGUYÊN TỐ

Hãy phân tích một số nguyên dương thành tích các thừa số nguyên tố.

Dữ liệu vào: Dòng đầu tiên ghi số bộ test. Mỗi bộ test viết trên một dòng số nguyên dương n không quá 9 chữ số.

Kết quả: Mỗi bộ test viết ra thứ tự bộ test, sau đó lần lượt là các số nguyên tố khác nhau có trong tích, với mỗi số viết thêm số lượng số đó. Xem ví dụ để hiểu rõ hơn về cách viết kết quả.

Ví dụ

Input	**Output**
3 60 128 10000	Test 1: 2(2) 3(1) 5(1) Test 2: 2(7) Test 3: 2(4) 5(4)

BÀI 10. SỐ ĐẸP

Một số được coi là đẹp nếu nếu nó có tính chất thuận nghịch và tổng chữ số chia hết cho 10. Bài toán đặt ra là cho trước số chữ số. Hãy đếm xem có bao nhiêu số đẹp với số chữ số như vậy.

Dữ liệu vào: Dòng đầu tiên ghi số bộ test. Mỗi bộ test viết trên một dòng số chữ số tương ứng cần kiểm tra (lớn hơn 1 và nhỏ hơn 10).

Kết quả: Mỗi bộ test viết ra số lượng số đẹp tương ứng.

Input	**Output**
2 2 5	1 90

BÀI 11. SỐ THUẦN NGUYÊN TỐ

Một số được coi là thuần nguyên tố nếu nó là số nguyên tố, tất cả các chữ số là nguyên tố và tổng chữ số của nó cũng là một số nguyên tố. Bài toán đặt ra là đếm xem trong một đoạn giữa hai số nguyên cho trước có bao nhiêu số thuần nguyên tố.

Dữ liệu vào: Dòng đầu tiên ghi số bộ test. Mỗi bộ test viết trên một dòng hai số nguyên dương tương ứng, cách nhau một khoảng trống. Các số đều không vượt quá 9 chữ số.

Kết quả: Mỗi bộ test viết ra số lượng các số thuần nguyên tố tương ứng.

Ví dụ

Input	**Ouput**
2 23 199 2345 6789	1 15

BÀI 12. GHÉP HÌNH

Cho ba hình chữ nhật. Các bạn được phép xoay hình nhưng không được phép xếp chồng lấn lên nhau, hỏi 3 hình chữ nhật đó có thể ghép thành một hình vuông được hay không

Dữ liệu vào: Có ba dòng, mỗi dòng ghi hai số nguyên dương là chiều rộng và chiều cao của hình chữ nhật (các số đều không quá 100).

Kết quả: Ghi ra YES nếu có thể tạo thành hình vuông, NO nếu không thể.

Ví dụ:

Input	**Output**
8 2 1 6 7 6	YES

1.2. Bài tập về Mảng và Xâu ký tự

BÀI 1. MẢNG ĐỐI XỨNG

Nhập một dãy số nguyên có n phần tử (n không quá 100, các phần tử trong dãy không quá 10^9). Hãy viết chương trình kiểm tra xem dãy có phải đối xứng hay không. Nếu đúng in ra YES, nếu sai in ra NO.

Dữ liệu vào: Dòng đầu ghi số bộ test, mỗi bộ test gồm hai dòng. Dòng đầu là số phần tử của dãy, dòng sau ghi ra dãy đó, mỗi số cách nhau một khoảng trống.

Kết quả: Ghi ra YES hoặc NO trên một dòng.

Ví dụ

Input	**Ouput**
2 4 1 4 4 1 5 1 5 5 5 3	YES NO

BÀI 2. TÍCH MA TRẬN VỚI CHUYỂN VỊ CỦA NÓ

Cho ma trận A chỉ gồm các số nguyên dương cấp N*M. Hãy viết chương trình tính tích của A với ma trận chuyển vị của A.

Dữ liệu vào: Dòng đầu tiên ghi số bộ test. Với mỗi bộ test: Dòng đầu tiên ghi hai số n và m là bậc của ma trân a; n dòng tiếp theo, mỗi dòng ghi m số của một dòng trong ma trận A.

Kết quả: Với mỗi bộ test ghi ra thứ tự bộ test, sau đó đến ma trận tích tương ứng, mỗi số cách nhau đúng một khoảng trống.

Ví dụ

Input	**Output**
1 2 2 1 2 3 4	Test 1: 5 11 11 25

BÀI 3. SỐ TĂNG GIẢM

Một số được gọi là số tăng giảm nếu số đó có các chữ số thỏa mãn hoặc không giảm, hoặc không tăng từ trái qua phải. Hãy kiểm tra xem một số có phải số tăng giảm hay không.

Dữ liệu vào: Dòng đầu tiên ghi số bộ test. Mỗi bộ test viết trên một dòng một số nguyên dương cần kiểm tra, không quá 500 chữ số.

Kết quả: Mỗi bộ test viết ra chữ YES nếu đó đúng là số tăng giảm, chữ NO nếu ngược lại.

Input	**Output**
3 2345566777777777778888888888899999999 9877777777777777777777655442222222111111111000 434353124325436576576587698980978764654656879 87	YES YES NO

BÀI 4. CHÈN MẢNG

Nhập 2 mảng (a, N) và (b, M) và số nguyên p ($0 \le p < M \le N < 100$). Hãy chèn mảng b vào vị trí p của mảng a.

Input: Dòng đầu ghi số bộ test, mỗi bộ test gồm 3 dòng: dòng đầu ghi 3 số N,M,p. Dòng thứ 2 ghi N số của mảng a. Dòng thứ 3 ghi M số của mảng b.

Output ghi ra thứ tự bộ test và dãy số sau khi chèn.

Ví dụ:

Input	**Output**
1 4 3 1 5 3 6 7 2 9 11	Test 1: 5 2 9 11 3 6 7

BÀI 5. SỐ LA MÃ

Bảng chữ số La Mã bao gồm các chữ cái với ý nghĩa I=1; V=5; X=10; L=50; C=100;D=500; M=1000. Một số quy tắc viết các số La Mã như sau:

- Tính từ trái sang phải giá trị của các chữ số và nhóm chữ số giảm dần.

- I chỉ có thể đứng trước V hoặc X, X chỉ có thể đứng trước L hoặc C, C chỉ có thể đứng trước D hoặc M.
- Các chữ cái I, X, C, M, không được lặp lại quá ba lần liên tiếp; các chữ cái V, L, D không được lặp lại quá một lần liên tiếp.

Bài toán đặt ra là cho một xâu ký tự mô tả **đúng** một số La Mã. Hãy tính giá trị thập phân của số đó

Input: Dòng đầu ghi số bộ test. Mỗi bộ test ghi trên một dòng dãy ký tự số La Mã. Độ dài không quá 10 ký tự.

Output: Với mỗi bộ test ghi ra kết quả tương ứng

Ví dụ:

Input	**Output**
3 XIX DC CD	19 600 400

BÀI 6. VÒNG TRÒN

Tí viết bảng chữ cái 2 lần lên trên một vòng tròn, mỗi kí tự xuất hiện đúng 2 lần. Sau đó nối lần lượt các kí tự giống nhau lại. Tổng cộng có 26 đoạn thẳng.

Hình vẽ quá chằng chịt, Tí muốn đố các bạn xem có tất cả bao nhiêu giao điểm?

Một giao điểm được tính khi hai đường thẳng của một cặp kí tự cắt nhau.

Input

Gồm một xâu có đúng 52 kí tự in hoa. Mỗi kí tự xuất hiện đúng 2 lần.

Output

In ra đáp án tìm được.

Ví dụ:

Input	Output
ABCCABDDEEFFGGHHIIJJKKLLMMNNOOPPQQRRSSTTUUVVWWXXYYZZ	1

Giải thích test: Chỉ có duy nhất cặp kí tự 'A', 'B' thỏa mãn.

BÀI 7. TÍNH TỔNG CÁC CHỮ SỐ

Cho xâu ký tự S bao gồm các ký tự 'A',..,'Z' và các chữ số '0',...,'9'. Nhiệm vụ của bạn in các ký tự từ 'A',.., 'Z' trong S theo thứ tự từ điển và nối với tổng các chữ số trong S ở cuối cùng. Ví dụ S ="ACCBA10D2EW30" ta nhận được kết quả: "AABCCDEW6".

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào T bộ test. Mỗi bộ test là một xâu ký tự S.
- T, S thỏa mãn ràng buộc: $1 \leq T \leq 100$; $1 \leq Length(S) \leq 10^5$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input:	Output:
2 AC2BEW3 ACCBA10D2EW30	ABCEW5 AABCCDEW6

BÀI 8. SỐ ĐẸP

Một số được coi là đẹp nếu đó là số thuận nghịch và chỉ toàn các chữ số chẵn. Viết chương trình đọc vào các số nguyên dương có không quá 500 chữ số và kiểm tra xem số đó có đẹp hay không.

Dữ liệu vào:

Dòng đầu tiên ghi số bộ test.

Mỗi bộ test viết trên một dòng số nguyên dương n không quá 500 chữ số.

Kết quả:

Mỗi bộ test viết ra trên một dòng chữ YES nếu đó là số đẹp, chữ NO nếu ngược lại

Ví dụ

Input	**Output**
4	NO

123456787654321	YES
86442824468	YES
8006000444422220000222244440006008	NO
2353657897876543245678565786543567865 56	

BÀI 9. CHUẨN HÓA XÂU HỌ TÊN

Một xâu họ tên được coi là viết chuẩn nếu chữ cái đầu tiên mỗi từ được viết hoa, các chữ cái khác viết thường. Các từ cách nhau đúng một dấu cách và không có khoảng trống thừa ở đầu và cuối xâu. Hãy viết chương trình đưa các xâu họ tên về dạng chuẩn.

Dữ liệu vào :

Dòng 1 ghi số bộ test.

Mỗi bộ test ghi trên một dòng xâu ký tự họ tên, không quá 100 ký tự.

Kết quả :

Với mỗi bộ test ghi ra xâu ký tự họ tên đã chuẩn hóa.

Ví dụ:

Input	**Output**
3 nGuYEN vAN naM tRan TRUNG hiEU vO le hOA bINh	Nguyen Van Nam Tran Trung Hieu Vo Le Hoa Binh

BÀI 10 ĐỊA CHỈ EMAIL

Địa chỉ email của các cán bộ, giảng viên PTIT được tạo ra bằng cách viết đầy đủ tên và ghép với các chữ cái đầu của họ và tên đệm. Nếu có nhiều người cùng email thì từ người thứ 2 sẽ thêm số thứ tự vào email đó.

Cho trước các xâu họ tên (có thể không chuẩn). Hãy tạo ra các địa email tương ứng.

Dữ liệu vào:

- Dòng 1 ghi số N là xâu họ tên trong danh sách
- N dòng tiếp theo ghi lần lượt các xâu họ tên (không quá 50 ký tự)

Kết quả: Ghi ra các email được tạo ra.

Ví dụ:

Input	Output
4 nGUYEn quaNG vInH tRan thi THU huOnG nGO quoC VINH lE tuAn aNH	vinhnq@ptit.edu.vn huongttt@ptit.edu.vn vinhnq2@ptit.edu.vn anhlt@ptit.edu.vn

BÀI 11. RÚT GỌN XÂU KÝ TỰ

Cho một xâu S. Mỗi bước, bạn được phép xóa đi 2 kí tự liền nhau mà giống nhau. Chẳng hạn xâu "aabcc" có thể trở thành "bcc" hoặc "aab" sau 1 lần xóa.

Hỏi xâu cuối cùng thu được là gì? Nếu xâu rỗng, in ra "Empty String".

Input:

Một xâu S chỉ gồm các chữ cái thường, có độ dài không vượt quá 100.

Output:

In ra đáp án tìm được.

Ví dụ:

Test 1	Test 2
Input: aaabccddd Output: abd	Input: abba Output: Empty String

1.3 Bài tập cơ bản áp dụng Java Collection

BÀI 1. ĐẾM CÁC SỐ NGUYÊN TỐ TRONG DÃY

Cho dãy số A có n phần tử chỉ bao gồm các số nguyên dương (không quá 10^5). Hãy xác định các số nguyên tố trong dãy và đếm xem mỗi số xuất hiện bao nhiêu lần.

Dữ liệu vào: Dòng đầu tiên ghi số bộ test. Với mỗi bộ test: dòng đầu ghi số n (không quá 10); dòng tiếp theo ghi n số của dãy.

Kết quả: Với mỗi bộ test ghi ra thứ tự bộ test, sau đó lần lượt là các số nguyên tố trong dãy **theo thứ tự từ nhỏ đến lớn** và số lần xuất hiện của nó.

Ví dụ:

Input	**Output**
1 10 1 7 2 8 3 3 2 1 3 2	Test 1: 2 xuat hien 3 lan 3 xuat hien 3 lan 7 xuat hien 1 lan

BÀI 2. TRỘN HAI DÃY VÀ SẮP XẾP

Cho hai dãy số nguyên dương A và B không quá 100 phần tử, các giá trị trong dãy không quá 30000 và số phần tử của hai dãy bằng nhau. Hãy trộn hai dãy với nhau sao cho dãy A được đưa vào các vị trí có chỉ số chẵn, dãy B được đưa vào các vị trí có chỉ số lẻ. Đồng thời, dãy A được sắp xếp tăng dần, còn dãy B được sắp xếp giảm dần. *(Chú ý: chỉ số tính từ 0)*

Dữ liệu vào: Dòng 1 ghi số bộ test. Với mỗi bộ test: dòng đầu tiên ghi số n. Dòng tiếp theo ghi n số nguyên dương của dãy A. Dòng tiếp theo ghi n số nguyên dương của dãy B

Kết quả: Với mỗi bộ test, đưa ra thứ tự bộ test và dãy kết quả.

Ví dụ:

Input	**Output**
2 5 1 2 3 1 2 3 1 2 3 1 4 4 2 7 1 5 6 2 8	Test 1: 1 3 1 3 2 2 2 1 3 1 Test 2: 1 8 2 6 4 5 7 2

BÀI 3. ĐẾM SỐ LẦN XUẤT HIỆN

Cho dãy số A có n phần tử chỉ bao gồm các số nguyên dương (không quá 10^5). Hãy đếm xem mỗi số xuất hiện bao nhiêu lần.

Dữ liệu vào: Dòng đầu tiên ghi số bộ test. Với mỗi bộ test: dòng đầu ghi số n (không quá 10); dòng tiếp theo ghi n số của dãy.

Kết quả: Với mỗi bộ test ghi ra thứ tự bộ test, sau đó lần lượt là các số nguyên tố trong dãy **theo thứ tự xuất hiện trong dãy** và số lần xuất hiện của nó.

Input	**Output**
1 10 1 7 2 8 3 3 2 1 3 2	Test 1: 1 xuat hien 2 lan 7 xuat hien 1 lan 2 xuat hien 3 lan 8 xuất hiện 1 lần 3 xuất hiện 3 lần

BÀI 4. SẮP XẾP THEO SỐ LẦN XUẤT HIỆN

Cho dãy số A[] gồm có N phần tử. Nhiệm vụ của bạn là hãy sắp xếp dãy số này theo tần suất xuất hiện của chúng. Số nào có số lần xuất hiện lớn hơn in ra trước. Nếu có 2 số có số lần xuất hiện bằng nhau, số nào xuất hiện trong dãy A[] trước sẽ được in ra trước.

Input: Dòng đầu tiên là số lượng bộ test T ($T \leq 10$). Mỗi test gồm số nguyên N ($1 \leq N \leq 100\,000$), số lượng phần tử trong dãy số ban đầu. Dòng tiếp theo gồm N số nguyên A[i] ($-10^9 \leq A[i] \leq 10^9$).

Output: Với mỗi test, in ra trên một dòng là dãy số thu được sau khi thực hiện sắp xếp.

Ví dụ:

Input	**Output**
2 8 2 5 2 8 5 6 8 8	8 8 8 2 2 5 5 6 8 8 8 2 2 5 5 6 -1 9999999

10 2 5 2 6 -1 9999999 5 8 8 8	

BÀI 5. SỐ ĐẦU TIÊN BỊ LẶP

Cho dãy số A[] gồm có N phần tử. Nhiệm vụ của bạn là hãy tìm số xuất hiện nhiều hơn 1 lần trong dãy số và số thứ tự là nhỏ nhất.

Input: Dòng đầu tiên là số lượng bộ test T ($T \leq 10$). Mỗi test gồm số nguyên N ($1 \leq N \leq 100\,000$), số lượng phần tử trong dãy số ban đầu. Dòng tiếp theo gồm N số nguyên A[i] ($0 \leq A[i] \leq 10^9$).

Output: Với mỗi test in ra đáp án của bài toán trên một dòng. Nếu không tìm được đáp án, in ra "NO".

Ví dụ:

Input	**Output**
2 7 10 5 3 4 3 5 6 4 1 2 3 4	5 NO

Giải thích test 1: Cả 5 và 3 đều xuất hiện 2 lần, nhưng số 5 có số thứ tự nhỏ hơn.

BÀI TẬP CHƯƠNG 2. LÝ THUYẾT TỔ HỢP

2.1. Bài tập về Bài toán đếm

BÀI 1. TỔ HỢP C(n, k)

Cho 2 số nguyên n, k. Bạn hãy tính C(n, k) modulo 10^9+7.

Input:

- Dòng đầu tiên là số lượng bộ test T ($T \leq 20$).
- Mỗi test gồm 2 số nguyên n, k ($1 \leq k \leq n \leq 1000$).

Output:

- Với mỗi test, in ra đáp án trên một dòng.

Ví dụ:

Input	Output
2 5 2 10 3	10 120

BÀI 2. BẬC THANG

Một chiếc cầu thang có N bậc. Mỗi bước, bạn được phép bước lên trên tối đa K bước. Hỏi có tất cả bao nhiêu cách bước để đi hết cầu thang? (Tổng số bước đúng bằng N).

Input:

- Dòng đầu tiên là số lượng bộ test T ($T \leq 100$).
- Mỗi test gồm hai số nguyên dương N và K($1 \leq N \leq 100000$, $1 \leq K \leq 100$).

Output:

- Với mỗi test, in ra đáp án tìm được trên một dòng theo modulo 10^9+7.

Ví dụ:

Input	Output
2	2

2 2 4 2	5

Giải thích test 1: Có 2 cách đó là (1, 1) và (2).

Giải thích test 2: 5 cách đó là: (1, 1, 1, 1), (1, 1, 2), (1, 2, 1), (2, 1, 1), (2, 2).

BÀI 3. CATALAN NUMBER

Catalan Number là dãy số thỏa mãn biểu thức:

$$C_n = \begin{cases} 0 & nếu\ \ n = 0 \\ \sum_{i=0}^{n-1} C_i C_{n-i-1} & nếu\ n > 0 \end{cases}$$

Dưới đây là một số số Catalan với n=0, 1,2,.. : 1, 1, 2, 5, 14, 42, 132, 429,… Cho số tự nhiên N. Nhiệm vụ của bạn là đưa ra số Catalan thứ N.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là một số nguyên n.
- T, n thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le n \le 100$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input		**Output**
3 5 4 10		42 14 16796

BÀI 4. TÍNH P(N,K)

P(n, k) là số phép biểu diễn các tập con có thứ tự gồm k phần tử của tập gồm n phần tử. Số P(n, k) được định nghĩa theo công thức sau:

$$P(n,k) = \begin{cases} 0 & nếu\ k > n \\ \dfrac{n!}{(n-k)!} = n.(n-1)\dots(n-k+1) & nếu\ k < n \end{cases}$$

Cho số hai số n, k. Hãy tìm P(n,k) theo modulo 10^9+7.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là một cặp số n, k được viết trên một dòng.
- T, n, k thỏa mãn ràng buộc: 1 ≤ T ≤ 100; 1 ≤ n,k ≤ 1000.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	**Output**
2 5 2 4 2	20 12

BÀI 5. TỔNG CÁC XÂU CON

Cho số nguyên dương N được biểu diễn như một xâu ký tự số. Nhiệm vụ của bạn là tìm tổng của tất cả các số tạo bởi các xâu con của N. Ví dụ N="1234" ta có kết quả là 1670 = 1 + 2 + 3 + 4 + 12 + 23 + 34 + 123 + 234 + 1234.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là một số N.
- T, N thỏa mãn ràng buộc: 1≤T≤100; 1≤N ≤10^{12}.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	**Output**
2 1234 421	1670 491

BÀI 6. TỔNG BẰNG K

Cho một mảng A[] gồm N số nguyên và số K. Tính số cách lấy tổng các phần tử của A[] để bằng K. Phép lấy lặp các phần tử hoặc sắp đặt lại các phần tử được chấp thuận. Ví dụ với mảng A[] = {1, 5, 6}, K = 7 ta có 6 cách sau:

7 = 1 + 1 + 1+1 + 1 + 1+1 (lặp số 1 7 lần)

7 = 1 + 1 + 5 (lặp số 1)

7 = 1 + 5 + 1 (lặp và sắp đặt lại số 1)

7 = 1 + 6

7 = 6 + 1

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai phần: phần thứ nhất đưa vào N và K; dòng tiếp theo đưa vào N số của mảng A[]; các số được viết cách nhau một vài khoảng trống.
- T, N, K, A[i] thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 1000$; $1 \le A[i] \le 100$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng. Khi kết quả quá lớn đưa ra kết quả dưới dạng modulo với 10^9+7.

Ví dụ:

Input	**Output**
2 3 7 1 5 6 4 14 12 3 1 9	6 150

BÀI 7. CON ẾCH

Một con ếch có thể nhảy 1, 2, 3 bước để có thể lên đến một đỉnh cần đến. Hãy đếm số các cách con ếch có thể nhảy đến đỉnh.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là số n là số bước con ếch có thể lên được đỉnh.
- T, n thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le n \le 50$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	Output
2 1 5	1 13

BÀI 8. GIẢI MÃ

Một bản tin M đã mã hóa bí mật thành các con số theo ánh xạ như sau: 'A'->1, 'B'->2, .., 'Z'->26. Hãy cho biết có bao nhiêu cách khác nhau để giải mã bản tin M. Ví dụ với bản mã M="123" nó có thể được giải mã thành ABC (1 2 3), LC (12 3), AW(1 23).

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là một xâu ký tự số M.
- T, M thỏa mãn ràng buộc: $1 \leq T \leq 100$; $1 \leq length(M) \leq 40$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	**Output**
2 123 2563	3 2

BÀI 9. TỔNG BÌNH PHƯƠNG

Mọi số nguyên dương N đều có thể phân tích thành tổng các bình phương của các số nhỏ hơn N. Ví dụ số $100 = 10^2$ hoặc $100 = 5^2 + 5^2 + 5^2 + 5^2$. Cho số nguyên dương N. Nhiệm vụ của bạn là tìm số lượng ít nhất các số nhỏ hơn N mà có tổng bình phương bằng N.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi test là một số tự nhiên N được viết trên 1 dòng.

- T, N thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 10000$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	**Output**
3 100 6 25	1 3 1

2.2. Bài tập về Bài toán liệt kê

BÀI 1. XÂU NHỊ PHÂN KẾ TIẾP

Cho xâu nhị phân X[], nhiệm vụ của bạn là hãy đưa ra xâu nhị phân tiếp theo của X[]. Ví dụ X[] =”010101” thì xâu nhị phân tiếp theo của X[] là “010110”.

Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là một xâu nhi phân X.
- T, X[] thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le \text{length}(X) \le 10^3$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Input	**Output**
2 010101 111111	010110 000000

BÀI 2. TẬP CON KẾ TIẾP

Cho hai số N, K và một tập con K phần tử X[] =(X_1, X_2,.., X_K) của 1, 2, .., N. Nhiệm vụ của bạn là hãy đưa ra tập con K phần tử tiếp theo của X[]. Ví dụ N=5, K=3, X[] ={2, 3, 4} thì tập con tiếp theo của X[] là {2, 3, 5}.

Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai dòng: dòng thứ nhất là hai số N và K; dòng tiếp theo đưa vào K phần tử của X[] là một tập con K phần tử của 1, 2, .., N.
- T, K, N, X[] thỏa mãn ràng buộc: $1 \leq T \leq 100$; $1 \leq K \leq N \leq 10^3$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Input	Output
2 5 3 1 4 5 5 3 3 4 5	2 3 4 1 2 3

BÀI 3. HOÁN VỊ KẾ TIẾP

Cho số tự nhiên N và một hoán vị X[] của 1, 2, .., N. Nhiệm vụ của bạn là đưa ra hoán vị tiếp theo của X[]. Ví dụ N=5, X[] = {1, 2, 3, 4, 5} thì hoán vị tiếp theo của X[] là {1, 2, 3, 5, 4}.

Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai dòng: dòng thứ nhất là số N; dòng tiếp theo đưa vào hoán vị X[] của 1, 2, .., N.
- T, N, X[] thỏa mãn ràng buộc: $1 \leq T \leq 100$; $1 \leq N \leq 10^3$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Input	Output
2 5 1 2 3 4 5 5 5 4 3 2 1	1 2 3 5 4 1 2 3 4 5

BÀI 4. SINH TỔ HỢP

Cho hai số nguyên dương N và K. Nhiệm vụ của bạn là hãy liệt kê tất cả các tập con K phần tử của 1, 2, .., N. Ví dụ với N=5, K=3 ta có 10 tập con của 1, 2, 3, 4, 5 như sau: {1, 2, 3}, {1, 2, 4},{1, 2, 5},{1, 3, 4},{1, 3, 5},{1, 4, 5},{2, 3, 4},{2, 3, 5},{2, 4, 5},{3, 4, 5}.

Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là một cặp số tự nhiên N, K được viết trên một dòng.
- T, n thỏa mãn ràng buộc: $1 \leq T \leq 100$; $1 \leq k \leq n \leq 15$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Input	Output
2 4 3 5 3	123 124 134 234 123 124 125 134 135 145 234 235 245 345

BÀI 5. SINH HOÁN VỊ

Cho số nguyên dương N. Nhiệm vụ của bạn là hãy liệt kê tất cả các hoán vị của 1, 2, .., N. Ví dụ với N = 3 ta có kết quả: 123, 132, 213, 231, 312, 321.

Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là một số tự nhiên N được viết trên một dòng.
- T, n thỏa mãn ràng buộc: $1 \leq T, N \leq 10$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Input	Output
2 2 3	12 21 123 132 213 231 312 321

BÀI 6. PHÂN TÍCH SỐ

Cho số nguyên dương N. Nhiệm vụ của bạn là hãy liệt kê tất cả các cách phân tích số tự nhiên N thành tổng các số tự nhiên nhỏ hơn hoặc bằng N. Phép hoán vị vủa một cách được xem là giống nhau. Ví dụ với N = 5 ta có kết quả là: (5), (4, 1), (3, 2), (3, 1, 1), (2, 2, 1), (2, 1, 1, 1), (1, 1, 1, 1, 1) .

Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là một số tự nhiên N được viết trên một dòng.
- T, n thỏa mãn ràng buộc: 1≤T, N≤10.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Input	Output
2 4 5	(4) (3 1) (2 2) (2 1 1) (1 1 1 1) (5) (4 1) (3 2) (3 1 1) (2 2 1) (2 1 1 1) (1 1 1 1 1)

BÀI 7. HOÁN VỊ NGƯỢC

Cho số nguyên dương N. Nhiệm vụ của bạn là hãy liệt kê tất cả các hoán vị của 1, 2, .., N theo thứ tự ngược. Ví dụ với N = 3 ta có kết quả: 321, 312, 231, 213, 132, 123.

Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là một số tự nhiên N được viết trên một dòng.
- T, n thỏa mãn ràng buộc: 1≤T, N≤10.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Input	Output
2 2 3	21 12 321 312 231 213 132 123

2.3. Bài tập về Bài toán tối ưu

BÀI 1. TÌM MAX

Cho mảng A[] gồm N phần tử.Nhiệm vụ của bạn là tìm $max = \sum_{i=0}^{n-1} A_i * i$ bằng cách sắp đặt lại các phần tử trong mảng. Chú ý, kết quả của bài toán có thể rất lớn vì vậy bạn hãy đưa ra kết quả lấy modulo với 10^9+7.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng thứ nhất đưa vào số phần tử của mảng N; dòng tiếp theo đưa vào N số A[i] tương ứng với các phần tử của mảng A[]; các số được viết cách nhau một vài khoảng trống.
- T, N, A[i] thỏa mãn ràng buộc: 1≤T≤100; 1≤N, A[i] ≤10^7.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	Output
2 5 5 3 2 4 1 3 1 2 3	40 8

BÀI 2. TỔNG NHỎ NHẤT

Cho mảng A[] gồm các số từ 0 đến 9. Nhiệm vụ của bạn là tìm tổng nhỏ nhất của hai số được tạo bởi các số trong mảng A[]. Chú ý, tất cả các số trong mảng A[] đều được sử dụng để tạo nên hai số.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng thứ nhất đưa vào số phần tử của mảng N; dòng tiếp theo đưa vào N số A[i] tương ứng với các phần tử của mảng A[]; các số được viết cách nhau một vài khoảng trống.
- T, N, A[i] thỏa mãn ràng buộc: 1≤T≤100; 1≤N ≤20; 0≤A[i]≤9.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	Output

2 6 6 8 4 5 2 3 5 5 3 0 7 4	604 82

BÀI 3. CHIA MẢNG

Cho mảng A[] gồm N số nguyên không âm và số K. Nhiệm vụ của bạn là hãy chia mảng A[] thành hai mảng con có kích cỡ K và N-K sao cho hiệu giữa tổng hai mảng con là lớn nhất. Ví dụ với mảng A[] = {8, 4, 5, 2, 10}, K=2 ta có kết quả là 17 vì mảng A[] được chia thành hai mảng {4, 2} và { 8, 5,10} có hiệu của hai mảng con là 23-6=17 là lớn nhất.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng thứ nhất đưa vào số phần tử của mảng N và số K; dòng tiếp theo đưa vào N số A[i] tương ứng với các phần tử của mảng A[]; các số được viết cách nhau một vài khoảng trống.
- T, N, K, A[i] thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le K < N \le 50$; $0 \le A[i] \le 1000$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	Output
2 5 2 8 4 5 2 10 8 3 1 1 1 1 1 1 1 1	17 2

BÀI 4. SẮP XẾP THAM LAM

Cho mảng A[] gồm N số và thực hiện các thao tác theo nguyên tắc dưới đây:

- Ta chọn một mảng con sao cho phần tử ở giữa của mảng con cũng là phần tử ở giữa của mảng A[] (trong trường hợp N lẻ).
- Đảo ngược mảng con đã chọn trong mảng A[]. Ta được phép chọn mảng con và phép đảo ngược mảng con bao nhiêu lần tùy ý.

Ví dụ với mảng A[] = {1, 6, **3, 4, 5**, 2, 7} ta có câu trả lời là Yes vì: ta chọn mảng con {3, 4, 5} và đảo ngược để nhận được mảng A[]={1, 6, **5, 4, 3**, 2, 7}, chọn tiếp mảng con {6, 5, 4, 3, 2} và đảo ngược ta nhận được mảng A[]={1, 2, 3, 4, 5, 6, 7}. Hãy cho biết ta có thể sắp xếp được mảng A[] bằng cách thực hiện các thao tác kể trên hay không?

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng thứ nhất đưa vào số phần tử của mảng N; dòng tiếp theo đưa vào N số A[i] tương ứng với các phần tử của mảng A[]; các số được viết cách nhau một vài khoảng trống.
- T, N, A[i] thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 50$; $0 \le A[i] \le 1000$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	Output
2 7 1 6 3 4 5 2 7 7 1 6 3 4 5 7 2	Yes No

BÀI 5. GIÁ TRỊ NHỎ NHẤT CỦA BIỂU THỨC

Cho mảng A[], B[] đều có N phần tử. Nhiệm vụ của bạn là tìm giá trị nhỏ nhất của biểu thức P = A[0]*B[0] + A[1]*B[1] + ..+A[N-1]*B[N-1] bằng cách tráo đổi vị trí các phần tử của cả mảng A[] và B[].

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm 3 dòng: dòng thứ nhất đưa vào số phần tử của mảng N; dòng tiếp theo đưa vào N số A[i]; dòng cuối cùng đưa vào N số B[i] các số được viết cách nhau một vài khoảng trống.
- T, N, A[i], B[i] thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 10^7$; $0 \le A[i], B[i] \le 10^{18}$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	Output
2 7 1 6 3 4 5 2 7 1 1 1 2 3 4 3 7 1 6 3 5 5 2 2 0 1 9 0 1 2 3	45 27

BÀI 6. SỐ KHỐI LẬP PHƯƠNG

Một số X được gọi là số khối lập phương nếu X là lũy thừa bậc 3 của số Y ($X= Y^3$). Cho số nguyên dương N, nhiệm vụ của bạn là tìm số khối lập phương lớn nhất bằng cách loại bỏ đi các chữ số của N. Ví dụ số 4125 ta có kết quả là $125 = 5^3$.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là một số tự nhiên N được viết trên một dòng.
- T, N thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 10^{18}$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng. Nếu không tìm được đáp án in ra -1.

Ví dụ:

Input	Output
2 4125 976	125 -1

BÀI 7. SỐ NHỎ NHẤT

Cho hai số nguyên dương S và D, trong đó S là tổng các chữ số và D là số các chữ số của một số. Nhiệm vụ của bạn là tìm số nhỏ nhất thỏa mãn S và D? Ví dụ với S = 9, D = 2 ta có số nhỏ nhất thỏa mãn S và D là 18.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là bộ 2 số S và D được viết trên một dòng.
- T, S, D thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le S, D \le 1000$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng. Nếu không có đáp án, in ra -1.

Ví dụ:

Input	Output
2 9 2 20 3	18 299

BÀI 8. GIÁ TRỊ NHỎ NHẤT CỦA XÂU

Cho xâu ký tự S. Ta gọi giá trị của xâu S là tổng bình phương số lần xuất hiện mỗi ký tự trong S. Hãy tìm giá trị nhỏ nhất của xâu S sau khi thực hiện K lần loại bỏ ký tự.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai phần: phần thứ nhất là số K; phần thứ hai là một xâu ký tự S được viết trên một dòng.
- T, S, K thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le length(S) \le 10000$; $1 \le K \le 1000$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	Output
2 2 ABCCBC 2	6 2

AAAB	

BÀI 9. SỐ MAY MẮN

Hoàng yêu thích các số may mắn. Ta biết rằng một số là *số may mắn* nếu biểu diễn thập phân của nó chỉ chứa các chữ số may mắn là 4 và 7. Ví dụ, các số 47, 744, 4 là số may mắn và 5, 17, 467 không phải. Hoàng muốn tìm số may mắn bé nhất có tổng các chữ số bằng n. Hãy giúp anh ấy

Dữ liệu vào: Dòng đầu ghi số bộ test, mỗi bộ test có một dòng chứa số nguyên n ($1 \le n \le 10^6$) — tổng các chữ số của số may mắn cần tìm.

Kết quả: In ra trên 1 dòng số may mắn bé nhất, mà tổng các chữ số bằng n. Nếu không tồn tại số thỏa mãn, in ra -1.

Ví dụ:

Input	**Output**
2 11 10	47 -1

BÀI 10. PHÂN SỐ ĐƠN VỊ

Một phân số đơn vị nếu tử số của phân số đó là 1. Mọi phân số nguyên dương đều có thể biểu diễn thành tổng các phân số đơn vị. Ví dụ 2/3 = 1/2 + 1/6. Cho phân số nguyên dương P/Q bất kỳ (P < Q), hãy biểu diễn phân số nguyên dương thành tổng phân số đơn vị.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là bộ đôi tử số P và mẫu số Q của phân số nguyên dương được viết trên một dòng.
- T, P, Q thỏa mãn ràng buộc: 1≤T≤100; 1≤P, Q≤100.

Output:

- Đưa ra đáp án tìm được trên 1 dòng, theo dạng "1/a + 1/b + …"

Ví dụ:

Input	Output
2	1/2 + 1/6

2 3 1 3	1/3

BÀI TẬP CHƯƠNG 3. CÁC MÔ HÌNH THUẬT TOÁN CƠ BẢN

3.1. Bài tập về Thuật toán Tham lam

BÀI 1. SẮP XẾP CÔNG VIỆC 1

Cho hệ gồm N hành động. Mỗi hành động được biểu diễn như một bộ đôi <S_i, F_i> tương ứng với thời gian bắt đầu và thời gian kết thúc của mỗi hành động. Hãy tìm phương án thực hiện nhiều nhất các hành động được thực hiện bởi một máy hoặc một người sao cho hệ không xảy ra mâu thuẫn.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm 3 dòng: dòng thứ nhất đưa vào số lượng hành động N; dòng tiếp theo đưa vào N số S_i tương ứng với thời gian bắt đầu mỗi hành động; dòng cuối cùng đưa vào N số F_i tương ứng với thời gian kết thúc mỗi hành động; các số được viết cách nhau một vài khoảng trống.
- T, N, S_i, F_i thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N, F_i, S_i \le 1000$.

Output:

- Đưa số lượng lớn nhất các hành động có thể được thực thi bởi một máy hoặc một người.

Ví dụ:

Input	Output
1 6 1 3 0 5 8 5 2 4 6 7 9 9	4

BÀI 2. SẮP XẾP CÔNG VIỆC 2

Cho N công việc. Mỗi công việc được biểu diễn như một bộ 3 số nguyên dương <JobId, Deadline, Profit>, trong đó JobId là mã của việc, Deadline là thời gian kết thúc của việc, Profit là lợi nhuận đem lại nếu hoàn thành việc đó đúng thời gian. Thời gian để hoàn toàn mỗi công việc là **1 đơn vị thời gian**. Hãy cho biết lợi nhuận lớn nhất có thể thực hiện các việc với giả thiết mỗi việc được thực hiện đơn lẻ.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai phần: phần thứ nhất là số lượng Job N; phần thứ hai đưa vào 3×N số tương ứng với N job.
- T, N, JobId, Deadline, Profit thỏa mãn ràng buộc:$1 \le T \le 100$; $1 \le N \le 1000$; $1 \le$ JobId ≤ 1000; $1 \le$ Deadline ≤ 1000; $1 \le$ Profit ≤ 1000.

Output:

- Đưa số lượng công việc tương ứng và lợi nhuận lớn nhất có thể đạt được.

Ví dụ:

Input	Output
2 4 1 4 20 2 1 10 3 1 40 4 1 30 5 1 2 100 2 1 19 3 2 27 4 1 25 5 1 15	2 60 2 127

BÀI 3. NỐI DÂY 1

Cho N sợi dây với độ dài khác nhau được lưu trong mảng A[]. Nhiệm vụ của bạn là nối N sợi dây thành một sợi sao cho tổng chi phí nối dây là nhỏ nhất. Biết chi phí nối sợi dây thứ i và sợi dây thứ j là tổng độ dài hai sợi dây A[i] và A[j].

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai dòng: dòng thứ nhất đưa vào số lượng sợi dây N; dòng tiếp theo đưa vào N số A[i] là độ dài của các sợi dây; các số được viết cách nhau một vài khoảng trống.
- T, N, A[i] thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 10^6$; $0 \le A[i] \le 10^6$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	Output
2 4 4 3 2 6 5 4 2 7 6 9	29 62

BÀI 4. NỐI DÂY 2

Cho N sợi dây với độ dài khác nhau được lưu trong mảng A[]. Nhiệm vụ của bạn là nối N sợi dây thành một sợi sao cho tổng chi phí nối dây là nhỏ nhất. Biết chi phí nối sợi dây thứ i và sợi dây thứ j là tổng độ dài hai sợi dây A[i] và A[j].

Dữ liệu vào

Dòng đầu ghi số bộ test T (T<10). Mỗi bộ test gồm 2 dòng. Dòng đầu tiên là số nguyên N ($N \leq 2*10^6$).

Dòng tiếp theo gồm N số nguyên dương c[i] ($1 \leq A[i] \leq 10^9$).

Kết quả

In ra đáp án của bộ test trên từng dòng, theo modulo 10^9+7.

Ví dụ:

Input:	Output
1 7 2 4 1 2 10 2 3	59

BÀI 5. SẮP ĐẶT XÂU KÝ TỰ 1

Cho xâu ký tự S bao gồm các ký tự in thường. Nhiệm vụ của bạn là kiểm tra xem ta có thể sắp đặt lại các ký tự trong S để hai ký tự giống nhau đều không kề nhau hay không? Đưa ra 1 nếu có thể sắp đặt lại các ký tự trong S thỏa mãn yêu cầu bài toán, ngược lại đưa ra -1.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là một xâu ký tự S được viết trên một dòng.
- T, S thỏa mãn ràng buộc: $1 \leq T \leq 100$; $1 \leq length(S) \leq 10000$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	Output
3 geeksforgeeks bbbabaaacd bbbbb	1 1 -1

BÀI 6. SẮP ĐẶT XÂU KÝ TỰ 2

Cho xâu ký tự S bao gồm các ký tự in thường và số D. Nhiệm vụ của bạn là kiểm tra xem ta có thể sắp đặt lại các ký tự trong S để tất cả các ký tự giống nhau đều có khoảng cách là D hay không? Đưa ra 1 nếu có thể sắp đặt lại các ký tự trong S thỏa mãn yêu cầu bài toán, ngược lại đưa ra -1.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai dòng: dòng thứ nhất là số D; dòng tiếp theo là xâu S.
- T, S, D thỏa mãn ràng buộc: $1 \leq T \leq 100$; $1 \leq length(S) \leq 10000$; $1 \leq D \leq 100$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	Output
2 2 ABB 2 AAA	1 -1

BÀI 7. MUA LƯƠNG THỰC

Giả sử bạn là một người nghèo trong địa phương của bạn. Địa phương của bạn có duy nhất một cửa hàng bán lương thực. Cửa hàng của bạn mở cửa tất cả các ngày trong tuần ngoại trừ chủ nhật. Cho bộ ba số N, S, M thỏa mãn ràng buộc sau:

- N : số đơn vị lương thực nhiều nhất bạn có thể mua trong ngày.
- S : số lượng ngày bạn cần được sử dụng lương thực để tồn tại.
- M : số đơn vị lương thực cần có mỗi ngày để bạn tồn tại.

Giả sử bạn đang ở ngày thứ 2 trong tuần và cần tồn tại trong S ngày tới. Hãy cho biết số lượng ngày ít nhất bạn cần phải mua lương thực từ của hàng để tồn tại hoặc bạn sẽ bị chết đói trong S ngày tới.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là bộ 3 số N, S, M được viết trên một dòng.
- T, N, S, M thỏa mãn ràng buộc: $1 \leq T \leq 100$; $1 \leq N, S, M \leq 30$.

Output:

- Đưa ra số ngày ít nhất bạn có thể mua lương thực để tồn tại hoặc đưa ra -1 nếu bạn bị chết đói.

Ví dụ:

Input	Output
2 16 10 2 20 10 30	2 -1

BÀI 8. BIỂU THỨC ĐÚNG

Cho một mảng S gồm 2×N ký tự, trong đó có N ký tự '[' và N ký tự ']'. Xâu S được gọi là viết đúng nếu S có dạng S2[S1] trong đó S, S2 là các xâu viết đúng. Nhiệm vụ của bạn là tìm số các phép đổi chỗ ít nhất các ký tự kề nhau của xâu S viết sai để S trở thành viết đúng. Ví dụ với xâu S ="]]][][" ta có số phép đổi chỗ kề nhau ít nhất là 2.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là một xâu S viết sai theo nguyên tắc kể trên.
- T, S thỏa mãn ràng buộc: $1 \leq T \leq 100$; $1 \leq \text{length}(S) \leq 100000$.

Output:

- Đưa kết quả trên một dòng.

Ví dụ:

Input	Output
2][][ ][][]	2 0

3.2. Bài tập về Thuật toán Chia và trị

BÀI 1. LŨY THỪA

Cho số nguyên dương N và K. Hãy tính N^K modulo 10^9+7.

Input:

Dòng đầu tiên là số lượng bộ test T ($T \leq 20$).

Mỗi test gồm 1 số nguyên N và K ($1 \leq N \leq 1000$, $1 \leq K \leq 10^9$).

Output:

Với mỗi test, in ra đáp án trên một dòng.

Ví dụ:

Input:	**Output**
2 2 3 4 2	8 16

BÀI 2. TÌM KIẾM NHỊ PHÂN

Cho dãy số A[] gồm có N phần tử đã được sắp xếp tăng dần và số K.

Nhiệm vụ của bạn là kiểm tra xem số K có xuất hiện trong dãy số hay không. Nếu có hãy in ra vị trí trong dãy A[], nếu không in ra "NO".

Input:

Dòng đầu tiên là số lượng bộ test T ($T \leq 10$).

Mỗi test bắt đầu bằng số nguyên N và K ($N \leq 100\,000$, $0 \leq K \leq 10^6$).

Dòng tiếp theo gồm N số nguyên A[i] ($0 \leq A[i] \leq 10^6$), các phần tử là riêng biệt.

Output:

Với mỗi test in ra trên một dòng đáp án tìm được.

Ví dụ:

Input:	**Output**
2 5 3 1 2 3 4 5 6 5 0 1 2 3 9 10	3 NO

BÀI 3. GẤP ĐÔI DÃY SỐ

Một dãy số tự nhiên bắt đầu bởi con số 1 và được thực hiện N-1 phép biến đổi "gấp đôi" dãy số như sau:

Với dãy số A hiện tại, dãy số mới có dạng A, x, A trong đó x là số tự nhiên bé nhất chưa xuất hiện trong A.

Ví dụ với 2 bước biến đổi, ta có [1] → [1 2 1] → [1 2 1 3 1 2 1].

Các bạn hãy xác định số thứ K trong dãy số cuối cùng là bao nhiêu?

Input:

Dòng đầu tiên là số lượng bộ test T ($T \leq 20$).

Mỗi test gồm số nguyên dương N và K ($1 \leq N \leq 50$, $1 \leq K \leq 2^N - 1$).

Output:

Với mỗi test, in ra đáp án trên một dòng.

Ví dụ:

Input	**Output**

2 3 2 4 8	2 4

Giải thích test 1: Dãy số thu được là [1, 2, 1, 3, 1, 2, 1].

Giải thích test 2: Dãy số thu được là [1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1].

BÀI 4. DÃY XÂU FIBONACI

Một dãy xâu ký tự G chỉ bao gồm các chữ cái A và B được gọi là dãy xâu Fibonacci nếu thỏa mãn tính chất: *G(1) = A; G(2) = B; G(n) = G(n-2)+G(n-1).* Với phép cộng (+) là phép nối hai xâu với nhau. Bài toán đặt ra là tìm ký tự ở vị trí thứ i (tính từ 1) của xâu Fibonacci thứ n.

Dữ liệu vào: Dòng 1 ghi số bộ test. Mỗi bộ test ghi trên một dòng 2 số nguyên N và i (1<N<93). Số i đảm bảo trong phạm vi của xâu G(N) và không quá 18 chữ số. **Kết quả:** Ghi ra màn hình kết quả tương ứng với từng bộ test.

Input	**Output**
2 6 4 8 19	A B

BÀI 5. SỐ FIBONACCI THỨ N

Dãy số Fibonacci được xác định bằng công thức như sau:

F[0] = 0, F[1] = 1;

F[n] = F[n-1] + F[n-2] với mọi n >= 2.

Các phần tử đầu tiên của dãy số là 0, 1, 1, 2, 3, 5, 8, ...

Nhiệm vụ của bạn là hãy xác định số Fibonaci thứ n. Do đáp số có thể rất lớn, in ra kết quả theo modulo 10^9+7.

Input:

Dòng đầu tiên là số lượng bộ test T ($T \leq 1000$).

Mỗi test bắt gồm một số nguyên N ($1 \leq N \leq 10^9$).

Output:

Với mỗi test, in ra đáp án trên một dòng.

Ví dụ:

Input:	**Output**
3 2 6 20	1 8 6765

BÀI 6. LŨY THỪA MA TRẬN

Cho ma trận vuông A kích thước N x N. Nhiệm vụ của bạn là hãy tính ma trận $X = A^K$ với K là số nguyên cho trước. Đáp số có thể rất lớn, hãy in ra kết quả theo modulo 10^9+7.

Input:

Dòng đầu tiên là số lượng bộ test T ($T \leq 100$).

Mỗi test bắt gồm một số nguyên N và K ($1 \leq N \leq 10$, $1 \leq K \leq 10^9$) là kích thước của ma trận và số mũ.

Output:

Với mỗi test, in ra kết quả của ma trận X.

Ví dụ:

Input:	**Output**
2 2 5 1 1 1 0 3 1000000000 1 2 3 4 5 6	8 5 5 3 597240088 35500972 473761863 781257150 154135232 527013321 965274212 272769492 580264779

7 8 9	

BÀI 7. DÃY CON LIÊN TIẾP CÓ TỔNG LỚN NHẤT

Cho mảng A[] gồm N số có cả các số âm và số dương. Nhiệm vụ của bạn là tìm mảng con liên tục có tổng lớn nhất của mảng. Ví dụ với mảng A[]={-2, -5, 6, -2,-3, 1, 5, -6} ta có kết quả là 7 tương ứng với dãy con {6, -2, -3, 1, 5}.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng thứ nhất đưa vào hai số N tương ứng với số phần tử của mảng; dòng tiếp theo đưa vào N số A[i]; các số được viết cách nhau một vài khoảng trống.
- T, N, A[i] thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 100$; $-100 \le A[i] \le 100$.

Output:

- Đưa ra tổng con liên tục lớn nhất của mỗi test theo từng dòng.

Ví dụ:

Input	Output
1 8 -2 -5 6 -2 -3 1 5 -6	7

3.3. Bài tập về Thuật toán Quy hoạch động

BÀI 1. XÂU CON CHUNG DÀI NHẤT

Cho 2 xâu S1 và S2. Hãy tìm xâu con chung dài nhất của 2 xâu này *(các phần tử không nhất thiết phải liên tiếp nhau).*

Input: Dòng đầu tiên là số lượng bộ test T ($T \le 20$). Mỗi test gồm hai dòng, mô tả xâu S1 và S2, mỗi xâu có độ dài không quá 1000 và chỉ gồm các chữ cái in hoa.

Output: Với mỗi test, in ra độ dài dãy con chung dài nhất trên một dòng.

Ví dụ:

Input	Output
2 AGGTAB GXTXAYB AA BB	4 0

Giải thích test 1: Dãy con chung là G, T, A, B.

BÀI 2. DÃY CON TĂNG DÀI NHẤT

Cho một dãy số nguyên gồm N phần tử A[1], A[2], ... A[N].

Biết rằng dãy con tăng là 1 dãy $A[i_1]$,... $A[i_k]$

thỏa mãn $i_1 < i_2 < ... < i_k$ và $A[i_1] < A[i_2] < .. < A[i_k]$.

Hãy cho biết dãy con tăng dài nhất của dãy này có bao nhiêu phần tử?

Input: Dòng 1 gồm 1 số nguyên là số N ($1 \le N \le 1000$). Dòng thứ 2 ghi N số nguyên A[1], A[2], .. A[N] ($1 \le A[i] \le 1000$).

Output: Ghi ra độ dài của dãy con tăng dài nhất.

Ví dụ:

Input	Output
6 1 2 5 4 6 2	4

BÀI 3. DÃY CON CÓ TỔNG BẰNG S

Cho N số nguyên dương tạo thành dãy $A=\{A_1, A_2, ..., A_N\}$. Tìm ra một dãy con của dãy A (không nhất thiết là các phần tử liên tiếp trong dãy) có tổng bằng S cho trước.

Input: Dòng đầu ghi số bộ test T (T<10). Mỗi bộ test có hai dòng, dòng đầu tiên ghi hai số nguyên dương N và S ($0 < N \le 200$) và S ($0 < S \le 40000$). Dòng tiếp theo lần lượt ghi N số hạng của dãy A là các số $A_1, A_2, ..., A_N$ ($0 < A_i \le 200$).

Output: Với mỗi bộ test, nếu bài toán vô nghiệm thì in ra "NO", ngược lại in ra "YES"

Ví dụ:

Input	Output
2 5 6 1 2 4 3 5 10 15 2 2 2 2 2 2 2 2 2 2	YES NO

BÀI 4. DÃY CON DÀI NHẤT CÓ TỔNG CHIA HẾT CHO K

Cho một dãy gồm n ($n \leq 1000$) số nguyên dương A_1, A_2, ..., A_n và số nguyên dương k ($k \leq 50$). Hãy tìm dãy con gồm nhiều phần tử nhất của dãy đã cho sao cho tổng các phần tử của dãy con này chia hết cho k.

Input: Dòng đầu ghi số bộ test T (T<10). Mỗi bộ test gồm 2 dòng. Dòng đầu tiên chứa hai số n, k. Dòng tiếp theo ghi n số của dãy A. Các số đều không vượt quá 100.

Output: Gồm 1 dòng duy nhất ghi số lượng phần tử của dãy con dài nhất thoả mãn. Dữ liệu vào luôn đảm bảo sẽ có ít nhất một dãy con có tổng chia hết cho k.

Ví dụ:

Input	Output
1 10 3 2 3 5 7 9 6 12 7 11 15	9

BÀI 5. XÂU CON ĐỐI XỨNG DÀI NHẤT

Cho xâu S chỉ bao gồm các ký tự viết thường và dài không quá 1000 ký tự.

Hãy tìm xâu con đối xứng dài nhất của S.

Input:

- Dòng đầu tiên là số lượng bộ test T ($T \leq 10$).
- Mỗi test gồm một xâu S có độ dài không vượt quá 1000, chỉ gồm các kí tự thường.

Output: Với mỗi test, in ra đáp án tìm được.

Ví dụ:

Input	Output
2 abcbadd aaaaa	5 5

BÀI 6. HÌNH VUÔNG LỚN NHẤT

Cho một bảng số N hàng, M cột chỉ gồm 0 và 1. Bạn hãy tìm hình vuông có kích thước lớn nhất, sao cho các số trong hình vuông toàn là số 1.

Input:

- Dòng đầu tiên là số lượng bộ test T ($T \leq 10$).
- Mỗi test bắt đầu bởi 2 số nguyên N, M ($1 \leq N, M \leq 500$).
- N dòng tiếp theo, mỗi dòng gồm M số mô tả một hàng của bảng.

Output:

- Với mỗi test, in ra đáp án là kích thước của hình vuông lớn nhất tìm được trên một dòng.

Ví dụ:

Input:	Output
2 6 5 0 1 1 0 1 1 1 0 1 0 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 0 0 0 0 2 2 0 0	3 0

0 0	

BÀI 7. CÁI TÚI

Một người có cái túi thể tích V (V<1000). Anh ta có N đồ vật cần mang theo (N≤1000), mỗi đồ vật có thể tích là A[i] (A[i]≤100) và giá trị là C[i] (C[i]≤100). Hãy xác định tổng giá trị lớn nhất của các đồ vật mà người đó có thể mang theo, sao cho tổng thể tích không vượt quá V.

Input

- Dòng đầu ghi số bộ test T (T<10)
- Mỗi bộ test gồm ba dòng. Dòng đầu ghi 2 số N và V. Dòng tiếp theo ghi N số của mảng A. Sau đó là một dòng ghi N số của mảng C.
- Dữ liệu vào luôn đảm bảo không có đồ vật nào có thể tích lớn hơn V.

Output

- Với mỗi bộ test, ghi trên một dòng giá trị lớn nhất có thể đạt được.

Ví dụ

Input	Output
1 15 10 5 2 1 3 5 2 5 8 9 6 3 1 4 7 8 1 2 3 5 1 2 5 8 7 4 1 2 3 2 1	15

BÀI 8. BIẾN ĐỔI XÂU

Cho hai xâu ký tự str1, str2 bao gồm các ký tự in thường và các thao tác dưới đây:

- **Insert:** chèn một ký tự bất kỳ vào str1.
- **Delete:** loại bỏ một ký tự bất kỳ trong str1.
- **Replace:** thay một ký tự bất kỳ trong str1.

Nhiệm vụ của bạn là đếm số các phép Insert, Delete, Replace ít nhất thực hiện trên str1 để trở thành str2.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test là bộ đôi hai xâu str1 và str2.

- T, str1, str2 thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le length(str1), length(str2) \le 100$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	Output
1 geek gesek	1

3.4. Bài tập về Thuật toán Sắp xếp và tìm kiếm

BÀI 1. SẮP XẾP ĐỔI CHỖ TRỰC TIẾP

Hãy thực hiện thuật toán sắp xếp đổi chỗ trực tiếp trên dãy N số nguyên. Ghi ra các bước thực hiện thuật toán. **Dữ liệu vào:** Dòng 1 ghi số N (không quá 100). Dòng 2 ghi N số nguyên dương (không quá 100). **Kết quả:** Ghi ra màn hình từng bước thực hiện thuật toán. Mỗi bước trên một dòng, các số trong dãy cách nhau đúng một khoảng trống.

Ví dụ:

Input	Output
4 5 7 3 2	Buoc 1: 2 7 5 3 Buoc 2: 2 3 7 5 Buoc 3: 2 3 5 7

BÀI 2. SẮP XẾP CHỌN

Hãy thực hiện thuật toán sắp xếp chọn trên dãy N số nguyên. Ghi ra các bước thực hiện thuật toán.

Dữ liệu vào: Dòng 1 ghi số N (không quá 100). Dòng 2 ghi N số nguyên dương (không quá 100).

Kết quả: Ghi ra màn hình từng bước thực hiện thuật toán. Mỗi bước trên một dòng, các số trong dãy cách nhau đúng một khoảng trống.

Ví dụ:

Input	Output
4 5 7 3 2	Buoc 1: 2 7 3 5 Buoc 2: 2 3 7 5 Buoc 3: 2 3 5 7

BÀI 3. SẮP XẾP CHÈN

Hãy thực hiện thuật toán sắp xếp chèn trên dãy N số nguyên. Ghi ra các bước thực hiện thuật toán.

Dữ liệu vào: Dòng 1 ghi số N (không quá 100). Dòng 2 ghi N số nguyên dương (không quá 100).

Kết quả: Ghi ra màn hình từng bước thực hiện thuật toán. Mỗi bước trên một dòng, các số trong dãy cách nhau đúng một khoảng trống.

Ví dụ:

Input	Output
4 5 7 3 2	Buoc 0: 5 Buoc 1: 5 7 Buoc 2: 3 5 7 Buoc 3: 2 3 5 7

BÀI 4. SẮP XẾP NỔI BỌT

Hãy thực hiện thuật toán sắp xếp nổi bọt trên dãy N số nguyên. Ghi ra các bước thực hiện thuật toán.

Dữ liệu vào: Dòng 1 ghi số N (không quá 100). Dòng 2 ghi N số nguyên dương (không quá 100).

Kết quả: Ghi ra màn hình từng bước thực hiện thuật toán. Mỗi bước trên một dòng, các số trong dãy cách nhau đúng một khoảng trống.

Ví dụ:

Input	Output

4 5 3 2 7	Buoc 1: 3 2 5 7 Buoc 2: 2 3 5 7

BÀI 5. MERGE SORT

Cho mảng A[] gồm N phần tử chưa được sắp xếp. Nhiệm vụ của bạn là sắp xếp các phần tử của mảng A[] theo thứ tự tăng dần bằng thuật toán Merge Sort.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai phần: phần thứ nhất đưa vào số N tương ứng với số phần tử của mảng A[]; phần thứ 2 là N số của mảng A[]; các số được viết cách nhau một vài khoảng trống.
- T, N, A[i] thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N, A[i] \le 10^6$.

Output:

- Đưa ra kết quả các test theo từng dòng.

Input	Output
2 5 4 1 3 9 7 10 10 9 8 7 6 5 4 3 2 1	1 3 4 7 9 1 2 3 4 5 6 7 8 9 10

BÀI 6. QUICK SORT

Cho mảng A[] gồm N phần tử chưa được sắp xếp. Nhiệm vụ của bạn là sắp xếp các phần tử của mảng A[] theo thứ tự tăng dần bằng thuật toán Quick Sort.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai phần: phần thứ nhất đưa vào số N tương ứng với số phần tử của mảng A[]; phần thứ 2 là N số của mảng A[]; các số được viết cách nhau một vài khoảng trống.
- T, N, A[i] thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N, A[i] \le 10^6$.

Output:

- Đưa ra kết quả các test theo từng dòng.

Input	Output
2 5 4 1 3 9 7 10 10 9 8 7 6 5 4 3 2 1	1 3 4 7 9 1 2 3 4 5 6 7 8 9 10

BÀI 7. TÌM KIẾM.

Cho mảng A[] gồm n phần tử đã được sắp xếp. Hãy đưa ra 1 nếu X có mặt trong mảng A[], ngược lại đưa ra -1.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai dòng: dòng thứ nhất đưa vào n, X là số các phần tử của mảng A[] và số X cần tìm; dòng tiếp theo đưa vào n số A[i] ($1 \le i \le n$) các số được viết cách nhau một vài khoảng trống.
- T, n, A, X thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N, X, A[i] \le 10^6$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Input:	Output:
2 5 16 2 4 7 9 16 7 98 1 22 37 47 54 88 96	1 -1

BÀI 8. TÌM SỐ CÒN THIẾU.

Cho mảng A[] gồm n-1 phần tử bao gồm các khác nhau từ 1, 2, .., n. Hãy tìm số không có mặt trong mảng A[].

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai dòng: dòng thứ nhất đưa vào n l; dòng tiếp theo đưa vào n-1 số A[i]; các số được viết cách nhau một vài khoảng trống.

- T, n, A thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N, A[i] \le 10^7$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Input:	Output:
2 5 1 2 3 5 10 1 2 3 4 5 6 7 8 10	4 9

BÀI 9. TÌM KIẾM TRONG DÃY SẮP XẾP VÒNG.

Một mảng được sắp được chia thành hai đoạn tăng dần được gọi là mảng sắp xếp vòng. Ví dụ mảng A[] = { 5, 6, 7, 8, 9, 10, 1, 2, 3, 4} là mảng sắp xếp vòng. Cho mảng A[] gồm n phần tử, hãy tìm vị trí của phần tử x trong mảng A[] với thời gian log(n).

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai dòng: dòng thứ nhất đưa vào n và x; dòng tiếp theo đưa vào n số A[i]; các số được viết cách nhau một vài khoảng trống.
- T, n, A[i], x thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N, x, A[i] \le 10^7$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Input:	Output:
2 10 3 5 6 7 8 9 10 1 2 3 4 10 3 1 2 3 4 5 6 7 8 9 10	9 3

BÀI 10. SỐ NHỎ NHẤT VÀ SỐ NHỎ THỨ HAI.

Cho mảng A[] gồm n phần tử, hãy đưa ra số nhỏ nhất và số nhỏ thứ hai của mảng. Nếu không có số nhỏ thứ hai, hãy đưa ra -1.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.

- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai dòng: dòng thứ nhất đưa vào n là số phần tử của mảng A[]; dòng tiếp theo đưa vào n số A[i]; các số được viết cách nhau một vài khoảng trống.
- T, n, A[i] thỏa mãn ràng buộc: $1 \leq T \leq 100$; $1 \leq N, A[i] \leq 10^7$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Input:	Output:
2 10 5 6 7 8 9 10 1 2 3 4 5 1 1 1 1 1	1 2 -1

BÀI TẬP CHƯƠNG 4. LÝ THUYẾT ĐỒ THỊ

4.1. Bài tập về Duyệt đồ thị

BÀI 1. DFS TRÊN ĐỒ THỊ VÔ HƯỚNG

Cho đồ thị vô hướng G=<V, E> được biểu diễn dưới dạng danh sách cạnh. Hãy viết thuật toán duyệt theo chiều sâu bắt đầu tại đỉnh $u \in V$ (DFS(u)=?)

Input:

- Dòng đầu tiên đưa vào T là số lượng bộ test.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm |E| +1 dòng: dòng đầu tiên đưa vào ba số |V|, |E| tương ứng với số đỉnh và số cạnh của đồ thị, và u là đỉnh xuất phát; |E| dòng tiếp theo đưa vào các bộ đôi $u \in V$, $v \in V$ tương ứng với một cạnh của đồ thị.
- T, |V|, |E| thỏa mãn ràng buộc: $1 \le T \le 200$; $1 \le |V| \le 10^3$; $1 \le |E| \le |V|(|V|-1)/2$;

Output:

- Đưa ra danh sách các đỉnh được duyệt theo thuật toán DFS(u) của mỗi test theo khuôn dạng của ví dụ dưới đây.

Ví dụ:

Input:	Output:
1 6 9 5 1 2 1 3 2 3 2 4 3 4 3 5 4 5 4 6 5 6	5 3 1 2 4 6

BÀI 2. BFS TRÊN ĐỒ THỊ VÔ HƯỚNG

Cho đồ thị vô hướng G=<V, E> được biểu diễn dưới dạng danh sách cạnh. Hãy viết thuật toán duyệt theo chiều rộng bắt đầu tại đỉnh $u \in V$ (BFS(u)=?)

Input:

- Dòng đầu tiên đưa vào T là số lượng bộ test.

- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào ba số |V|, |E|, u∈V tương ứng với số đỉnh, số cạnh và đỉnh bắt đầu duyệt; Dòng tiếp theo đưa vào các bộ đôi u∈V, v∈V tương ứng với một cạnh của đồ thị.
- T, |V|, |E| thỏa mãn ràng buộc: $1 \le T \le 200$; $1 \le |V| \le 10^3$; $1 \le |E| \le |V|(|V|-1)/2$;

Output:

- Đưa ra danh sách các đỉnh được duyệt theo thuật toán BFS(u) của mỗi test theo khuôn dạng của ví dụ dưới đây.

Ví dụ:

Input:	**Output:**
1 6 9 1 1 2 1 3 2 3 2 5 3 4 3 5 4 5 4 6 5 6	1 2 3 5 4 6

BÀI 3. TÌM ĐƯỜNG ĐI THEO DFS VỚI ĐỒ THỊ VÔ HƯỚNG

Cho đồ thị vô hướng G=<V, E> được biểu diễn dưới dạng danh sách cạnh. Hãy tìm đường đi từ đỉnh s∈V đến đỉnh t∈V trên đồ thị bằng thuật toán DFS.

Input:

- Dòng đầu tiên đưa vào T là số lượng bộ test.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào bốn số |V|, |E|, s∈V, t∈V tương ứng với số đỉnh, số cạnh, đỉnh u, đỉnh v; Dòng tiếp theo đưa vào các bộ đôi u∈V, v∈V tương ứng với một cạnh của đồ thị.
- T, |V|, |E| thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le |V| \le 10^3$; $1 \le |E| \le |V|(|V|-1)/2$;

Output:

- Đưa ra đường đi từ đỉnh s đến đỉnh t của mỗi test theo thuật toán DFS của mỗi test theo khuôn dạng của ví dụ dưới đây. Nếu không có đáp án, in ra -1.

Ví dụ:

Input:	**Output:**
1 6 9 1 6 1 2 1 3 2 3 2 5 3 4 3 5 4 5 4 6 5 6	1 2 3 4 5 6

BÀI 4. TÌM ĐƯỜNG ĐI THEO DFS VỚI ĐỒ THỊ CÓ HƯỚNG

Cho đồ thị có hướng G=<V, E> được biểu diễn dưới dạng danh sách cạnh. Hãy tìm đường đi từ đỉnh s∈V đến đỉnh t∈V trên đồ thị bằng thuật toán DFS.

Input:

- Dòng đầu tiên đưa vào T là số lượng bộ test.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào bốn số |V|, |E|, s∈V, t∈V tương ứng với số đỉnh, số cạnh, đỉnh u, đỉnh v; Dòng tiếp theo đưa vào các bộ đôi u∈V, v∈V tương ứng với một cạnh của đồ thị.
- T, |V|, |E| thỏa mãn ràng buộc: $1 \leq T \leq 100$; $1 \leq |V| \leq 10^3$; $1 \leq |E| \leq |V|(|V|-1)/2$;

Output:

- Đưa ra đường đi từ đỉnh s đến đỉnh t của mỗi test theo thuật toán DFS của mỗi test theo khuôn dạng của ví dụ dưới đây. Nếu không có đáp án, in ra -1.

Ví dụ:

Input:	**Output:**
1 6 9 1 6 1 2 2 5 3 1 3 2 3 5 4 3 5 4 5 6 6 4	1 2 5 6

BÀI 5. ĐẾM SỐ THÀNH PHẦN LIÊN THÔNG VỚI DFS.

Cho đồ thị vô hướng G=<V, E> được biểu diễn dưới dạng danh sách cạnh. Hãy tìm số thành phần liên thông của đồ thị bằng thuật toán DFS.

Input:

- Dòng đầu tiên đưa vào T là số lượng bộ test.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào hai số |V|, |E| tương ứng với số đỉnh và số cạnh; Dòng tiếp theo đưa vào các bộ đôi u∈V, v∈V tương ứng với một cạnh của đồ thị.
- T, |V|, |E| thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le |V| \le 10^3$; $1 \le |E| \le |V|(|V|-1)/2$;

Output:

- Đưa ra số thành phần liên thông của đồ thị bằng thuật toán DFS.

Ví dụ:

Input:	**Output:**
1 6 6 1 2 1 3 2 3 3 4 3 5 4 5	2

BÀI 6. LIỆT KÊ ĐỈNH TRỤ VỚI BFS

Cho đồ thị vô hướng liên thông G=<V, E> được biểu diễn dưới dạng danh sách cạnh. Sử dụng thuật toán BFS, hãy đưa ra tất cả các đỉnh trụ của đồ thị?

Input:

- Dòng đầu tiên đưa vào T là số lượng bộ test.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào hai số |V|, |E| tương ứng với số đỉnh và số cạnh; Dòng tiếp theo đưa vào các bộ đôi u∈V, v∈V tương ứng với một cạnh của đồ thị.
- T, |V|, |E| thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le |V| \le 10^3$; $1 \le |E| \le |V|(|V|-1)/2$;

Output:

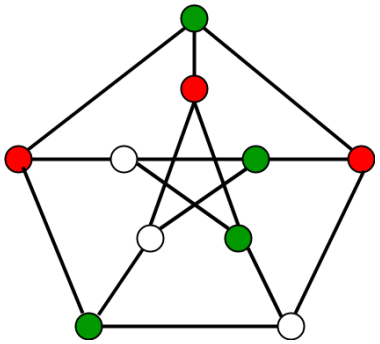
- Đưa ra danh sách các đỉnh trụ của mỗi test theo từng dòng.

Ví dụ:

Input:	**Output:**
1 5 5 1 2 1 3 2 3 2 5 3 4	2 3

BÀI 1. TÔ MÀU ĐỒ THỊ

Một trong những bài toán kinh điển của lý thuyết đồ thị là bài toán Tô màu đồ thị. Bài toán được phát biểu như sau: Cho đồ thị vô hướng G =<V, E> được biểu diễn dưới dạng danh sách cạnh và số M. Nhiệm vụ của bạn là kiểm tra xem đồ thị có thể tô màu các đỉnh bằng nhiều nhất M màu sao cho hai đỉnh kề nhau đều có màu khác nhau hay không?



Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai phần: phần thứ nhất đưa vào ba số V, E, M tương ứng với số đỉnh, số cạnh và số màu; phần thứ hai đưa vào các cạnh của đồ thị.
- T, V, E, M thỏa mãn ràng buộc: 1≤T ≤100; 1≤V≤10; 1≤ E ≤N(N-1), 1≤V≤N.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

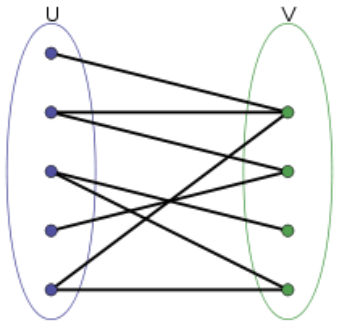
Ví dụ:

Input	**Output**
2 4 5 3 1 2 2 3 3 4 4 1 1 3 3 3 2 1 2 2 3	YES NO

1 3	

BÀI 3. ĐỒ THỊ HAI PHÍA

Đồ thị hai phía là một đồ thị đặc biệt, trong đó tập các đỉnh có thể được chia thành hai tập không giao nhau thỏa mãn điều kiện không có cạnh nối hai đỉnh bất kỳ thuộc cùng một tập. Cho đồ thị N đỉnh và M cạnh, bạn hãy kiểm tra đồ thị đã cho có phải là một đồ thị hai phía hay không?



Input:

- Dòng đầu tiên là số lượng bộ test T ($T \leq 20$).
- Mỗi test bắt đầu bởi số nguyên N và M ($1 \leq N, M \leq 1000$).
- M dòng tiếp theo, mỗi dòng gồm 2 số nguyên u, v cho biết có cạnh nối giữa đỉnh u và v.

Output:

- Với mỗi test, in ra "YES" nếu đồ thị đã cho là một đồ thị hai phía, in ra "NO" trong trường hợp ngược lại.

Ví dụ:

Input:	**Output**
2	YES

5 4 1 5 1 3 2 3 4 5 3 3 1 2 1 3 2 3	NO

BÀI 6. KẾT BẠN

Trường học X có N sinh viên, trong đó có M cặp là bạn bè của nhau. Bạn của bạn cũng là bạn, tức là nếu A là bạn của B, B là bạn của C thì A và C cũng là bạn bè của nhau.

Các bạn hãy xác định xem số lượng sinh viên nhiều nhất trong một nhóm bạn là bao nhiêu?

Input:

Dòng đầu tiên là số lượng bộ test T ($T \leq 20$).

Mỗi test bắt đầu bởi 2 số nguyên N và M ($N, M \leq 100\ 000$).

M dòng tiếp theo, mỗi dòng gồm 2 số nguyên u, v (u #v) cho biết sinh viên u là bạn của sinh viên v.

Output:

Với mỗi test, in ra đáp án tìm được trên một dòng.

Ví dụ:

Input:	**Output**
2 3 2 1 2 2 3	3 7

10 12 1 2 3 1 3 4 5 4 3 5 4 6 5 2 2 1 7 1 1 2 9 10 8 9	

BÀI 7. MẠNG XÃ HỘI

Tý đang xây dựng một mạng xã hội và mời các bạn của mình dùng thử. Bạn của bạn cũng là bạn. Vì vậy, Tý muốn mạng xã hội của mình là hoàn hảo, tức với mọi bộ ba X, Y, Z, nếu X kết bạn với Y, Y kết bạn với Z thì X và Z cũng phải là bạn bè của nhau trên mạng xã hội.

Các bạn hãy xác định xem mạng xã hội hiện tại của Tý có là hoàn hảo hay không? Nếu có hãy in ra "YES", "NO" trong trường hợp ngược lại.

Input:

- Dòng đầu tiên là số lượng bộ test T ($T \leq 20$).
- Mỗi test bắt đầu bởi 2 số nguyên N và M ($N, M \leq 100\ 000$).
- M dòng tiếp theo, mỗi dòng gồm 2 số nguyên u, v (u #v) cho biết u và v là kết bạn với nhau trên mạng xã hội của Tý.

Output:

- Với mỗi test, in ra đáp án tìm được trên một dòng.

Ví dụ:

Input:	Output
3 4 3 1 3 3 4 1 4 4 4 3 1 2 3 3 4 1 2 10 4 4 3 5 10 8 9 1 2	YES NO YES

4.2. Bài tập về đồ thị EULER và đồ thị HAMILTON

BÀI 1. ĐƯỜNG ĐI HAMILTON

Đường đi đơn trên đồ thị có hướng hoặc vô hướng đi qua tất cả các đỉnh của đồ thị mỗi đỉnh đúng một lần được gọi là đường đi Hamilton. Cho đồ thị vô hướng G = <V, E>, hãy kiểm tra xem đồ thị có đường đi Hamilton hay không?

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.

- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai phần: phần thứ nhất đưa vào hai số V, E tương ứng với số đỉnh, số cạnh của đồ thị; phần thứ hai đưa vào các cạnh của đồ thị.
- T, V, E thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le V \le 10$; $1 \le E \le 15$.

Output:

- Đưa ra 1 hoặc 0 tương ứng với test có hoặc không có đường đi Hamilton theo từng dòng.

Ví dụ:

Input	**Output**
2 4 4 1 2 2 3 3 4 2 4 4 3 1 2 2 3 2 4	1 0

BÀI 2. ĐƯỜNG ĐI VÀ CHU TRÌNH EULER VỚI ĐỒ THỊ VÔ HƯỚNG

Cho đồ thị vô hướng liên thông G=<V, E> được biểu diễn dưới dạng danh sách cạnh. Hãy kiểm tra xem đồ thị có đường đi Euler hay chu trình Euler hay không?

Đường đi Euler bắt đầu tại một đỉnh, và kết thúc tại một đỉnh khác.
Chu trình Euler bắt đầu tại một đỉnh, và kết thúc chu trình tại chính đỉnh đó.

Input:

- Dòng đầu tiên đưa vào T là số lượng bộ test.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào hai số |V|, |E| tương ứng với số đỉnh, số cạnh của đồ thị; Dòng tiếp theo đưa vào các bộ đôi $u \in V$, $v \in V$ tương ứng với một cạnh của đồ thị.
- T, |V|, |E| thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le |V| \le 10^3$; $1 \le |E| \le |V|(|V|-1)/2$;

Output:

- Đưa ra 1, 2, 0 kết quả mỗi test theo từng dòng tương ứng với đồ thị có đường đi Euler, chu trình Euler và trường hợp không tồn tại.

Ví dụ:

Input:	**Output:**
2	2

6 10 1 2 1 3 2 3 2 4 2 5 3 4 3 5 4 5 4 6 5 6 6 9 1 2 1 3 2 3 2 4 2 5 3 4 3 5 4 5 4 6	1

BÀI 3. CHU TRÌNH EULER TRONG ĐỒ THỊ CÓ HƯỚNG

Cho đồ thị có hướng liên thông yếu G=<V, E> được biểu diễn dưới dạng danh sách cạnh. Hãy kiểm tra xem đồ thị có chu trình Euler hay không?

Input:

- Dòng đầu tiên đưa vào T là số lượng bộ test.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào hai số |V|, |E| tương ứng với số đỉnh, số cạnh của đồ thị; Dòng tiếp theo đưa vào các bộ đôi u∈V, v∈V tương ứng với một cạnh của đồ thị.
- T, |V|, |E| thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le |V| \le 10^3$; $1 \le |E| \le |V|(|V|-1)/2$;

Output:

- Đưa ra 1, 0 kết quả mỗi test theo từng dòng tương ứng với đồ thị có chu trình Euler và trường hợp không tồn tại đáp án.

Ví dụ:

Input:	**Output:**
2 6 10 1 2 2 4 2 5 3 1 3 2 4 3 4 5 5 3 5 6 6 4 3 3 1 2 2 3 1 3	1 0

BÀI 4. KIỂM TRA ĐỒ THỊ CÓ PHẢI LÀ CÂY HAY KHÔNG

Một đồ thị N đỉnh là một cây, nếu như nó có đúng N-1 cạnh và giữa 2 đỉnh bất kì, chỉ tồn tại duy nhất 1 đường đi giữa chúng.

Cho một đồ thị N đỉnh và N-1 cạnh, hãy kiểm tra đồ thị đã cho có phải là một cây hay không?

Input:

- Dòng đầu tiên là số lượng bộ test T ($T \le 20$).

- Mỗi test bắt đầu bởi số nguyên N ($1 \leq N \leq 1000$).
- N-1 dòng tiếp theo, mỗi dòng gồm 2 số nguyên u, v cho biết có cạnh nối giữa đỉnh u và v.

Output:

- Với mỗi test, in ra "YES" nếu đồ thị đã cho là một cây, in ra "NO" trong trường hợp ngược lại.

Ví dụ:

Input	**Output**
2 4 1 2 1 3 2 4 4 1 2 1 3 2 3	YES NO

4.3. Bài tập về đồ thị trọng số

BÀI 1. KRUSKAL

Cho đồ thị vô hướng có trọng số G=<V, E, W>. Nhiệm vụ của bạn là hãy xây dựng một cây khung nhỏ nhất của đồ thị bằng thuật toán Kruskal.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai phần: phần thứ nhất đưa vào hai số V, E tương ứng với số đỉnh và số cạnh của đồ thị; phần thứ 2 đưa vào E cạnh của đồ thị, mỗi cạnh là một bộ 3: đỉnh đầu, đỉnh cuối và trọng số của cạnh.

- T, S, D thỏa mãn ràng buộc: 1≤T≤100; 1≤V≤100; 1≤E, W≤1000.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	**Output**
2 3 3 1 2 5 2 3 3 1 3 1 2 1 1 2 5	4 5

BÀI 2. PRIM

Cho đồ thị vô hướng có trọng số G=<V, E, W>. Nhiệm vụ của bạn là hãy xây dựng một cây khung nhỏ nhất của đồ thị bằng thuật toán PRIM.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai phần: phần thứ nhất đưa vào hai số V, E tương ứng với số đỉnh và số cạnh của đồ thị; phần thứ 2 đưa vào E cạnh của đồ thị, mỗi cạnh là một bộ 3: đỉnh đầu, đỉnh cuối và trọng số của cạnh.
- T, S, D thỏa mãn ràng buộc: 1≤T≤100; 1≤V≤100; 1≤E, W≤1000.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	**Output**
2 3 3 1 2 5 2 3 3	4 5

1 3 1 2 1 1 2 5	

BÀI 3. BRUVKA

Cho đồ thị vô hướng có trọng số G=<V, E, W>. Nhiệm vụ của bạn là hãy xây dựng một cây khung nhỏ nhất của đồ thị bằng thuật toán Bruvka.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm hai phần: phần thứ nhất đưa vào hai số V, E tương ứng với số đỉnh và số cạnh của đồ thị; phần thứ 2 đưa vào E cạnh của đồ thị, mỗi cạnh là một bộ 3: đỉnh đầu, đỉnh cuối và trọng số của cạnh.
- T, S, D thỏa mãn ràng buộc: 1≤T≤100; 1≤V≤100; 1≤E, W≤1000.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	**Output**
2 3 3 1 2 5 2 3 3 1 3 1 2 1 1 2 5	4 5

BÀI 4. DIJKSTRA.

Cho đồ thị có trọng số không âm G=<V, E> được biểu diễn dưới dạng danh sách cạnh trọng số. Hãy viết chương trình tìm đường đi ngắn nhất từ đỉnh u∈V đến tất cả các đỉnh còn lại trên đồ thị.

Input:

- Dòng đầu tiên đưa vào T là số lượng bộ test.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm |E|+1 dòng: dòng đầu tiên đưa vào hai ba số |V|, |E| tương ứng với số đỉnh và u∈V là đỉnh bắt đầu; |E|

dòng tiếp theo mỗi dòng đưa vào bộ ba u∈V, v∈V, w tương ứng với một cạnh cùng với trọng số canh của đồ thị.

- T, |V|, |E| thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le |V| \le 10^3$; $1 \le |E| \le |V|(|V|-1)/2$;

Output:

- Đưa ra kết quả của mỗi test theo từng dòng. Kết quả mỗi test là trọng số đường đi ngắn nhất từ đỉnh u đến các đỉnh còn lại của đồ thị theo thứ tự tăng dần các đỉnh.

Ví dụ:

Input:	**Output:**
1 9 12 1 1 2 4 1 8 8 2 3 8 2 8 11 3 4 7 3 6 4 3 9 2 4 5 9 4 6 14 5 6 10 6 7 2 6 9 6	0 4 12 19 21 11 9 8 14

BÀI 5. BELLMAN-FORD.

Cho đồ thị có hướng, có trọng số có thể âm hoặc không âm G=<V, E> được biểu diễn dưới dạng danh sách cạnh. Hãy viết chương trình tìm đường đi ngắn nhất từ đỉnh u∈V đến tất cả các đỉnh còn lại trên đồ thị.

Input:

- Dòng đầu tiên đưa vào T là số lượng bộ test.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm |E|+1 dòng: dòng đầu tiên đưa vào hai ba số |V|, |E| tương ứng với số đỉnh và u∈V là đỉnh bắt đầu; |E|

dòng tiếp theo mỗi dòng đưa vào bộ ba u∈V, v∈V, w tương ứng với một cạnh cùng với trọng số canh của đồ thị.

- T, |V|, |E| thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le |V| \le 10^3$; $1 \le |E| \le |V|(|V|-1)/2$;

Output:

- Đưa ra kết quả của mỗi test theo từng dòng. Kết quả mỗi test là trọng số đường đi ngắn nhất từ đỉnh u đến các đỉnh còn lại của đồ thị theo thứ tự tăng dần các đỉnh. Nếu tồn tại chu trình âm, in ra -1. Nếu không có đường đi ngắn nhất tới đỉnh u, in ra INFI.

Ví dụ:

Input:	**Output:**
2 5 8 1 1 2 -1 1 3 4 2 3 3 2 4 2 2 5 2 4 2 1 4 3 5 5 4 -3 3 3 1 1 2 -1 2 3 2 3 1 -2	0 -1 2 -2 1 -1

BÀI 6. NỐI ĐIỂM

Cho N điểm trên mặt phẳng Oxy. Để vẽ được đoạn thẳng nối A và B sẽ tốn chi phí tương đương với khoảng cách từ A tới B.

Nhiệm vụ của bạn là nối các điểm với nhau, sao cho N điểm đã cho tạo thành 1 thành phần liên thông duy nhất và chi phí để thực hiện là nhỏ nhất có thể.

Input:

- Dòng đầu tiên là số lượng bộ test T ($T \leq 20$).
- Mỗi test bắt đầu bởi số nguyên N ($1 \leq N \leq 100$).
- N dòng tiếp theo, mỗi dòng gồm 2 số thực x[i], y[i] là tọa độ của điểm thứ i ($|x[i]|, |y[i]| \leq 100$).

Output:

- Với mỗi test, in ra chi phí nhỏ nhất tìm được với độ chính xác 6 chữ số thập phân sau dấu phẩy.

Ví dụ:

Input:	**Output**
1 3 1.0 1.0 2.0 2.0 2.0 4.0	3.414214

BÀI 7. ĐƯỜNG ĐI NGẮN NHẤT 1

Cho đơn đồ thị vô hướng liên thông G = (V, E) gồm N đỉnh và M cạnh, các đỉnh được đánh số từ 1 tới N và các cạnh được đánh số từ 1 tới M.

Có Q truy vấn, mỗi truy vấn yêu cầu bạn tìm đường đi ngắn nhất giữa đỉnh X[i] tới Y[i].

Input:

- Dòng đầu tiên hai số nguyên N và M ($1 \leq N \leq 100$, $1 \leq M \leq N*(N-1)/2$).
- M dòng tiếp theo, mỗi dòng gồm 3 số nguyên u, v, c cho biết có cạnh nối giữa đỉnh u và v có độ dài bằng c ($1 \leq c \leq 1000$).
- Tiếp theo là số lượng truy vấn Q ($1 \leq Q \leq 100\ 000$).
- Q dòng tiếp theo, mỗi dòng gồm 2 số nguyên X[i], Y[i].

Output:

- Với mỗi truy vấn, in ra đáp án là độ dài đường đi ngắn nhất tìm được.

Ví dụ:

Input:	Output
5 6 1 2 6 1 3 7 2 4 8 3 4 9 3 5 1 4 5 2 3 1 5 2 5 4 3	8 10 3

BÀI 8. ĐƯỜNG ĐI NGẮN NHẤT 2

Cho đồ thị vô hướng liên thông G = (V, E) gồm N đỉnh và M cạnh, các đỉnh được đánh số từ 1 tới N và các cạnh được đánh số từ 1 tới M.

Nhiệm vụ của bạn là hãy tìm đường đi ngắn nhất từ 1 tới N và đếm xem có bao nhiêu tuyến đường có độ dài ngắn nhất như vậy?

Input:

- Dòng đầu ghi số bộ test, không quá 10. Mỗi bộ test gồm:
  - Dòng đầu tiên hai số nguyên N và M ($1 \le N \le 10^5$, $1 \le M \le \max(N*(N-1)/2, 10^6)$).
  - M dòng tiếp theo, mỗi dòng gồm 3 số nguyên u, v, c cho biết có cạnh nối giữa đỉnh u và v có độ dài bằng c ($1 \le c \le 10^6$).

Output:

Với mỗi test, in ra 2 số nguyên là độ dài đường đi ngắn nhất và số lượng đường đi ngắn nhất. Input đảm bảo số lượng đường đi ngắn nhất không vượt quá 10^{18}.

Ví dụ:

Input	Output
5 6 1 2 6 1 3 7 2 4 2 3 4 9 3 5 3 4 5 2	10 2

Có 2 tuyến đường ngắn nhất: 1→ 3 → 5 và 1 → 2 → 4 → 5.

BÀI 9. BẢNG SỐ

Cho một bảng số kích thước N x M. Chi phí khi đi qua ô (i,j) bằng A[i][j]. Nhiệm vụ của bạn là hãy tìm một đường đi từ ô (1, 1) tới ô (N, M) sao cho chi phí là nhỏ nhất. Tại mỗi ô, bạn được phép đi sang trái, sang phải, đi lên trên và xuống dưới.

Input:

- Dòng đầu tiên là số lượng bộ test T ($T \leq 20$).
- Mỗi test bắt đầu bởi hai số nguyên N và M ($1 \leq N, M \leq 500$).
- N dòng tiếp theo, mỗi dòng gồm M số nguyên A[i][j] ($0 \leq A[i][j] \leq 9$).

Output:

- Với mỗi test, in ra một số nguyên là chi phí nhỏ nhất cho đường đi tìm được.

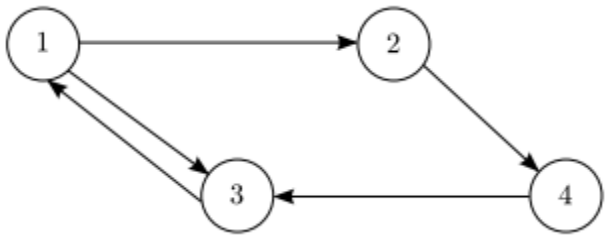
Ví dụ:

Input:	Output
3 4 5	24 15 13

0 3 1 2 9 7 3 4 9 9 1 7 5 5 3 2 3 4 2 5 1 6 0 1 2 3 4 5 5 5 1 1 1 9 9 9 9 1 9 9 1 1 1 9 9 1 9 9 9 9 1 1 1 1 1	

BÀI 10. ĐƯỜNG ĐI TRUNG BÌNH

Cho một đồ thị có hướng gồm N đỉnh và M cạnh. Nhiệm vụ của bạn là hãy tính khoảng cách trung bình ngắn nhất giữa hai nút bất kì nếu như chúng liên thông với nhau. Input đảm bảo rằng trong một nhóm liên thông, nếu như u đi tới được v thì v cũng đi tới được v với mọi cặp u, v.



Input:Dòng đầu tiên là số lượng bộ test T ($T \leq 20$). Mỗi test bắt đầu bởi hai số nguyên N và M ($1 \leq N \leq 100$, $M \leq N*(N-1)/2$). M dòng tiếp theo, mỗi dòng gồm 2 số nguyên u, v cho biết có cạnh nối đơn hướng từ u tới v.

Output: Với mỗi test, in ra đáp án tìm được với độ chính xác 2 chữ số sau dấu phảy.

Ví dụ:

Input:	**Output**

2 4 5 1 2 2 4 1 3 3 1 4 3 7 5 1 2 1 4 4 2 2 7 7 1	1.83 1.75

Giải thích test 1: Ta có

d(1, 2) = 1, d(1, 3) = 1, d(1, 4) = 2; d (2, 1) = 3, d(2, 3) = 2, d(2, 4) = 1;

d(3, 1) = 1, d(3, 2) = 2, d(3, 4) = 3; d(4, 1) = 2, d(4, 2) = 3, d(4, 3) = 1.

Trung bình bằng 22/12 = 1.83

BÀI TẬP CHƯƠNG 5. CÁC CẤU TRÚC DỮ LIỆU CƠ BẢN

5.1. Bài tập về Ngăn xếp

BÀI 1. TỔNG ĐA THỨC

Cho hai đa thức có bậc không quá 10000 (chỉ viết ra các phần tử có hệ số khác 0). Hãy sử dụng danh sách liên kết đơn để viết chương trình tính tổng hai đa thức đó.

Dữ liệu vào: Dòng đầu ghi số bộ test. Mỗi bộ test có hai dòng, mỗi dòng ghi một đa thức theo mẫu như trong ví dụ. Số phần tử của đa thức không quá 20.

Chú ý: Bậc của các hạng tử luôn theo thứ tự giảm dần, trong đa thức chỉ có phép cộng và luôn được viết đầy đủ hệ số + số mũ (kể cả mũ 0).

Kết quả: Ghi ra một dòng đa thức tổng tính được (theo mẫu như ví dụ)

Ví dụ:

Input	Output
1 3*x^8 + 7*x^2 + 4*x^0 11*x^6 + 9*x^2 + 2*x^1 + 3*x^0	3*x^8 + 11*x^6 + 16*x^2 + 2*x^1 + 7*x^0

BÀI 2. ĐẢO TỪ

Cho một xâu ký tự str bao gồm nhiều từ trong xâu. Hãy đảo ngược từng từ trong xâu?

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T;
- Những dòng tiếp theo mỗi dòng đưa vào một bộ test. Mỗi bộ test là một dòng ghi lại nhiều từ trong xâu str.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ràng buộc:

- T, str thỏa mãn ràng buộc: $1 \le T \le 100$; $2 \le \text{length(str)} \le 10^6$.

Ví dụ:

Input	Output
2 ABC DEF	CBA FED 321 654

123 456	

BÀI 3. KIỂM TRA DÃY NGOẶC ĐÚNG

Cho một xâu chỉ gồm các kí tự '(', ')', '[', ']', '{', '}'. Một dãy ngoặc đúng được định nghĩa như sau:

- Xâu rỗng là 1 dãy ngoặc đúng.
- Nếu A là 1 dãy ngoặc đúng thì (A), [A], {A} là 1 dãy ngoặc đúng.
- Nếu A và B là 2 dãy ngoặc đúng thì AB là 1 dãy ngoặc đúng.

Cho một xâu S. Nhiệm vụ của bạn là xác định xâu S có là dãy ngoặc đúng hay không?

Input:

Dòng đầu tiên là số lượng bộ test T ($T \leq 20$).

Mỗi test gồm 1 xâu S có độ dài không vượt quá 100 000.

Output:

Với mỗi test, in ra "YES" nếu như S là dãy ngoặc đúng, in ra "NO" trong trường hợp ngược lại.

Ví dụ:

Input:	**Output**
2 [()]{}{[()()]()} [(])	YES NO

BÀI 4. BIẾN ĐỔI TRUNG TỐ - HẬU TỐ

Hãy viết chương trình chuyển đổi biểu thức biểu diễn dưới dạng trung tố về dạng hậu tố.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T;
- Những dòng tiếp theo mỗi dòng đưa vào một bộ test. Mỗi bộ test là một biểu thức tiền tố exp.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ràng buộc:

- T, exp thỏa mãn ràng buộc: 1≤T≤100; 2≤length(exp)≤10.

Ví dụ:

Input	Output
2 (A+(B+C) ((A*B)+C)	ABC++ AB*C+

BÀI 5. TÍNH GIÁ TRỊ BIỂU THỨC HẬU TỐ

Hãy viết chương trình chuyển tính toán giá trị của biểu thức hậu tố.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T;
- Những dòng tiếp theo mỗi dòng đưa vào một bộ test. Mỗi bộ test là một biểu thức hậu tố exp. Các số xuất hiện trong biểu thức là các số đơn có 1 chữ số.

Output:

- Đưa ra kết quả mỗi test theo từng dòng, chỉ lấy giá trị phần nguyên.

Ràng buộc:

- T, exp thỏa mãn ràng buộc: 1≤T≤100; 2≤length(exp)≤20.

Ví dụ:

Input	**Output**
2 231*+9- 875*+9-	-4 34

BÀI 6. PHẦN TỬ BÊN PHẢI ĐẦU TIÊN LỚN HƠN

Cho dãy số A[] gồm N phần tử. Với mỗi A[i], bạn cần tìm phần tử bên phải đầu tiên lớn hơn nó. Nếu không tồn tại, in ra -1.

Input:

Dòng đầu tiên là số lượng bộ test T ($T \leq 20$).

Mỗi test bắt đầu bởi số nguyên N ($1 \leq N \leq 100000$).

Dòng tiếp theo gồm N số nguyên A[i] ($0 \leq A[i] \leq 10^9$).

Output:

Với mỗi test, in ra trên một dòng N số R[i], với R[i] là giá trị phần tử đầu tiên lớn hơn A[i].

Ví dụ

Input	Output
3 4 4 5 2 25 3 2 2 2 4 4 4 5 5	5 25 25 -1 -1 -1 -1 5 5 -1 -1

BÀI 7. KIỂM TRA BIỂU THỨC SỐ HỌC

Cho biểu thức số học, hãy cho biết biểu thức số học có dư thừa các cặp ký hiệu '(',') ' hay không?

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T;
- Những dòng tiếp theo mỗi dòng đưa vào một bộ test. Mỗi bộ test là một biểu thức tiền tố exp.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ràng buộc:

- T, exp thỏa mãn ràng buộc: $1 \leq T \leq 100$; $2 \leq \text{length(exp)} \leq 20$.

Ví dụ:

Input	Output
3 ((a+b)) (a + (b)/c) (a + b*(c-d))	Yes Yes No

BÀI 8. SỬA LẠI DÃY NGOẶC

Cho một xâu chỉ gồm các kí tự ‘(‘, ‘) và có độ dài chẵn. Hãy đếm số lượng dấu ngoặc cần phải đổi chiều ít nhất, sao cho xâu mới thu được là một dãy ngoặc đúng.

Input:

Dòng đầu tiên là số lượng bộ test T ($T \leq 20$).

Mỗi test gồm 1 xâu S có độ dài không vượt quá 100 000, chỉ gồm dấu (và).

Output:

Với mỗi test, in ra đáp án tìm được trên một dòng.

Ví dụ:

Input:	Output
4))((((((((())))(())(((	2 2 1 3

BÀI 9. BIỂU THỨC TƯƠNG ĐƯƠNG

Cho biểu thức đúng P chỉ bao gồm các phép toán +, -, các toán hạng cùng với các ký tự ‘(’, ‘)’. Hãy bỏ tất cả các ký tự ‘(’, ‘)’ trong P để nhận được biểu thức tương đương. Ví dụ với P = a – (b + c) ta có kết quả P = a – b – c .

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T;
- Những dòng tiếp theo mỗi dòng đưa vào một bộ test. Mỗi bộ test là một biểu thức P được viết trên một dòng.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ràng buộc:

- T, P thỏa mãn ràng buộc: $1 \leq T \leq 100$; $1 \leq length(P) \leq 10^3$.

Ví dụ:

Input	Output
2 a-(b+c) a-(b-c-(d+e))-f	a-b-c a-b+c+d+e-f

BÀI 10. SO SÁNH BIỂU THỨC

Cho P1, P2 là hai biểu thức đúng chỉ bao gồm các ký tự mở ngoặc '(' hoặc đóng ngoặc ')' và các toán hạng in thường. Nhiệm vụ của bạn là định xem P1 và P2 có giống nhau hay không.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T;
- Những dòng tiếp theo mỗi dòng đưa vào một bộ test. Mỗi bộ test gồm hai dòng: dòng thứ nhất đưa vào P1, dòng tiếp theo đưa vào P2.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ràng buộc:

- T, P thỏa mãn ràng buộc: $1 \leq T \leq 100$; $1 \leq length(P) \leq 100$.

Ví dụ:

Input	Output
2 -(a+b+c) -a-b-c a-b-(c-d) a-b-c-d	YES NO

5.2. Bài tập về Hàng đợi

BÀI 1. SỐ NHỊ PHÂN TỪ 1 ĐẾN N

Cho số tự nhiên n. Hãy in ra tất cả các số nhị phân từ 1 đến n.

Input:

- Dòng đầu tiên ghi lại số lượng test T ($T \leq 100$).
- Mỗi test là một số tự nhiên n được ghi trên một dòng ($n \leq 10000$).

Output:

- Đưa ra kết quả mỗi test trên một dòng.

Ví dụ:

Input	Output
2 2 5	1 10 1 10 11 100 101

BÀI 2. BIẾN ĐỔI S – T

Cho hai số nguyên dương S và T (S, T<10000) và hai thao tác (a), (b) dưới đây:

Thao tác (a): Trừ S đi 1 (S = S-1) ;

Thao tác (b): Nhân S với 2 (S = S*2);

Hãy dịch chuyển S thành T sao cho số lần thực hiện các thao tác (a), (b) là ít nhất. Ví dụ với S =2, T=5 thì số các bước ít nhất để dịch chuyển S thành T thông qua 4 thao tác sau:

Thao tác (a): 2*2 = 4;

Thao tác (b): 4-1 = 3;

Thao tác (a): 3*2 = 6;

Thao tác (b): 6-1 = 5;

Input:

- Dòng đầu tiên ghi lại số tự nhiên T là số lượng Test;
- T dòng kế tiếp mỗi dòng ghi lại một bộ Test. Mỗi test là một bộ đôi S và T.

Output: Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	Output
3 2 5 3 7 7 4	4 4 3

BÀI 3 BIẾN ĐỔI SỐ NGUYÊN TỐ

Cho cặp số S và T là các số nguyên tố có 4 chữ số (Ví dụ S = 1033, T = 8197 là các số nguyên tố có 4 chữ số). Hãy viết chương trình tìm cách dịch chuyển S thành T thỏa mãn đồng thời những điều kiện dưới đây:

a) Mỗi phép dịch chuyển chỉ được phép thay đổi một chữ số của số ở bước trước đó (ví dụ nếu S=1033 thì phép dịch chuyển S thành 1733 là hợp lệ);
b) Số nhận được cũng là một số nguyên tố có 4 chữ số (ví dụ nếu S=1033 thì phép dịch chuyển S thành 1833 là không hợp lệ, và S dịch chuyển thành 1733 là hợp lệ);
c) Số các bước dịch chuyển là ít nhất.

Ví dụ số các phép dịch chuyển ít nhất để S = 1033 thành T = 8179 là 6 bao gồm các phép dịch chuyển như sau:

8179 ← 8779← 3779← 3739← 3733← 1733← 1033.

Input:

- Dòng đầu tiên đưa vào số lượng test T (T≤100)
- Những dòng kế tiếp mỗi dòng đưa vào một test. Mỗi test là một bộ đôi S, T.

Output:

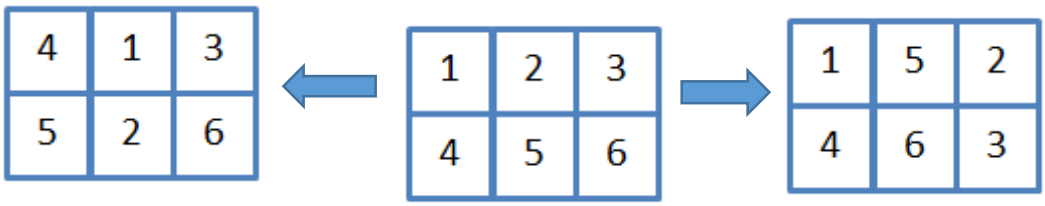
- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	**Output**
2 1033 8179 1033 8779	6 5

BÀI 4. QUAY HÌNH VUÔNG

Có một chiếc bảng hình chữ nhật với 6 miếng ghép, trên mỗi miếng ghép được điền một số nguyên trong khoảng từ 1 đến 6. Tại mỗi bước, chọn một hình vuông (bên trái hoặc bên phải), rồi quay theo chiều kim đồng hồ.



Yêu cầu: Cho một trạng thái của bảng, hãy tính số phép biến đổi ít nhất để đưa bảng đến trạng thái đích.

Input:

- Dòng đầu ghi số bộ test (không quá 10). Mỗi bộ test gồm hai dòng:
    - Dòng đầu tiên chứa 6 số là trạng thái bảng ban đầu (thứ tự từ trái qua phải, dòng 1 tới dòng 2).
    - Dòng thứ hai chứa 6 số là trạng thái bảng đích (thứ tự từ trái qua phải, dòng 1 tới dòng 2).

Output:

- Với mỗi test, in ra một số nguyên là đáp số của bài toán.

Ví dụ:

Input	**Output**
1 1 2 3 4 5 6 4 1 2 6 5 3	2

BÀI 5. BIẾN ĐỔI SỐ TỰ NHIÊN

Cho số tự nhiên N (N<10^9) và hai phép biến đổi (a), (b) dưới đây.

- **Thao tác (a)**: Trừ N đi 1 (N=N-1). Ví dụ N=17, thao tác (a) biến đổi N = N-1 =16.
- **Thao tác (b)**: N = max(u,v) nếu u*v =N (u>1, v>1). Ví dụ N=16, thao tác (b) có thể biến đổi N = max(2, 8)=8 hoặc N=max(4, 4)=4.

Chỉ được phép sử dụng hai thao tác (a) hoặc (b), hãy biến đổi N thành 1 sao số các thao tác (a), (b) được thực hiện ít nhất. Ví dụ với N=17, số các phép (a), (b) nhỏ nhất biến đổi N thành 1 là 4 bước như sau:

Thao tác (a): N = N-1 = 17-1 = 16

Thao tác (b): 16 = max(4,4) = 4

Thao tác (b): 4 = max(2,2) = 2

Thao tác (a): 2 = 2-1 = 1

Input:

- Dòng đầu tiên ghi lại số tự nhiên T là số lượng Test;
- T dòng kế tiếp mỗi dòng ghi lại một bộ Test. Mỗi test là một số N.

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	Output
3 17 50 100	4 5 5

BÀI 6. DI CHUYỂN TRONG MA TRẬN

Cho ma trận A[M][N]. Nhiệm vụ của bạn hãy tìm số bước đi ít nhất dịch chuyển từ vị trí A[1][1] đến vị trí A[M][N]. Biết mỗi bước đi ta chỉ được phép dịch chuyển đến vị trí A[i][j+A[i][j]] hoặc vị trí A[i+A[i][j]][j] bên trong ma trận.

Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm hai phần: phần thứ nhất là hai số M, N; phần thứ hai là các phần tử của ma trận A[][]; các số được viết cách nhau một vài khoảng trống.
- T, M, N, A[i][j] thỏa mãn ràng buộc: $1 \leq T \leq 100$; $1 \leq M, N, A[i][j] \leq 10^3$.

Output:

- Đưa ra kết quả mỗi test theo từng dòng. In ra -1 nếu không tìm được đáp án.

Ví dụ:

Input	Output
1 3 3 2 1 2 1 1 1 1 1 1	2

BÀI 7. DI CHUYỂN TRÁNH VẬT CẢN

Cho một bảng kích thước N x N, trong đó có các ô trống '.' và vật cản 'X'. Các hàng và các cột được đánh số từ 0.

Mỗi bước di chuyển, bạn có thể đi từ ô (x, y) tới ô (u, v) nếu như 2 ô này nằm trên cùng một hàng hoặc một cột, và không có vật cản nào ở giữa.

Cho điểm xuất phát và điểm đích. Bạn hãy tính số bước di chuyển ít nhất?

Input:

- Dòng đầu ghi số bộ test (không quá 10). Mỗi test gồm:
  - Dòng đầu tiên là số nguyên dương N ($1 \leq N \leq 100$).
  - N dòng tiếp theo, mỗi dòng gồm N kí tự mô tả bảng.
  - Cuối cùng là 4 số nguyên a, b, c, d với (a, b) là tọa độ điểm xuất phát, (c, d) là tọa độ đích. Dữ liệu đảm bảo hai vị trí này không phải là ô có vật cản.

Output:

- Với mỗi test, in ra một số nguyên là đáp số của bài toán.

Ví dụ:

Input	**Output**
1 3 .X. .X. ... 0 0 0 2	3

BÀI 8. GIEO MẦM

Trên một giá có kích thước R x C (R hàng, C cột), một số hạt mầm đã được tra vào các ô. Một số hạt mầm được bón thêm chất dinh dưỡng, nên đã nảy mầm sớm thành cây non.

Mỗi ngày, các cây non sẽ lan truyền chất dinh dưỡng của nó cho các mầm ở ô xung quanh (trái, trên, phải, dưới), làm cho các hạt mầm này phát triển thành cây non. Tuy nhiên, có thể có một số hạt mầm được gieo ở vị trí lẻ loi, do không nhận được chất dinh dưỡng nên không thể nảy mầm.

Các bạn hãy xác định xem cần ít nhất bao nhiêu ngày để tất cả các hạt đều mầm?

Input:

- Dòng đầu ghi số bộ test (không quá 10). Mỗi bộ test gồm:
  - Dòng đầu tiên gồm 2 số nguyên R và C ($1 \leq R, C \leq 500$).
  - R dòng tiếp theo, mỗi dòng gồm C số nguyên A[i][j].
  - A[i][j] = 0, ô (i, j) là ô trống.
  - A[i][j] = 1, ô (i, j) là hạt chưa nảy mầm.
  - A[i][j] = 2, ô (i, j) là cây non.

Output:

- Với mỗi test in ra thời gian ngắn nhất để tất cả các hạt đều nảy mầm. Nếu có hạt nào chưa nảy mầm, in ra -1.

Ví dụ:

Input	**Output**
2 3 5 2 1 0 2 1 1 0 1 2 1 1 0 0 2 1 3 5 2 1 0 2 1 0 0 1 2 1 1 0 0 2 1	2 -1

BÀI 9. DI CHUYỂN TRONG KHÔNG GIAN

Cho một hình hộp chữ nhật có kích thước A x B x C, trong đó A là chiều cao, B là chiều rộng và C là chiều dài. Mỗi ô có thể là một ô trống '.' hoặc vật cản '#'.

Mỗi bước, bạn được phép di chuyển sang một ô kề bên cạnh (không được đi chéo). Nhiệm vụ của bạn là tìm đường đi ngắn nhất bắt đầu 'S' tới vị trí kết thúc 'E'.

Input:

- Dòng đầu tiên là số lượng bộ test T ($1 \leq N \leq 50$).
- Mỗi test bắt đầu bởi 3 số nguyên A, B, C (A, B, C ≤ 30).
- Tiếp theo là A khối, mỗi khối gồm B x C kí tự mô tả một lát cắt của hình hộp chữ nhật. Giữa 2 khối có một dấu xuống dòng.

Output:

- In ra một số nguyên là đường đi ngắn nhất từ S tới E. Nếu không di chuyển được, in ra -1.

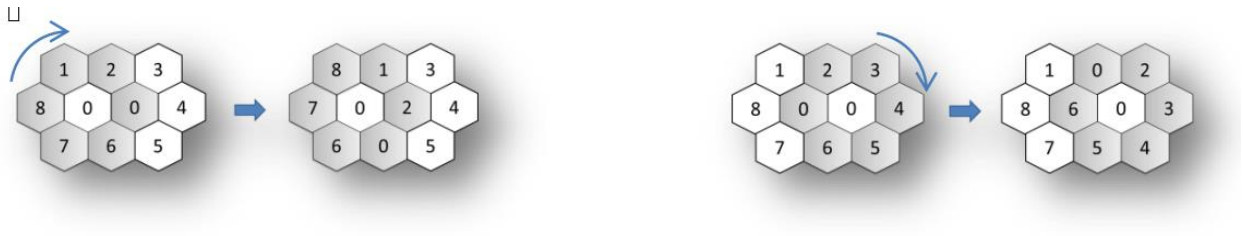
Ví dụ:

Input	**Output**
2 3 4 5 S.... .###. .##.. ###.# ##### ##### ##.## ##... ##### ##### #.### ####E 1 3 3 S##	11 -1

#E# ### 	

BÀI 10. HEXGAME

HEXGAME là một trò chơi xếp hình gồm 10 miếng ghép hình lục giác đều, trên mỗi miếng ghép được điền một số nguyên, có 8 miếng được điền số từ 1 đến 8 và có hai miếng điền số 0. Các miếng liên kết với nhau tạo thành lưới tổ ong. Ban đầu các miếng ghép ở vị trí như hình vẽ. Tại mỗi bước, chọn một miếng ghép có đúng 6 miếng ghép kề cạnh làm tâm, rồi xoay một nấc 6 miếng ghép kề cạnh đó theo chiều kim đồng hồ. Như vậy chỉ có hai cách chọn tâm, đó là chọn tâm bên trái và chọn tâm bên phải.



Yêu cầu: Cho một trạng thái của trò chơi (nhận được sau một dãy biến đổi từ trạng thái ban đầu), hãy tính số phép biến đổi ít nhất để đưa về trạng thái ban đầu.

Input:

- Dòng đầu ghi số bộ test (không quá 10). Mỗi bộ test gồm:
  - Dòng đầu tiên chứa 3 số ở 3 miếng ghép dòng thứ nhất (thứ tự từ trái qua phải).
  - Dòng thứ hai chứa 4 số ở 4 miếng ghép dòng thứ hai (thứ tự từ trái qua phải).
  - Dòng thứ 3 chứa 3 số ở 3 miếng ghép dòng thứ ba (thứ tự từ trái qua phải).

Output:

- Với mỗi bộ test in ra một số nguyên là số phép biến đổi ít nhất để đưa được về trạng thái ban đầu.

Ví dụ:

Input	Output
1	5

1 0 2 8 6 0 3 7 5 4	

5.3. Bài tập về Cây nhị phân

BÀI 1. DUYỆT CÂY 1

Cho phép duyệt cây nhị phân Inorder và Preorder, hãy đưa ra kết quả phép duyệt Postorder của cây nhị phân. Ví dụ với cây nhị phân có các phép duyệt cây nhị phân của cây dưới đây:

Inorder : 4 2 5 1 3 6
Preorder: : 1 2 4 5 3 6
Postorder : 4 5 2 6 3 1

Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 3 dòng: dòng đầu tiên đưa vào số N là số lượng node; dòng tiếp theo đưa vào N số theo phép duyệt Inorder; dòng cuối cùng đưa vào N số là kết quả của phép duyệt Preorder; các số được viết cách nhau một vài khoảng trống.
- T, N, node thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 1000$; $1 \le$ giá trị node $\le 10^4$;

Output:

- Đưa ra kết quả phép duyệt Postorder theo từng dòng.

Ví dụ:

Input	Output
1 6 4 2 5 1 3 6 1 2 4 5 3 6	4 5 2 6 3 1

BÀI 2. DUYỆT CÂY 2

Cho mảng A[] gồm N node là biểu diễn phép duyệt theo thứ tự giữa (Preorder) của cây nhị phân tìm kiếm. Nhiệm vụ của bạn là đưa ra phép duyệt theo thứ tự sau của cây nhị phân tìm kiếm.

Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào số N là số lượng node; dòng tiếp theo đưa vào N số A[i]; các số được viết cách nhau một vài khoảng trống.
- T, N, node thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 10^3$; $1 \le A[i] \le 10^4$;

Output:

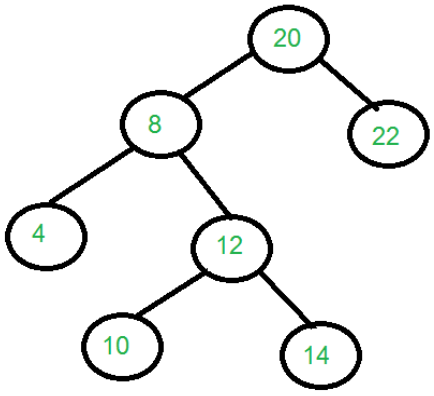
- Đưa ra kết quả phép duyệt Postorder theo từng dòng.

Ví dụ:

Input	Output
2 5 40 30 35 80 100 8 40 30 32 35 80 90 100 120	35 30 100 80 40 35 32 30 120 100 90 80 40

BÀI 3. DUYỆT CÂY 3

Cho cây nhị phân, nhiệm vụ của bạn là duyệt cây theo Level-order. Phép duyệt level-order trên cây là phép thăm node theo từng mức của cây. Ví dụ với cây dưới đây sẽ cho ta kết quả của phép duyệt level-order: 20 8 22 4 12 10 14.



Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào số N là số lượng cạnh của cây; dòng tiếp theo đưa vào N bộ ba (u, v, x), trong đó u là node cha, v là node con, x= R nếu v là con phải, x=L nếu v là con trái; u, v, x được viết cách nhau một vài khoảng trống.
- T, N, u, v, thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 10^3$; $1 \le u, v \le 10^4$;

Output:

- Đưa ra kết quả phép duyệt level-order theo từng dòng.

Ví dụ:

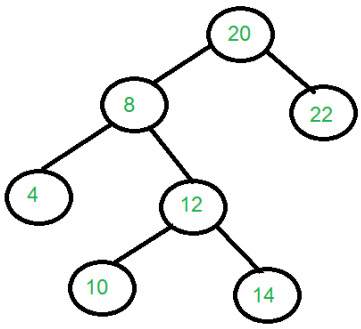
Input	Output
2 2 1 2 R 1 3 L 4 10 20 L 10 30 R 20 40 L 20 60 R	1 3 2 10 0 30 40 60

BÀI 4. DUYỆT CÂY 4

Cho hai mảng là phép duyệt Inorder và Level-order, nhiệm vụ của bạn là xây dựng cây nhị phân và đưa ra kết quả phép duyệt Postorder. Level-order là phép duyệt theo từng mức của cây. Ví dụ như cây dưới đây ta có phép Inorder và Level-order như dưới đây:

Inorder : 4 8 10 12 14 20 22

Level order: 20 8 22 4 12 10 14



Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 3 dòng: dòng đầu tiên đưa vào số N là số lượng node; dòng tiếp theo đưa vào N số là phép duyệt Inorder; dòng cuối cùng đưa vào N số là phép duyệt Level-order; các số được viết cách nhau một vài khoảng trống.
- T, N, node thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 10^3$; $1 \le A[i] \le 10^4$;

Output:

- Đưa ra kết quả phép duyệt Postorder theo từng dòng.

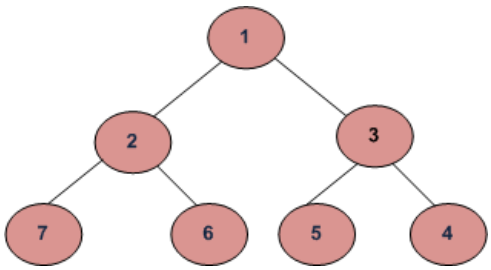
Ví dụ:

Input	Output
2 3 1 0 2	1 2 0 3 4 1 5 6 2 0

0 1 2 7 3 1 4 0 5 2 6 0 1 2 3 4 5 6	

BÀI 5. DUYỆT CÂY 5

Cho cây nhị phân, nhiệm vụ của bạn là duyệt cây theo xoắn ốc (spiral-order). Phép. Ví dụ với cây dưới đây sẽ cho ta kết quả của phép duyệt spiral-order: 1 2 3 4 5 6 7.



Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào số N là số lượng cạnh của cây; dòng tiếp theo đưa vào N bộ ba (u, v, x), trong đó u là node cha, v là node con, x= R nếu v là con phải, x=L nếu v là con trái; u, v, x được viết cách nhau một vài khoảng trống.
- T, N, u, v, thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 10^3$; $1 \le u, v \le 10^4$;

Output:

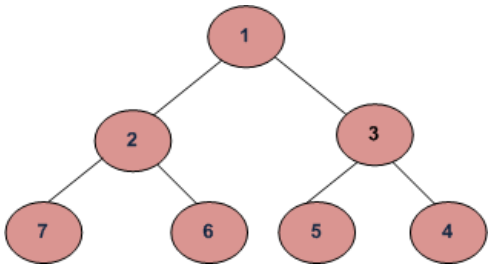
- Đưa ra kết quả phép duyệt level-order theo từng dòng.

Ví dụ:

Input	**Output**
2 2 1 2 R 1 3 L 4 10 20 L 10 30 R 20 40 L 20 60 R	1 3 2 10 0 30 60 40

BÀI 6. DUYỆT CÂY 6

Cho cây nhị phân, nhiệm vụ của bạn là duyệt cây theo mức đảo ngược (revese-level-order). Với cây dưới đây sẽ cho ta kết quả của phép duyệt theo mức đảo ngược là : 7 6 5 4 3 2 1.



Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào số N là số lượng cạnh của cây; dòng tiếp theo đưa vào N bộ ba (u, v, x), trong đó u là node cha, v là node con, x= R nếu v là con phải, x=L nếu v là con trái; u, v, x được viết cách nhau một vài khoảng trống.
- T, N, u, v, thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 10^3$; $1 \le u, v \le 10^4$;

Output:

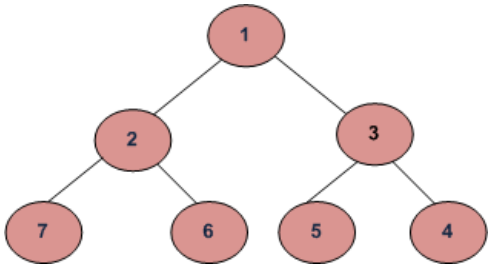
- Đưa ra kết quả phép duyệt revese-level-order theo từng dòng.

Ví dụ:

Input	**Output**
2 2 1 2 R 1 3 L 4 10 20 L 10 30 R 20 40 L 20 60 R	3 2 1 40 20 30 10

BÀI 7. KIỂM TRA NODE LÁ

Cho cây nhị phân, nhiệm vụ của bạn là kiểm tra xem tất cả các node lá của cây có cùng một mức hay không? Ví dụ với cây dưới đây sẽ cho ta kết quả là Yes.



Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào số N là số lượng cạnh của cây; dòng tiếp theo đưa vào N bộ ba (u, v, x), trong đó u là node cha, v là node con, x= R nếu v là con phải, x=L nếu v là con trái; u, v, x được viết cách nhau một vài khoảng trống.
- T, N, u, v, thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 10^3$; $1 \le u, v \le 10^4$;

Output:

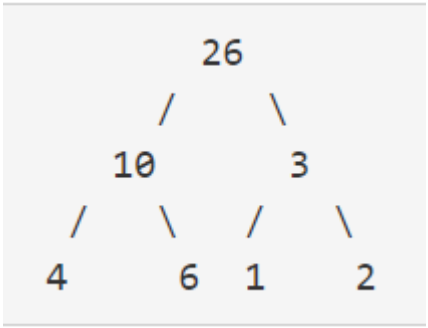
- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	Output
2 2 1 2 R 1 3 L 4 10 20 L 10 30 R 20 40 L 20 60 R	1 0

BÀI 8. CÂY NHỊ PHÂN TỔNG

Cho cây nhị phân, nhiệm vụ của bạn là kiểm tra xem cây nhị phân có phải là một cây tổng hay không? Một cây nhị phân được gọi là cây tổng nếu tổng các node con của node trung gian bằng giá trị node cha. Node không có node con trái hoặc phải được hiểu là có giá trị 0. Ví dụ dưới đây là một cây tổng



Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào số N là số lượng cạnh của cây; dòng tiếp theo đưa vào N bộ ba (u, v, x), trong đó u là node cha, v là node con, x= R nếu v là con phải, x=L nếu v là con trái; u, v, x được viết cách nhau một vài khoảng trống.
- T, N, u, v, thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 10^3$; $1 \le u, v \le 10^4$;

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	**Output**
2 2 3 1 L 3 2 R 4 10 20 L 10 30 R 20 40 L 20 60 R	1 0

BÀI 9. CÂY NHỊ PHÂN HOÀN HẢO

Cho cây nhị phân, nhiệm vụ của bạn là kiểm tra xem cây nhị phân có phải là một cây hoàn hảo hay không (perfect tree)? Một cây nhị phân được gọi là cây hoàn hảo nếu tất cả các node trung gian của nó đều có hai node con và tất cả các node lá đều có cùng một mức.

Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào số N là số lượng cạnh của cây; dòng tiếp theo đưa vào N bộ ba (u, v, x), trong đó u là node cha, v là node con, x= R nếu v là con phải, x=L nếu v là con trái; u, v, x được viết cách nhau một vài khoảng trống.
- T, N, u, v, thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 10^3$; $1 \le u, v \le 10^4$;

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	Output
3 6 10 20 L 10 30 R 20 40 L 20 50 R 30 60 L 30 70 R 2 18 15 L 18 30 R 5 1 2 L 2 4 R 1 3 R 3 5 L 3 6 R	Yes Yes No

BÀI 10. CÂY NHỊ PHÂN ĐỦ

Cho cây nhị phân, nhiệm vụ của bạn là kiểm tra xem cây nhị phân có phải là một cây đủ hay không (full binary tree)? Một cây nhị phân được gọi là cây đủ nếu tất cả các node trung gian của nó đều có hai node con.

Input:

- Dòng đầu tiên đưa vào số lượng test T.
- Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào số N là số lượng cạnh của cây; dòng tiếp theo đưa vào N bộ ba (u, v, x), trong đó u là node cha, v là node con, x= R nếu v là con phải, x=L nếu v là con trái; u, v, x được viết cách nhau một vài khoảng trống.
- T, N, u, v, thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N \le 10^3$; $1 \le u, v \le 10^4$;

Output:

- Đưa ra kết quả mỗi test theo từng dòng.

Ví dụ:

Input	Output
2 4 1 2 L 1 3 R 2 4 L 2 5 R 3 1 2 L 1 3 R 2 4 L	1 0

TÀI LIỆU THAM KHẢO

[1] Nguyễn Duy Phương, Bài giảng Toán rời rạc 1 và Toán rời rạc 2, Học viện Công nghệ Bưu chính Viễn thông, 2015.

[2] Nguyễn Duy Phương, Nguyễn Mạnh Sơn, Bài tập Kỹ thuật lập trình hướng đối tượng, Học viện Công nghệ Bưu chính Viễn thông, 2020.

[2] Nguyễn Duy Phương, Nguyễn Mạnh Sơn, Bài giảng Cấu trúc dữ liệu và Giải thuật, Học viện Công nghệ Bưu chính Viễn thông, 2020.

[3] Nguyễn Mạnh Sơn, Bài giảng Lập trình hướng đối tượng, Học viện Công nghệ Bưu chính Viễn thông, 2020.

[4] Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser. Data Structures & Algorithms in Java. 6th edition, Wiley, 2014.

[5] https://www.geeksforgeeks.org/

[6] https://codeforces.com/