



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



Thuật toán ứng dụng

Nguyễn Khánh Phương

Computer Science department
School of Information and Communication technology
E-mail: phuongnk@soict.hust.edu.vn

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

Nội dung khóa học

Chương 1. Các cấu trúc dữ liệu và thư viện

Chương 2. Kỹ thuật đệ quy và nhánh cận

Chương 3. Chia để trị

Chương 4. Quy hoạch động

Chương 5. Các thuật toán trên đồ thị và ứng dụng

Chương 6. Các thuật toán xử lý xâu và ứng dụng

Chương 7. Lớp bài toán NP-đầy đủ

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

Nội dung chương 1

1. Các kiểu dữ liệu cơ bản
2. Xử lý số nguyên lớn
3. Thư viện cấu trúc dữ liệu và thuật toán
4. Biểu diễn tập hợp bằng Bitmask
5. Biểu diễn đồ thị
6. Cấu trúc dữ liệu mở

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

Nội dung chương 1

- 1. Các kiểu dữ liệu cơ bản**
2. Xử lý số nguyên lớn
3. Thư viện cấu trúc dữ liệu và thuật toán
4. Biểu diễn tập hợp bằng Bitmask
5. Biểu diễn đồ thị
6. Cấu trúc dữ liệu mở

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

1. Các kiểu dữ liệu cơ bản

- Các kiểu dữ liệu phải biết
 - `bool`: biến bool (true/false)
 - `char`: biến nguyên 8 bit (thường được sử dụng để biểu diễn các ký tự ASCII)
 - `string`: biến xâu ký tự
 - `int`: biến nguyên 32 bit
 - `long`: biến nguyên 32 bit
 - `float`: biến thực 32 bit
 - `double`: biến thực 64 bit
- Các modifier
 - `signed` – `unsigned char, signed char, unsigned int, signed int, unsigned long, signed long`
 - `unsigned` – `short int, unsigned short int,`
 - `short` – `long int, unsigned long int, long double`
 - `long` – `...`

5

1. Các kiểu dữ liệu cơ bản

DATA TYPE	SIZE (IN BYTES)	RANGE
<code>short int</code>	2	-32,768 to 32,767
<code>unsigned short int</code>	2	0 to 65,535
<code>unsigned int</code>	4	0 to 4,294,967,295
<code>int</code>	4	-2,147,483,648 to 2,147,483,647
<code>long int</code>	4	-2,147,483,648 to 2,147,483,647
<code>unsigned long int</code>	4	0 to 4,294,967,295
<code>long long int</code>	8	-(2^{63}) to (2^{63})-1
<code>unsigned long long int</code>	8	0 to 18,446,744,073,709,551,615
<code>signed char</code>	1	-128 to 127
<code>unsigned char</code>	1	0 to 255
<code>float</code>	4	$\approx -3.4 \times 10^{-38}$ to $\approx 3.4 \times 10^{-38}$ ≈ 7 chữ số
<code>double</code>	8	$\approx -1.7 \times 10^{-308}$ to $\approx 1.7 \times 10^{-308}$ ≈ 14 chữ số
<code>long double</code>	12	

Trên trình biên dịch GCC 64 bit

6

Nội dung chương 1

1. Các kiểu dữ liệu cơ bản
- 2. Xử lý số nguyên lớn**
3. Thư viện cấu trúc dữ liệu và thuật toán
4. Biểu diễn tập hợp bằng Bitmask
5. Biểu diễn đồ thị
6. Cấu trúc dữ liệu mở

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

2. Xử lý số nguyên lớn

- Làm thế nào để thực hiện tính toán trên các số nguyên lớn không thể lưu trữ bằng kiểu long long int ?
- Ý tưởng:
 - lưu trữ số nguyên dưới dạng string
 - Thực hiện phép toán trên từng chữ số từ phải sang trái, có lưu phần nhớ như đã học ở tiểu học

NGUYỄN KHÁNH PHƯƠNG 8
KHMT – SOICT-ĐHBK HN

Ví dụ 1: cộng hai số nguyên lớn

A1. ADD

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Cho hai số nguyên a và b . Tính tổng $a + b$

Input

Gồm hai dòng, mỗi dòng ghi một số nguyên

$$2^{10} \approx 10^3$$

$$\Rightarrow 2^{63} \approx 9 \times 10^{18} \sim \text{long long int}$$

Output

Một dòng chứa số nguyên là kết quả bài toán

Scoring

$0 \leq a, b \leq 9 \times 10^{18}$ Có 50% test với $a, b \leq 10^9$

Examples**input**

1

1

[Copy]

output

2

[Copy]

NGUYỄN KHÁNH PHƯƠNG 9
KHMT – SOICT-ĐHBK HN

Ví dụ 1: cộng hai số nguyên lớn

$$\begin{array}{r} + 12345678919 \\ 00000012345 \end{array} \quad \begin{array}{l} \text{string } a \\ \text{string } b \end{array}$$

string Sum(string a, string b)

- Bước 1: vì mỗi số nguyên này đều được biểu diễn bởi 1 string, nên ta sẽ làm cho 2 xâu này có độ dài bằng nhau bằng cách chèn thêm kí tự “0” vào trước xâu có độ dài nhỏ hơn.

```
string Sum(string a, string b){  
    while (a.length() < b.length())  
        a = "0" + a;  
  
    while (b.length() < a.length())  
        b = "0" + b;
```

NGUYỄN KHÁNH PHƯƠNG 10
KHMT – SOICT-ĐHBK HN

Ví dụ 1: cộng hai số nguyên lớn

$$\begin{array}{r} + \quad 12345678919 \\ 00000012345 \end{array}$$

string a
string b

- Bước 2: thực hiện phép cộng như học ở lớp 1: duyệt từng ký tự của 2 xâu theo thứ tự từ phải sang trái, **cộng hai ký tự** tương ứng với nhau và lưu ý phần “nhó” nếu có

$$\begin{array}{r} a[i] + b[i] ??? \\ (a[i] - '0') + (b[i] - '0') \\ \hline 48 & 0 \\ 49 & 1 \\ 50 & 2 \\ 51 & 3 \\ 52 & 4 \\ 53 & 5 \\ 54 & 6 \\ 55 & 7 \\ 56 & 8 \\ 57 & 9 \end{array}$$
 11

```
string kq;
int tong, nho = 0;
int sokitu = a.length();
for (int i=sokitu - 1; i>=0; i--)
{
    tong = (a[i] - '0') + (b[i] - '0') + nho;
    nho = tong/10;
    kq = char(tong%10 + '0') + kq;
}
if (nho == 1) kq = "1" + kq;
```

Ví dụ 1: Source code cộng hai số nguyên lớn

```
#include <iostream>
using namespace std;
string Sum(string a, string b){
    while (a.length() < b.length())    a = '0' + a;
    while (b.length() < a.length())    b = '0' + b;

    string kq;
    int tong, nho = 0;
    int sokitu = a.length();
    for (int i=sokitu - 1; i>=0; i--)
    {
        tong = (a[i] - '0') + (b[i] - '0') + nho; //tong = (a[i] - 48) + (b[i] - 48) + nho;
        nho = tong/10;
        kq = char(tong%10 + '0') + kq; //kq = char(tong%10 + 48) + kq;
    }
    if (nho == 1) kq = "1" + kq;
    return kq;
}
int main(){
    string a, b;
    cin >> a >> b;
    cout << Sum(a, b);
}
```

Ví dụ 1: Source code cộng hai số nguyên lớn

```
#include<bits/stdc++.h>
using namespace std;

string Sum(string a, string b){
    while (a.length() < b.length())    a = '0' + a;
    while (b.length() < a.length())    b = '0' + b;

    string kq;
    int tong, nho = 0;
    int sokitu = a.length();
    for (int i=sokitu - 1; i>=0; i--)
    {
        tong = (a[i] - '0') + (b[i] - '0') + nho; //tong = (a[i] - 48) + (b[i] - 48) + nho;
        nho = tong/10;
        kq.push_back(tong%10 + '0');
    }
    if (nho == 1) kq.push_back(nho+'0');
    reverse(kq.begin(), kq.end());
    return kq;
}

int main(){
    string a, b;
    cin >> a >> b;
    cout << Sum(a, b);
}
```

NGUYỄN KHÁNH PHƯƠNG 13
KHMT – SOICT-DHBK HN

Ví dụ 1: cộng hai số nguyên lớn

$$\begin{array}{r} + 12345678911 \\ 00000012345 \end{array} \quad \begin{array}{l} \text{string } a \\ \text{string } b \end{array}$$

string Sum(string a, string b)

- Cách 2: Thay vì điền “0” để hai xâu có độ dài bằng nhau như đã làm ở slide trước, ta sẽ đảo ngược hai xâu

$$\begin{array}{r} 1198765321 \\ 54321 \end{array}$$

sau đó, như ở ví dụ này, ta sẽ chỉ tiến hành cộng 5 ký tự phải nhất của xâu a với xâu b theo thứ tự từ phải sang trái, được kết quả bao nhiêu, ta ghép nối với các ký tự còn lại của xâu a (chú ý: vẫn phải cộng thêm “nhó” nếu có). Đảo ngược xâu kết quả thu được, ta sẽ có được lời giải của bài toán

14

Ví dụ 1: Source code cộng hai số nguyên lớn: cách 2

```
#include<bits/stdc++.h>
using namespace std;

string Sum(string a, string b)
{
    //Xau a luon co do dai lon hon xau b
    if (a.length() < b.length()) swap(a, b);
    string kq = "";
    int n1 = a.length(), n2 = b.length();
    //Dao nguoc hai xau:
    reverse(a.begin(), a.end()); reverse(b.begin(), b.end());
    int nho = 0;
    for (int i=0; i<n2; i++)
    {
        int sum = ((a[i]-'0')+(b[i]-'0')+ nho);
        kq.push_back(sum%10 + '0');
        nho = sum/10;
    }
    //Them cac chu so con lai cua so lon hon vao ket qua
    for (int i=n2; i<n1; i++)
    {
        int sum = ((a[i]-'0')+nho);
        kq.push_back(sum%10 + '0');
        nho = sum/10;
    }
    //Cong nho neu co:
    if (nho) kq.push_back(nho+'0');
    //dao nguoc xau ket qua:
    reverse(kq.begin(), kq.end());
    return kq;
}
```

NGUYỄN KHÁNH PHƯƠNG 15
KHMT – SOICT - ĐHBK HN

Ví dụ 2: Bài toán Integer Inquiry

Input

The input will consist of at most 100 lines of text, each of which contains a single VeryLongInteger. Each VeryLongInteger will be 100 or fewer characters in length, and will only contain digits (no VeryLongInteger will be negative).

The final input line will contain a single zero on a line by itself.

Output

Your program should output the sum of the VeryLongIntegers given in the input.

Sample Input

123456789012345678901234567890
123456789012345678901234567890
123456789012345678901234567890
0

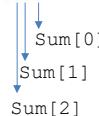
Có nhiều nhất là 100 dòng
Mỗi dòng là một số nguyên dương có ≤ 100 chữ số
Dòng cuối cùng chứa chữ số 0 để đánh dấu sự kết thúc

Tổng của 100 số, mỗi số có 100 chữ số, là số nguyên
có bao nhiêu chữ số ????

Sample Output

370370367037037036703703703670

Tổng là số nguyên chứa tối đa $100 \times 2 = 200$ chữ số



int Sum[201]={ 0 };

NGUYỄN KHÁNH PHƯƠNG 16
KHMT – SOICT - ĐHBK HN

Ví dụ 2: Bài toán Integer Inquiry

```
123456789012345678901234567890  
123456789012345678901234567890  
123456789012345678901234567890  
0
```

```
Sum[0] += 0 ➔ Sum[0] = 0  
Sum[1] += 9 ➔ Sum[1] = 9  
Sum[2] += 8 ➔ Sum[2] = 8  
.....
```

NGUYỄN KHÁNH PHƯƠNG 17
KHMT – SOICT-ĐHBK HN

Ví dụ 2: Bài toán Integer Inquiry

```
123456789012345678901234567890  
123456789012345678901234567890  
123456789012345678901234567890  
0
```

Sum[0] += 0 ➔ Sum[0] = 0	Sum[0] += 0 ➔ Sum[0] = 0
Sum[1] += 9 ➔ Sum[1] = 9	Sum[1] += 9 ➔ Sum[1] = 9+9=18
Sum[2] += 8 ➔ Sum[2] = 8	Sum[2] += 8 ➔ Sum[2] = 8+8=16
.....

NGUYỄN KHÁNH PHƯƠNG 18
KHMT – SOICT-ĐHBK HN

Ví dụ 2: Bài toán Integer Inquiry

123456789012345678901234567890

123456789012345678901234567890

123456789012345678901234567890

0

Sum[0] += 0 → Sum[0] = 0

Sum[1] += 9 → Sum[1] = 9

Sum[2] += 8 → Sum[2] = 8

.....

Sum[0] += 0 → Sum[0] = 0

Sum[1] += 9 → Sum[1] = 9+9=18

Sum[2] += 8 → Sum[2] = 8+8=16

.....

Sum[0] += 0 → Sum[0] = 0

Sum[1] += 9 → Sum[1] = 18+9=27

Sum[2] += 8 → Sum[2] = 16+8=24

.....

NGUYỄN KHÁNH PHƯƠNG 19
KHMT – SOICT - ĐHBK HN

Ví dụ 2: Bài toán Integer Inquiry

123456789012345678901234567890

123456789012345678901234567890

123456789012345678901234567890

0

#include<bits/stdc++.h>

using namespace std;

int main() {

char s[201];

int Sum[201] = {0}, i, j, length;

while(gets(s)) {

if(!strcmp(s, "0")) break;

length = strlen(s);

for(i = 0, j = length-1; i < length; i++, j--)

Sum[i] += s[j] - '0';

}

//Điền xử lý “nhớ” vào đây

return 0;

Sum[0] += 0 → Sum[0] = 0

Sum[1] += 9 → Sum[1] = 18+9=27

Sum[2] += 8 → Sum[2] = 16+8=24

.....

Xử lý “nhớ”

Sum[0] = 0

Sum[1] = 7; nhớ 2 → cộng vào Sum[2]

Sum[2] = 4 + 2 (nhớ từ Sum[1]) = 6; nhớ 2

→ Cộng vào Sum[3]

.....

NGUYỄN KHÁNH PHƯƠNG 20
KHMT – SOICT - ĐHBK HN

Ví dụ 2: Bài toán Integer Inquiry

```

#include<bits/stdc++.h>
using namespace std;

int main() {
    char s[201];
    int Sum[201] = {0}, i, j, length;
    while(gets(s)) {
        if(!strcmp(s, "0")) break;
        length = strlen(s);
        for(i = 0, j = length-1; i < length; i++, j--)
            Sum[i] += s[j] - '0';
    }
    //Xử lý nhó:
    for(i = 0; i < 200; i++)
        if(Sum[i] >= 10) {
            Sum[i+1] += Sum[i]/10;
            Sum[i] %= 10;
        }
    //In kết quả phép cộng: for(i = 200; i >= 0; i--) ??? In ra màn hình số nào
    //                                cout<<Sum[i];
}

return 0;
}

```

Sum[0] += 0 ➔ Sum[0] = 0
 Sum[1] += 9 ➔ Sum[1] = 18 + 9 = 27
 Sum[2] += 8 ➔ Sum[2] = 16 + 8 = 24

 Xử lý “nhó”
 Sum[0] = 0
 Sum[1] = 7; nhó 2 ➔ cộng vào Sum[2]
 Sum[2] = 4 + 2 (nhó từ Sum[1]) = 6; nhó 2
 ➔ Cộng vào Sum[3]

NGUYỄN KHÁNH PHƯƠNG 21
 HỌC TẬP CÁC BÀI TẬP LẬP TRÌNH C

NGUYỄN KHÁNH PHƯƠNG 21
KHMT – SOICT- ĐHBK HN

Ví dụ 2: Bài toán Integer Inquiry

??? Làm thế nào để loại bỏ các chữ số “0” không cần thiết ra khỏi kết quả

```
//In kết quả phép cộng: for(i = 200; i >= 0; i--) ??? In ra màn hình số nào  
cout<<Sum[i];
```

NGUYỄN KHÁNH PHƯƠNG 22
KHMT – SOICT-ĐHBK HN

Ví dụ 2: Bài toán Integer Inquiry

```
#include<bits/stdc++.h>
using namespace std;

int main() {
    char s[201];
    int Sum[201] = {0}, i, j, length;
    while(gets(s)) {
        if(strcmp(s, "0")) break;
        length = strlen(s);
        for(i = 0, j = length-1; i < length; i++, j--)
            Sum[i] += s[j] - '0';
    }
    //Xử lý nhö:
    for(i = 0; i < 200; i++)
        if(Sum[i] >= 10) {
            Sum[i+1] += Sum[i]/10;
            Sum[i] %= 10;
        }

    //In ket qua:
    i = 200;
    while(Sum[i] == 0 && i >= 0)    i--;
    if(i == -1) printf("0");
    else
        for(; i >= 0; i--) printf("%d", Sum[i]);
    return 0;
}
```

NGUYỄN KHÁNH PHƯƠNG 23
KHMT – SOICT-ĐHBKHN

Nội dung chương 1

1. Các kiểu dữ liệu cơ bản
2. Xử lý số nguyên lớn
- 3. Thư viện cấu trúc dữ liệu và thuật toán**
4. Biểu diễn tập hợp bằng Bitmask
5. Biểu diễn đồ thị
6. Cấu trúc dữ liệu mở

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBKHN

3. Thư viện cấu trúc dữ liệu và thuật toán

Thư viện mẫu chuẩn STL (Standard Template Library) trong C++ chia làm 4 thành phần:

- Containers Library : chứa các cấu trúc dữ liệu mẫu (template)
 - Sequence containers (cấu trúc tuần tự): Vector, Deque, List
 - Containers adaptors: Stack, Queue, Priority_queue
 - Associative containers (cấu trúc liên kết): Set, Multiset, Map, Multimap, Bitset
- Algorithms Library: một số thuật toán để thao tác trên dữ liệu
- Iterator Library: giống như con trỏ, dùng để truy cập đến các phần tử dữ liệu của container.
- Numeric library
- Để sử dụng STL, ta cần khai báo từ khóa “using namespace std;” sau các khai báo thư viện (các “#include”, hay “#define”,...)

25

3. Thư viện cấu trúc dữ liệu và thuật toán

Một số cấu trúc dữ liệu thông dụng:

- Mảng tĩnh `int arr[10]`
- Mảng động `vector<int>`
- Danh sách liên kết `list <int>`
- Ngăn xếp `stack <int>`
- Hàng đợi `queue <int>`
- Hàng đợi có ưu tiên `priority_queue <int>`
- Hàng đợi hai đầu `deque <int>`
- Tập hợp `set <int>`
- Ánh xạ `map <int, int>`

26

3.1. Thư viện STL trong C++: Vector

#include <vector> Vector có thể xem như là một mảng động (dynamic array): một mảng có thể thay đổi kích thước.

Vector 1 chiều:

```
vector <int> vec1; //khai báo vector 1 chiều, rỗng, có kiểu dữ liệu là int
vector <int> vec2 (4, 100); //tạo vector có 4 phần tử int, khởi tạo giá trị cho cả 4
phần tử này là 100
vector <int> vec3 (vec2.begin(), vec2.end()); //khai báo vector kiểu int có
tên là vec3 sao chép từ đầu đến cuối vector vec2
vector <int> vec4(vec2); //tạo vector vec4 kiểu int và sao chép tất cả phần tử của
vec2; cách khai báo này cho kết quả giống như khai báo vec3
```

Vector 2 chiều:

```
vector <vector <int> > v; //khai báo vector 2 chiều, rỗng, có kiểu dữ liệu là int
vector <vector <int> > v(5,10); //khai báo vector 2 chiều kích thước 5*10
vector <vector <int> > v(5); khai báo vector 2 chiều, có 5 vector 1 chiều đều rỗng
vector <vector <int> > v(5, vector <int> (10,1)); //khai báo vector 5x10
với các phần tử khởi tạo giá trị đều là 1
```

27

3.1. Thư viện STL trong C++: Vector

```
vector <int> v;
v.size() : trả về số lượng phần tử
v.empty() : true/false

v[i] : phần tử thứ i trong vector v
v.at(i) : phần tử thứ i trong vector v
v.front() : phần tử đầu tiên trong vector v
v.back() : phần tử cuối cùng trong vector v

v.push_back(x) : thêm phần tử có giá trị x vào cuối vector v
v.pop_back() : loại bỏ phần tử cuối cùng ra khỏi vector v
```

NGUYỄN KHÁNH PHƯƠNG 28
KHMT – SOICT - ĐHBKHN

3.1. Thư viện STL trong C++: Vector

v.insert(iterator position, const x) : Chèn phần tử có giá trị x vào **trước** vị trí position.

v.insert(iterator position, int n, const x) : Chèn n phần tử có giá trị x vào **trước** vị trí position.

v.insert(iterator position, iterator a, iterator b) : Chèn vào **trước** vị trí position của vector hiện hành tất cả các phần tử trong khoảng [a, b) của một vector khác, tức là từ phần tử thứ a đến phần tử thứ (b-1).

```
#include <iostream>
#include <vector>
using namespace std ;
int main ()
{
    vector<int> myVector (5,100); // tao vector 5 phan tu: 100 100 100 100 100
    vector<int>::iterator it; //khai bao iterator
    it = myVector.begin();
    // them 200 vao vi tri thu 2:
    myVector.insert(it+1,200) ; // 100 200 100 100 100
    //them 2 gia tri 5 vao dau vector
    myVector.insert(myVector.begin(),2,5) ; //5 5 100 200 100 100 100
    //print vector
    for(int i = 0 ; i < myVector.size() ; i++) cout<<myVector[i]<<" " ;
    return 0;
}
```

9

3.1. Thư viện STL trong C++: Vector

v.insert(iterator position, iterator a, iterator b) : Chèn vào **trước** vị trí position của vector hiện hành tất cả các phần tử trong khoảng [a, b) của một vector khác.

Ngoài việc chèn vector này vào vector khác, ta có thể chèn một mảng các phần tử vào vector:

```
#include <iostream>
#include <vector>
using namespace std ;

int main ()
{
    // create vector
    vector<int> vec1 (5,100); // 100 100 100 100 100
    vector<int> vec2(3,2) ; // 2 2 2
    //Chen toan bo vec2 vao truoc phan tu thu 2 cua vec1:
    vec1.insert(vec1.begin()+1, vec2.begin(), vec2.end()) ; // 100 2 2 2 100 100 100

    //create array
    int myArray[3] = {3,4,5} ; // 3 4 5
    //Chen hai phan tu dau tien cua myarray vao truoc phan tu thu nhat cua vector vec1:
    vec1.insert(vec1.begin(), myArray, myArray +2 ) ; // 3 4 100 2 2 2 100 100 100

    //print vector
    for(int i = 0 ; i < vec1.size() ; i++) cout<<vec1[i]<<" " ;

    return 0;
}
```

NGUYỄN KHÁNH PHƯƠNG 30
KHMT – SOICT - ĐHBKHN

3.1. Thư viện STL trong C++: Vector

- `v.erase(iterator position)`: Xóa phần tử ở vị trí position
- `v.erase(iterator first, iterator last)`: Xóa tất cả các phần tử trong nửa khoảng [frist, last], có nghĩa là từ phần tử thứ first đến phần tử thứ (last-1).
- `v.swap(vector v1)`: hoán đổi vector v1 với vector hiện hành v
- `v.clear()`: xóa toàn bộ vector v

NGUYỄN KHÁNH PHƯƠNG 31
KHMT – SOICT-ĐHBK HN

3.2. Thư viện STL trong C++: iterator

- Iterator được hiểu đơn giản là con trỏ dùng để truy xuất đến các container trong thư viện STL. Vì vậy cách dùng iterator còn phụ thuộc vào container.
- Iterator có hai loại: iterator “thuận chiều” (iterator) và iterator “ngược chiều” (reverse_iterator).

Ví dụ iterator của container vector:

- Khai báo iterator có tên là “it” dùng để truy xuất vector:

```
vector <int> :: iterator it;
```

- Khai báo reverse_iterator có tên là “rit” dùng để truy xuất vector:

```
vector <int> :: reverse_iterator rit;
```

NGUYỄN KHÁNH PHƯƠNG 32
KHMT – SOICT-ĐHBK HN

```
#include <iostream>
#include <vector>
using namespace std;
```

```
int main()
{
    vector<int>:: iterator it;

    vector<int> myVector ;
    for(int i = 0 ; i <10; i++) myVector.push_back(i) ;

    it = myVector.begin() ;
    cout<<"it is begin, We have value: " << *it<<endl ;

    it = myVector.end() ;
    cout<<"it is end, We have value: " << *it<<endl ;

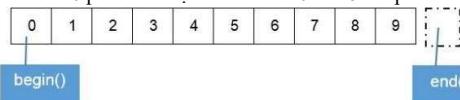
    myVector[10] = 10 ;
    cout<<"it is end, We have value: " << *it<<endl ;
    return 0;
}
```

Lưu ý:

iterator là một con trỏ nên ta phải dùng toán tử "*" phía trước tên biến `it` để truy xuất giá trị con trỏ đang trỏ đến.

`it=myVector.begin();` //trỏ đến vị trí đầu tiên của vector
`it=myVector.end();` //trỏ đến vị trí kết thúc của vector

Vị trí kết thúc **không** trỏ đến phần tử cuối cùng, mà là vị trí phần tử liền sau phần tử cuối cùng. Ví dụ, nếu vector có 10 phần tử được đánh số từ 0 đến 9 thì phần tử `end()` là phần tử có số thứ tự là 10.



33

iterator

```
it is begin, We have value: 0
it is end, We have value: 1778385259
it is end, We have value: 10
```

```
#include <iostream>
#include <vector>
using namespace std;
```

```
int main()
{
    vector<int>:: reverse_iterator rit ;
    vector<int> myVector ;
    for(int i = 0 ; i <10; i++) myVector.push_back(i) ;

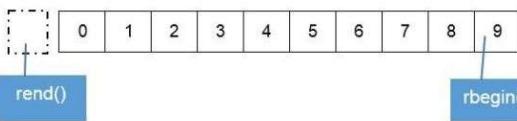
    rit = myVector.rbegin() ;
    cout<<"rit is rbegin, We have value: " << *rit<<endl;

    rit = myVector.rend() ;
    cout<<"rit is rend, We have value: " << *rit<<endl;

    myVector[-1] = -1 ;
    cout<<"rit is rend, We have value: " << *rit<<endl;
    return 0;
}
```

Lưu ý:

`it=myVector.rbegin();` //trỏ đến vị trí đầu tiên của vector theo chiều ngược (tức là phần tử cuối cùng)
`it=myVector.rend();` //trỏ đến vị trí kết thúc của vector theo chiều ngược, chú ý: `rend()` không trỏ đến phần tử đầu tiên của vector. Nếu phần tử đầu tiên của vector là phần tử thứ 0 thì `rend()` của vector là phần tử thứ -1.



reverse_iterator

```
rit is rbegin, We have value: 9
rit is rend, We have value: 268478965
rit is rend, We have value: -1
```

34

3.3. Thư viện STL trong C++: list

#include <list> List trong thư viện STL chính là danh sách liên kết đôi.

Khai báo:

```
list <int> list1; //khai báo list1 rỗng có kiểu dữ liệu là int
list <int> list2 (4, 100); //tạo list có 4 phần tử int, khởi tạo giá trị cho cả 4 phần tử này là 100
list <int> list3 (list2.begin(), list2.end()); //khai báo list kiểu int có tên là list3 sao chép từ đầu đến cuối list2
list <int> list4(list2); //tạo list list4 kiểu int và sao chép tất cả phần tử của list2; cách khai báo này cho kết quả giống như khai báo list3
```

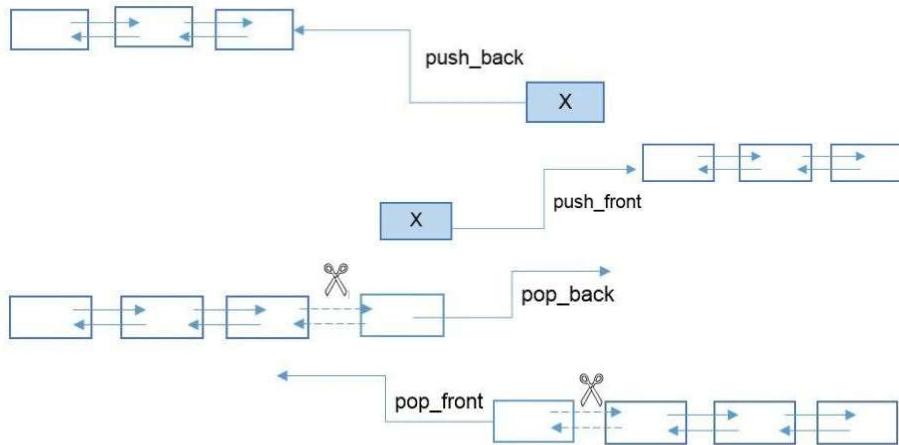
NGUYỄN KHÁNH PHƯƠNG 35
KHMT – SOICT - ĐHBK HN

3.3. Thư viện STL trong C++: list

Capacity	
size()	Trả về số lượng phần tử có trong list
empty()	True/False
Truy cập phần tử	
front()	Truy cập phần tử đầu tiên
back()	Truy cập phần tử cuối cùng
Truy vấn	
push_back(x)	Thêm phần tử có giá trị x vào cuối list
push_front(x)	Thêm phần tử có giá trị x vào đầu list
pop_back()	Loại bỏ phần tử cuối ra khỏi list
pop_front()	Loại bỏ phần tử đầu ra khỏi list
insert(iterator pos, const x)	Chèn x vào trước vị trí pos (x có thể là một phần tử hoặc iterator của một đoạn phần tử)
insert(iterator pos, int n, const x)	Chèn n phần tử có giá trị x vào trước vị trí pos
erase(iterator pos)	Xóa phần tử tại vị trí pos ra khỏi list

36

3.3. Thư viện STL trong C++: list



push_back(x)	Thêm phần tử có giá trị x vào cuối list
push_front(x)	Thêm phần tử có giá trị x vào đầu list
pop_back()	Loại bỏ phần tử cuối ra khỏi list
pop_front()	Loại bỏ phần tử đầu ra khỏi list

NGUYỄN KHÁNH PHƯƠNG 37
KHMT – SOICT - ĐHBK HN

3.3. Thư viện STL trong C++: list

Truy vấn	
swap(list L)	Hoán đổi các phần tử của L và list hiện hành
clear()	Xóa list
Operations	
splice (iterator pos, list& other)	Ghép list other vào vị trí pos của list hiện hành, sau khi nối ghép, list other không còn phần tử nào.
splice (iterator pos, list& other, iterator first, iterator last)	Ghép các phần tử trong khoảng [first,last) của list other và list hiện hành. Sau khi ghép nối, list other chỉ còn lại các phần tử trong đoạn [last, other.end())
splice (iterator positon, list& other, iterator it)	Ghép phần tử có vị trí là it của list other vào vị trí pos của list hiện hành. Sau khi ghép nối, list other bị mất đi phần tử ở vị trí it.
remove (const val)	Loại bỏ tất cả phần tử có giá trị bằng val trong list.
remove_if (predicate pred)	Loại bỏ tất các phần tử trong list nếu pred return true. Biểu thức predicate pred có thể là một hàm (function) trả về giá trị bool hoặc một class.

NGUYỄN KHÁNH PHƯƠNG 38
KHMT – SOICT - ĐHBK HN

3.3. Thư viện STL trong C++: list

Operations	
sort()	Sắp xếp các phần tử của list theo chiều tăng dần.
sort (compare comp)	Sắp xếp các phần tử của list tăng dần theo tiêu chí so sánh comp. Biểu thức compare comp có thể là một hàm trả về giá trị bool.
reverse()	Đảo ngược lại các phần tử của list.
unique()	Loại bỏ các phần tử thích hợp sao cho các phần tử còn lại trong list chỉ xuất hiện đúng duy nhất 1 lần.
unique (BinaryPredicate binary_pred)	Loại bỏ các phần tử thích hợp sao cho các phần tử còn lại trong list chỉ xuất hiện đúng duy nhất 1 lần dựa theo tiêu chí của hàm binary_pred. Biểu thức binary_pred có thể là hàm (function) hoặc lớp (class). Lưu ý: Các phần tử trong list phải được sắp xếp, nếu không sắp xếp thì phương thức unique sẽ cho kết quả sai.
merge (list& L)	Ghép list L vào sau vị trí cuối cùng list hiện hành. Sau khi ghép nối, list L không còn phần tử nào.
merge (list& L, compare comp)	Ghép các phần tử của list L vào list hiện hành với vị trí dựa theo biểu thức comp. Biểu thức comp là một hàm trả về giá trị bool.

Ví dụ: list

```
#include <iostream>
#include <list>
using namespace std;

// a predicate implemented as a function:
bool single_digit (const int& value) {
    return (value<10);
}
// a predicate implemented as a class:
struct is_odd {
    bool operator() (const int& value) {
        return (value%2)==1;
    }
};
int main ()
{
    int myArray[] = {17,1,10,2,20,3,30,4,15, 40};
    std::list<int> mylist (myArray,myArray+10); // 17 1 10 2 20 3 30 4 15 40

    //Loai bo cac phan tu co 1 chu so ra khoi list:
    mylist.remove_if (single_digit); // 17 10 20 30 15 40
    cout << "mylist sau khi xoa cac so co 1 chu so:";
    for (list<int>::iterator it = mylist.begin(); it!=mylist.end(); it++) cout << *it<< " ";

    //Loai bo cac phan tu le ra khoi list:
    mylist.remove_if (is_odd()); // 10 20 30 40
    cout << endl<< "mylist sau khi xoa cac so le:";
    for (list<int>::iterator it = mylist.begin(); it!=mylist.end(); it++) cout << *it<< " ";
    return 0;
}
```

40

Ví dụ: Tạo list chứa các phần tử kiểu string bao gồm “AAAA”, “BBB”, “CC”, “D”.
Sắp xếp lại các phần tử trong list theo chiều tăng dần độ dài của xâu.

```
#include <iostream>
#include <list>
using namespace std;

bool compare_string (const string& first, const string& second)
{
    if(first.length() < second.length())
        return true;
    else
        return false ;
}

int main ()
{
    //create list
    string myArray[] = {"AAAA", "BBB", "CC", "D"};
    std::list<string> mylist (myArray,myArray+4);

    //sap xep theo thu tu tang dang ve do dai cua cac xau:
    mylist.sort(compare_string) ;

    cout << "mylist contains:" ;
    for (list<string>::iterator it = mylist.begin(); it!=mylist.end(); it++) cout << *it<< " ";
    return 0;
}
```

7.1

Ví dụ: Cho một list bao gồm các 8 số thập phân: 1.1,2.5,3.1,4.7,1.4,2.4,3.5,4.2. Giả sử các số thập phân có phần nguyên giống nhau thì được xem là một số duy nhất. Hãy loại bỏ các phần tử sao cho các phần tử giống nhau chỉ xuất hiện 1 lần.

```
#include <iostream>
#include <list>
using namespace std;

bool same_integral_part (double first, double second)
{
    return ( int(first)==int(second) );
}

int main ()
{
    double myArray[] = {1.1,2.5,3.1,4.7,1.4,2.4,3.5,4.2};
    list<double> mylist (myArray,myArray+8); //1.1,2.5,3.1,4.7,1.4,2.4,3.5,4.2

    /*Truoc khi dung ham unique phai sap xep mang theo thu tu tang dan
    neu ko ham unique se cho ket qua sai*/
    mylist.sort(); // 1.1 1.4 2.4 2.5 3.1 3.5 4.2 4.7

    //Trong cac so co cung phan nguyen cua list, chi giu lai 1 so:
    mylist.unique(same_integral_part); //1.1 2.4 3.1 4.2
    cout << "mylist gom cac so: " ;
    for (list<double>::iterator it = mylist.begin(); it!=mylist.end(); it++)
        cout << *it<< " ";

    return 0;
}
```

42

Bài tập: Kết quả in ra màn hình là gì ? Giải thích

```
#include <iostream>
#include <list>
using namespace std ;

bool mycomparison (double first, double second)
{
    return ( (first)>(second) );
}

int main ()
{
    list<double> first, second;

    first.push_back (3.1);
    first.push_back (2.2);
    first.push_back (2.9); // 3.1 2.2 2.9

    second.push_back (3.7);
    second.push_back (7.1);
    second.push_back (1.4);
    second.push_back (2.1); // 3.7 7.1 1.4 2.1

    first.merge(second,mycomparison);

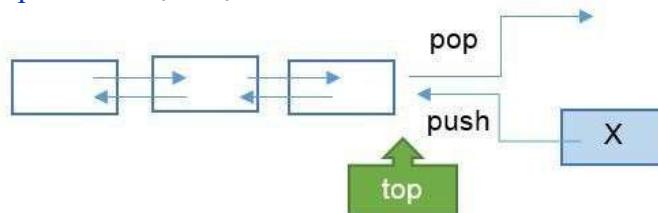
    cout << "first list gom cac so:" ;
    for (list<double>::iterator it=first.begin(); it!=first.end();it++)
        cout <<*it<< " " ;

    return 0;
}
```

NGUYỄN KHÁNH PHƯƠNG 43
KHMT – SOICT-ĐHBKHN

3.4. Thư viện STL trong C++: stack

`#include <stack>` một kiểu danh sách (list) mà việc bổ sung và loại bỏ một phần tử được thực hiện ở cuối danh sách.



Ví dụ khai báo:

`stack <int> s1; //khai báo stack s1 có kiểu dữ liệu là int`

- Ta không thể khai báo `stack<int> myStack (5)` để khai báo 5 phần tử rỗng tương tự như vector hay list vì stack không cho phép ta khai báo phần tử rỗng.
- Tương tự, ta không thể khai báo `stack<int> myStack (5,100)` để khai báo 5 phần tử của stack có giá trị 100.

NGUYỄN KHÁNH PHƯƠNG 44
KHMT – SOICT-ĐHBKHN

3.4. Thư viện STL trong C++: stack

Capacity	
size()	Trả về số lượng phần tử có trong stack
empty()	True/False
Truy cập phần tử	
top()	Truy cập phần tử ở đỉnh stack, tức là phần tử được thêm vào sau cùng
Truy vấn	
push(x)	Thêm phần tử có giá trị x vào đỉnh stack. Kích thước stack sẽ tăng thêm 1
pop()	Loại bỏ phần tử ở đỉnh stack
swap(stack s1, stack s2)	Hoán đổi các phần tử của 2 stack s1 và s2 với nhau

NGUYỄN KHÁNH PHƯƠNG 45
KHMT – SOICT - ĐHBK HN

Ví dụ: Viết chương trình cho người dùng nhập vào kích thước stack và nhập vào giá trị các phần tử. Sau đó, loại bỏ đi một nửa các phần tử ở đỉnh stack.

```
#include <iostream>
#include <stack>
using namespace std;
int main()
{
    stack<int> myStack;
    int n; // size of stack
    cout<<"Size: " ; cin>>n ;
    // create stack
    int tempNumber ;
    for(int i = 0 ; i<n; i++) {
        cin>>tempNumber ;
        myStack.push(tempNumber ) ;
    }
    // delete element
    for(int i = 0 ; i<n/2; i++) myStack.pop() ;

    // print stack
    n = myStack.size() ; // after using pop(), the size of stack < n
    for(int i = 0 ; i<n; i++) {
        tempNumber = myStack.top() ;
        myStack.pop();
        cout<<tempNumber<<" " ;
    }
    return 0;
}
```

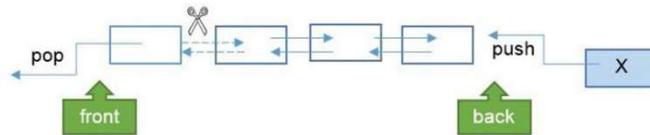
Lưu ý: Không thể sử dụng vòng lặp với biến iterator như đối list để in tất cả các phần tử vì stack chỉ cho phép truy xuất duy nhất phần tử ở đỉnh stack, ví dụ mã lệnh truy xuất stack như bên dưới là sai:

```
// WRONG way to print stack
stack<int>::iterator it ;
for( it = myStack.begin() ; it!=myStack.end(); it++) cout<<*it<<" " ;
```

46

3.5. Thư viện STL trong C++: queue

- `#include <queue>` một kiểu danh sách (list) mà việc bổ sung được thực hiện ở cuối danh sách và loại bỏ ở đầu danh sách.



Ví dụ khai báo:

```
queue <int> s1; //khai báo stack s1 có kiểu dữ liệu là int
```

- Tương tự như stack, ta không thể khai báo `queue<int> myQueue (5)` để khai báo 5 phần tử rỗng tương tự như vector hay list vì queue không cho phép ta khai báo phần tử rỗng.
- Tương tự, ta không thể khai báo `queue<int> myQueue (5, 100)` để khai báo 5 phần tử của queue có giá trị 100.

NGUYỄN KHÁNH PHƯƠNG 47
KHMT – SOICT - ĐHBK HN

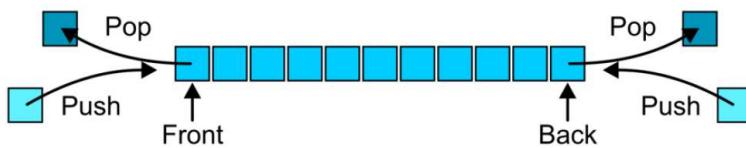
3.5. Thư viện STL trong C++: queue

Capacity	
<code>size()</code>	Trả về số lượng phần tử có trong queue
<code>empty()</code>	True/False
Truy cập phần tử	
<code>front()</code>	Truy cập phần tử ở đầu queue, tức là phần tử được thêm vào đầu tiên
<code>back()</code>	Truy cập phần tử ở cuối queue, tức là phần tử cuối cùng được thêm vào
Truy vấn	
<code>push(x)</code>	Thêm phần tử có giá trị x vào cuối queue. Kích thước queue sẽ tăng thêm 1
<code>pop()</code>	Loại bỏ phần tử ở đầu queue. Kích thước queue sẽ giảm 1
<code>swap(queue q1, queue q2)</code>	Hoán đổi các phần tử của 2 queue q1 và q2 với nhau

NGUYỄN KHÁNH PHƯƠNG 48
KHMT – SOICT - ĐHBK HN

3.6. Thư viện STL trong C++: deque

- `#include <deque>` Hàng đợi hai đầu (Deque = Double-Ended Queue; đọc là deck): cấu trúc dữ liệu có tính chất của cả Stack và Queue, nghĩa là cho phép thêm và xóa ở cả 2 đầu.
- deque tương tự như vector nhưng ta có thể thao tác thêm hoặc loại phần tử ở cả hai đầu của danh sách.
- Deque có các ưu điểm như:
 - Các phần tử có thể truy cập thông qua chỉ số vị trí của nó (tương tự như vector).
 - Chèn hoặc xóa phần tử ở cuối hoặc đầu của dãy (vector chỉ chèn hoặc xóa phần tử ở cuối dãy).



NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - ĐHBK HN

3.6. Thư viện STL trong C++: deque

Capacity	
<code>size()</code>	Trả về số lượng phần tử có trong queue
<code>empty()</code>	True/False
Truy cập phần tử	
<code>Operator [int i]</code>	Truy cập phần tử thứ i của deque, tương tự như ở mảng
<code>at(int i)</code>	Truy cập phần tử thứ i của deque
<code>front()</code>	Truy cập phần tử đầu tiên của deque
<code>back()</code>	Truy cập phần tử cuối cùng của deque
Truy vấn	
<code>push_back(x)</code>	Thêm phần tử có giá trị x vào cuối deque.
<code>push_front(x)</code>	Thêm phần tử có giá trị x vào đầu deque.
<code>pop_back()</code>	Loại bỏ phần tử cuối ra khỏi deque.
<code>pop_front()</code>	Loại bỏ phần tử đầu ra khỏi deque.

NGUYỄN KHÁNH PHƯƠNG 50
KHMT – SOICT - ĐHBK HN

3.6. Thư viện STL trong C++: deque

Truy vấn	
insert (iterator pos,const x)	Chèn phần tử có giá trị x vào trước vị trí pos.
insert (iterator pos,int n, const x)	Chèn n phần tử có giá trị x vào trước vị trí pos.
insert (iterator pos, iterator a, iterator b)	Chèn vào trước vị trí pos tất cả các phần tử trong khoảng [a,b) của một deque khác.
erase (iterator pos)	Xóa phần tử ở vị trí pos
erase (iterator first, iterator last)	xóa tất cả các phần tử trong khoảng [first,last), tức là từ phần tử thứ first đến phần tử thứ (last-1)
swap(vector v)	Hoán đổi các phần tử của deque hiện hành và vector v
clear()	Xóa deque

NGUYỄN KHÁNH PHƯƠNG 51
KHMT – SOICT-ĐHBK HN

Ví dụ: tạo một deque có 5 phần tử đều có giá trị là 100. Tạo một mảng int có 3 phần tử lần lượt là 200, 300, 400.

Bước 1: Chèn 2 phần tử đầu của mảng vào đầu deque.

Bước 2: Chèn 2 phần tử đầu của mảng vào trước phần tử thứ hai của deque

```
#include <iostream>
#include <deque>
using namespace std ;
int main ()
{
    // create deque
    deque<int> myDeque (5,100); // 100 100 100 100 100
    //create array
    int myArray[3] = {200,300,400} ; // 200 300 400

    //chèn hai phan tu dau tien cua myArray vao dau deque
    myDeque.insert(myDeque.begin(), myArray, myArray +2 ) ; // 200 300 100 100 100 100 100
    // print deque
    cout<<"Deque co cac so: ";
    for(int i = 0 ; i < myDeque.size() ; i++) cout<<myDeque[i]<< " ";

    //chèn hai phan tu dau tien cua myArray vao truoc phan tu thu 2 cua deque
    myDeque.insert(myDeque.begin()+1, myArray, myArray +2 ) ; // 200 200 300 300 100 100 100 100
    // print deque
    cout<<endl<<"Deque co cac so: ";
    for(int i = 0 ; i < myDeque.size() ; i++) cout<<myDeque[i]<< " ";

    return 0;
}
```

NGUYỄN KHÁNH PHƯƠNG 52
KHMT – SOICT-ĐHBK HN

3.7. Hàng đợi có ưu tiên - Priority Queues

- Cho tập S thường xuyên biến động, mỗi phần tử x được gán với một giá trị gọi là khóa (hay độ ưu tiên). Cần một cấu trúc dữ liệu hỗ trợ hiệu quả các thao tác chính sau:
 - $\text{Insert}(S, x)$: bổ sung phần tử x vào S
 - $\text{Max}(S)$: trả lại phần tử lớn nhất
 - $\text{Extract-Max}(S)$: loại bỏ và trả lại phần tử có khóa lớn nhất (ưu tiên nhất)
 - $\text{Increase-Key}(S, x, k)$: tăng khóa của x thành k

Cấu trúc dữ liệu đáp ứng các yêu cầu đó là **hàng đợi có ưu tiên**.

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - ĐHBK HN

Các phép toán đối với hàng đợi có ưu tiên Operations on Priority Queues

- Hàng đợi có ưu tiên (max)** có các phép toán cơ bản sau:
 - $\text{Insert}(S, x)$: bổ sung phần tử x vào tập S
 - $\text{Extract-Max}(S)$: loại bỏ và trả lại phần tử của S với khóa lớn nhất
 - $\text{Maximum}(S)$: trả lại phần tử của S với khóa lớn nhất
 - $\text{Increase-Key}(S, x, k)$: tăng giá trị của khóa của phần tử x lên thành k (Giả sử $k \geq$ khóa hiện tại của x)
- Hàng đợi có ưu tiên (min)** có các phép toán cơ bản sau:
 - $\text{Insert}(S, x)$: bổ sung phần tử x vào tập S
 - $\text{Extract-Min}(S)$: loại bỏ và trả lại phần tử của S với khóa nhỏ nhất
 - $\text{Minimum}(S)$: trả lại phần tử của S với khóa nhỏ nhất
 - $\text{Decrease-Key}(S, x, k)$: giảm giá trị của khóa của phần tử x xuống thành k (Giả sử $k \leq$ khóa hiện tại của x)

3.7. Hàng đợi ưu tiên (Priority queue)

Một số cách cài đặt hàng đợi có ưu tiên:

- Mảng (Array)
- Danh sách liên kết (Linked list)
- Đống (Heap)
- Priority_queue có sẵn trong thư viện <queue> của STL

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

(1) Cài đặt Priority queue: dùng Mảng

Hàng đợi có ưu tiên max: các phần tử trong mảng sắp xếp theo thứ tự giảm dần của giá trị khóa

Các thao tác cơ bản:

- Tìm phần tử lớn nhất: là phần tử đầu tiên của mảng $O(1)$
- Lấy và loại phần tử lớn nhất: xóa phần tử đầu tiên, dịch chuyển các phần tử còn lại lên trước 1 vị trí $O(n)$
- Bổ sung phần tử mới: tìm vị trí đúng để chèn phần tử mới $O(n)$
- Tăng giá trị phần tử: tìm vị trí mới cho phần tử $O(n)$

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

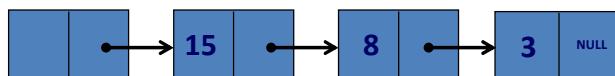
(2) Cài đặt Priority queue: dùng danh sách liên kết

Hàng đợi có ưu tiên max: các phần tử trong danh sách liên kết sắp xếp theo thứ tự giảm dần của giá trị khóa

Các thao tác cơ bản:

- Tìm phần tử lớn nhất: $O(1)$
- Lấy và loại phần tử lớn nhất: $O(1)$
- Bổ sung phần tử mới: $O(n)$
- Tăng giá trị phần tử: $O(n)$

header



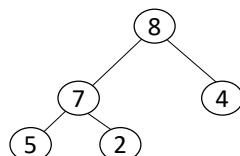
NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - ĐHBK HN

(3) Cài đặt Priority queue: dùng Heap

Hàng đợi có ưu tiên max: cài đặt bằng max-heap

Các thao tác cơ bản:

- Tìm phần tử lớn nhất: Heap-Maximum $O(1)$
- Lấy và loại phần tử lớn nhất: Heap-Extract-Max $O(\log n)$
- Bổ sung phần tử mới: Heap-Insert $O(\log n)$
- Tăng giá trị phần tử: Heap-Increase-Key $O(\log n)$



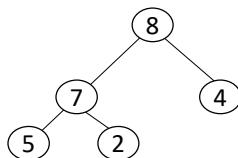
NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - ĐHBK HN

Cấu trúc dữ liệu đống

- Định nghĩa: Đống (heap) là cây nhị phân có hai tính chất sau:
 - Tính cấu trúc (Structural property): tất cả các mức đều là đầy, ngoại trừ mức cuối cùng, mức cuối được điền từ trái sang phải
 - Tính có thứ tự hay tính chất đống (heap property): với mỗi nút x

Parent(x) $\geq x$: max-heap

OR Parent(x) $\leq x$: min-heap



Từ tính chất đống, suy ra:
“Gốc chứa phần tử lớn nhất của max-heap!”

MAX-Heap

Đống là cây nhị phân được điền theo thứ tự

(4) Cách cài đặt Priority queue: dùng thư viện queue có sẵn trên C++

Các thao tác cơ bản:

- priority_queue::size()**: trả về số phần tử đang có trong queue.
- priority_queue::empty()**: queue rỗng thì hàm trả về giá trị true.
- priority_queue::push()**: thêm một phần tử vào hàng đợi.
- priority_queue::pop()**: xóa phần tử đang nằm ở đầu hàng đợi ra khỏi nó.
- priority_queue::top()**: trả về phần tử đang nằm ở đầu hàng đợi (phần tử có độ ưu tiên cao nhất)

Hai loại hàng đợi có ưu tiên:

- Hàng đợi có ưu tiên max (max priority queue)**: các phần tử trong queue được sắp xếp theo thứ tự giảm dần của độ ưu tiên (phần tử có độ ưu tiên cao nhất sẽ đứng đầu hàng đợi)
- Hàng đợi có ưu tiên min (min priority queue)**: các phần tử trong queue được sắp xếp theo thứ tự tăng dần của độ ưu tiên (phần tử có độ ưu tiên thấp nhất sẽ đứng đầu hàng đợi)

(4) Cách cài đặt Priority queue: dùng thư viện queue có sẵn trên C++

Cách khai báo max priority queue:

```
priority_queue <objectType> queue_name;  
giống như max_heap
```

Ví dụ:

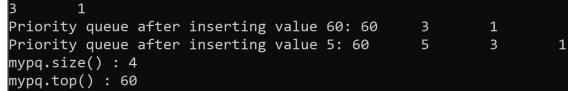
- Khai báo **max priority queue** gồm các phần tử kiểu số nguyên
 - priority_queue <int> myqueue;

Các số nguyên trong queue sẽ sắp xếp theo giá trị giảm dần (độ ưu tiên giảm dần), tức là số nguyên có giá trị lớn nhất (ưu tiên nhất) sẽ nằm ở đầu hàng đợi, tiếp đến là số nguyên có giá trị lớn thứ hai, thứ ba,...

- Khai báo max priority queue gồm các phần tử là kiểu **pair** (sắp xếp theo thành phần đầu tiên của pair theo thứ tự ưu tiên giảm dần, nếu thành phần đầu tiên bằng nhau thì sắp xếp theo thành phần thứ hai của pair theo thứ tự ưu tiên giảm dần):
 - pair 2 số nguyên:** priority_queue <pair<int, int> > myqueue;
 - pair (số nguyên, kí tự):** priority_queue <pair<int, char> > myqueue;

(4) Cách cài đặt Priority queue: dùng thư viện queue có sẵn trên C++

```
#include <iostream>  
#include <queue>  
using namespace std;  
void displaypq(priority_queue <int> pq)  
{  
    while (!pq.empty())  
    {  
        cout << pq.top() << " \t";  
        pq.pop();  
    }  
    cout << '\n';  
}  
int main ()  
{  
    priority_queue <int> mypq; → Max priority queue gồm các số nguyên  
    mypq.push(1); mypq.push(3); displaypq(mypq);  
    mypq.push(60);  
    cout << "\nPriority queue after inserting value 60: ";  
    displaypq(mypq);  
    mypq.push(5);  
    cout << "\nPriority queue after inserting value 5: ";  
    displaypq(mypq);  
    cout << "mypq.size() : " << mypq.size() << endl;  
    cout << "mypq.top() : " << mypq.top() << endl;  
    cout << "mypq.pop() : " ; mypq.pop(); displaypq(mypq);  
    return 0;  
}
```



(4) Cách cài đặt Priority queue: dùng thư viện queue có sẵn trên C++

```
#include <bits/stdc++.h>
using namespace std;
void displaypq(priority_queue<pair<int, int> > pq)
{
    while (!pq.empty())
    {
        pair<int, int> top = pq.top();
        cout << "(" << top.first << ", " << top.second << ")" << endl;
        pq.pop();
    }
    cout << '\n';
}
int main()
{
    // By default a max heap is created ordered by first element of pair.
    priority_queue<pair<int, int> > pq;
    pq.push(make_pair(10, 200));    So sánh thành phần thứ nhất: 20 > 10
    pq.push(make_pair(20, 100));    → (20, 100) sẽ đứng ở đầu queue, tiếp theo là (10, 200)
    displaypq(pq);
    pq.push(make_pair(15, 400));
    cout << "\nPriority queue after inserting pair (15, 400):\n ";
    displaypq(pq);
    cout << "pq.size() : " << pq.size() << endl;
    pair<int, int> top = pq.top();
    cout << "pq.top() : (" << top.first << ", " << top.second << ")" << endl;
    cout << "pq.pop() : "; pq.pop(); displaypq(pq);
    return 0;
}
```

(4) Cách cài đặt Priority queue: dùng thư viện queue có sẵn trên C++

Cách khai báo min priority queue:

```
priority_queue<objectType, vector<objectType>, greater<objectType> > queue_name;  
giống như min_heap
```

Ví dụ:

- Khai báo min priority queue gồm các phần tử kiểu số nguyên:

```
priority_queue<int, vector<int>, greater<int> > pq;
```
- Khai báo min priority queue gồm các phần tử là kiểu **pair** (sắp xếp theo thành phần đầu tiên của pair theo thứ tự tăng dần, nếu thành phần thứ nhất bằng nhau thì sắp xếp theo thành phần thứ hai của pair theo thứ tự tăng dần):
– **pair 2 số nguyên:**

```
typedef pair<int, int> pi;
priority_queue<pi, vector<pi>, greater<pi> > pq;
– pair (số nguyên, kí tự):
typedef pair<int, char> pi;
priority_queue<pi, vector<pi>, greater<pi> > pq;
```

(4) Cách cài đặt Priority queue: dùng thư viện queue có sẵn trên C++

Cách khai báo min priority queue:

```
priority_queue <objectType, vector <objectType>, greater <objectType> > queue_name;

#include <iostream>
#include <queue>
using namespace std;
void displaypq(priority_queue <int, vector <int>, greater<int> > pq)
{
    while (!pq.empty())
    {
        cout << pq.top() << " \t";
        pq.pop();
    }
    cout << '\n';
}
int main ()
{
    priority_queue <int, vector <int>, greater<int> > mypq;

    mypq.push(3); mypq.push(1); mypq.push(15); displaypq(mypq);
    mypq.push(60);
    cout << "\nPriority queue after inserting value 60: ";
    displaypq(mypq);
    mypq.push(5);
    cout << "\nPriority queue after inserting value 5: ";
    displaypq(mypq);
    cout << "mypq.size() : " << mypq.size() << endl;
    cout << "mypq.top() : " << mypq.top() << endl;
    cout << "mypq.pop() : "; mypq.pop(); displaypq(mypq);
    return 0;
}
```

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - ĐHBK HN

(4) Cách cài đặt Priority queue: dùng thư viện queue có sẵn trên C++

Cách khai báo min priority queue:

```
priority_queue <objectType, vector <objectType>, greater <objectType> > queue_name;
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pi;

void displaypq(priority_queue<pi, vector<pi>, greater<pi> > pq)
{
    while (!pq.empty())
    {
        pair<int, int> top = pq.top();
        cout << "(" << top.first << ", " << top.second << ")" \t";
        pq.pop();
    }
    cout << '\n';
}
int main()
{
    // By default a max heap is created ordered by first element of pair.
    priority_queue<pi, vector<pi>, greater<pi> > pq;
    pq.push(make_pair(30, 100)); pq.push(make_pair(10, 200)); pq.push(make_pair(20, 100));
    displaypq(pq);
    pq.push(make_pair(10, 100));
    cout << "\nPriority queue after inserting pair (10, 100): \n ";
    displaypq(pq);
    cout << "pq.size() : " << pq.size() << endl;
    pair<int, int> top = pq.top();
    cout << "pq.top() : (" << top.first << ", " << top.second << ")" << endl;
    cout << "pq.pop() : "; pq.pop(); displaypq(pq);
    return 0;
}
```

Bài tập 1: Hàng đợi có ưu tiên min

- Bình cần thực hiện n công việc: mỗi công việc i có
 - p_i là lượng thời gian cần để hoàn thành công việc i
 - r_i là thời điểm sớm nhất có thể bắt đầu thực hiện công việc i

Gọi c_i là thời điểm mà công việc i sẽ hoàn thành. Hãy xác định trình tự thực hiện các công việc sao cho thời gian hoàn thành trung bình của n công việc này là nhỏ nhất: $\min \frac{1}{n} \sum_{i=1}^n c_i$

Thuật toán tham lam: công việc nào có lượng thời gian hoàn thành còn lại ít nhất thì được thực hiện trước.

Ví dụ: có 4 công việc cần thực hiện $(r_i, p_i) = (4, 4), (3, 7), (7, 2)$

- Sắp xếp các công việc theo thứ tự tăng dần của thời điểm r_i : $(3, 7)_{cv2}, (4, 4)_{cv1}, (7, 2)_{cv3}$
- Thời điểm 3: thực hiện cv2 đến thời điểm 4 (vì lúc này xuất hiện thêm cv1).
- Thời điểm 4: có 2 công việc $(4, 4)_{cv1}, (3, 6)_{cv2}$ → thực hiện cv1 vì $p_1 < p_2$, thực hiện cho đến thời điểm 7 (vì lúc này xuất hiện thêm cv3).
- Thời điểm 7: có 3 công việc $(4, 1)_{cv1}, (7, 2)_{cv3}, (3, 6)_{cv2}$ → thực hiện nốt cv1 vì $p_1 < p_3 < p_2$.
- Thời điểm 8: có 2 công việc $(7, 2)_{cv3}, (3, 6)_{cv2}$ → thực hiện hết cv3 đến thời điểm 10.
- Thời điểm 10: thực hiện hết cv2 cho đến thời điểm 16 thì kết thúc.

Vậy thời gian hoàn thành trung bình = $(c_1 + c_2 + c_3) / 3 = (8 + 16 + 10) / 3 = 11.3$

Sử dụng hàng đợi có ưu tiên min Q: lưu trữ lượng thời gian hoàn thành còn lại của **các công việc**

67

Bài tập 1: Hàng đợi có ưu tiên min

- Bình cần thực hiện n công việc: mỗi công việc i có
 - p_i là lượng thời gian cần để hoàn thành công việc i
 - r_i là thời điểm sớm nhất có thể bắt đầu thực hiện công việc i

Gọi c_i là thời điểm mà công việc i sẽ hoàn thành. Hãy xác định trình tự thực hiện các công việc sao cho thời gian hoàn thành trung bình của n công việc này là nhỏ nhất: $\min \frac{1}{n} \sum_{i=1}^n c_i$

Thuật toán tham lam: công việc nào có lượng thời gian hoàn thành còn lại ít nhất thì được thực hiện trước:

- Sắp xếp các công việc theo thứ tự tăng dần của thời điểm r_i .
- Sử dụng hàng đợi có ưu tiên min Q chứa lượng thời gian hoàn thành còn lại của các công việc đang được thực hiện: công việc nào có lượng thời gian hoàn thành còn lại ít hơn thì có độ ưu tiên cao hơn.
- Duyệt lần lượt từng công việc theo thứ tự tăng dần của r_i :
 - Đến thời điểm r_i : tức là ta có thể bắt đầu thực hiện công việc i → bổ sung i vào danh sách các công việc đang được thực hiện => cập nhật lại hàng đợi ưu tiên Q

68

Bài tập 1: Hãy viết chương trình thực hiện yêu cầu

Yêu cầu: Hãy xác định trình tự cùng thời điểm thực hiện các công việc mà Bình cần tuân theo để thời gian hoàn thành trung bình của n công việc này là nhỏ nhất, tức là tìm min của $(1/n) \sum_{i=1}^n c_i$.

Dữ liệu vào

- Đòng đầu tiên ghi số nguyên dương n là số lượng công việc Bình cần thực hiện;
- Đòng tiếp theo là n nhóm số nguyên dương: mỗi nhóm gồm hai số nguyên dương p_i và r_i ($i = 1, \dots, n$).

Kết quả

- Đòng đầu tiên ghi thời gian trung bình nhỏ nhất tìm được;
- Đòng tiếp theo là dãy K ($K \geq n$) các nhóm số nguyên dương ghi lại nhật ký làm việc của Bình: mỗi nhóm gồm 3 số nguyên dương i t t' trong đó i là tên công việc, t, t' lần lượt là thời điểm bắt đầu/tiếp tục và tạm dừng/hoàn thành thực hiện công việc i .

Ví dụ

stdin	stdout
2 3 3 6 2	8.5 2 2 3 1 3 6 2 6 11

69

```
#include <bits/stdc++.h>
using namespace std;

int n;
vector<pair<int, int>> a;

int main()
{
    cin >> n;
    long long p, r;
    for (int i=0; i<n; i++)
    {
        cin >> p; cin >> r;
        a.push_back(make_pair(r,p));
    }
    /* Sap xep theo thu tu tang dan cua thanh phan thu nhat cua vector a
     (tuc la theo gia tri r tang dan cua cac cong viec)*/
    sort(a.begin(),a.end());

    /*Khai bao hang doi uu tien Q chua cac so nguyen La
     Luong thoi gian con Lai de hoan thanh cac cong viec dang duoc thuc hien*/
    priority_queue<int, vector<int>, greater<int> > Q;
```

70

```

int t = 0; //thoi diem hien tai
int i = 0; int result = 0;
while (i < n || !Q.empty() )
{
    if (Q.empty())
    {
        t = a[i].first;
        Q.push(a[i].second);
        i++;
    }
    else
    {
        if (i < n)
        {
            /*Tinh Luong thoi gian tinh tu thoi diem hien tai
            den khi co the bat dau thuc hien cong viec i*/
            int remain = a[i].first - t;
            if (remain < Q.top() )
            {
                //thuc hien tiep cong viec hien tai cho den thoi diem co the bat dau thuc hien cv i
                int u = Q.top();
                Q.pop();
                Q.push(u - remain); //cap nhap lai luong thoi gian con lai de hoan thanh cv hien tai
                Q.push(a[i].second); //bo sung luong thoi gian thuc hien cong viec i vao Q
                t = a[i].first;
                i++;
            }
            else
            {
                t += Q.top(); //thuc hien noi cong viec hien tai cho den khi hoan thanh no
                result += t;
                Q.pop();
            }
            else
            {
                t += Q.top();
                result += t;
                cout<<result<<endl;
                Q.pop();
            }
        }
    }
}//end while
printf("%.2f", 1.0*result/n); //2 1 3 2 4 1
return 0;

```

71

Bài tập 2: Thuật toán Dijkstra

- Thuật toán Dijkstra được đề xuất để giải bài toán: Tìm đường đi ngắn nhất từ một đỉnh nguồn s đến tất cả các đỉnh còn lại trên đồ thị với **trọng số không âm** trên cạnh (Giả thiết: đồ thị không chứa chu trình âm)
- Trong quá trình thực hiện thuật toán, với mỗi đỉnh v ta sẽ lưu trữ nhãn của đỉnh gồm các thông tin sau:
 - $k[v]$: biến bun có giá trị true (=1) nếu ta đã tìm được đường đi ngắn nhất từ s đến v , ban đầu biến này được khởi tạo giá trị false (=0).
 - $d[v]$: khoảng cách ngắn nhất hiện biết từ s đến v . Ban đầu biến này được khởi tạo giá trị $+\infty$ đối với mọi đỉnh, ngoại trừ $d[s]$ được đặt bằng 0.
 - $p[v]$: là đỉnh đi trước đỉnh v trong đường đi có độ dài $d[v]$. Ban đầu, các biến này được khởi tạo rỗng (chưa biết).



Edsger W. Dijkstra

(1930-2002)

72

Giảm cận trên

Đang xây dựng đường đi ngắn nhất từ s đến v :

Giả sử đường đi ngắn nhất hiện biết từ s đến v : $s \dots z \rightarrow v$

Sử dụng cạnh (u, v) để kiểm tra xem đường đi đến v đã tìm được có thể làm ngắn hơn nhờ đi qua u hay không.

Relax(u, v)

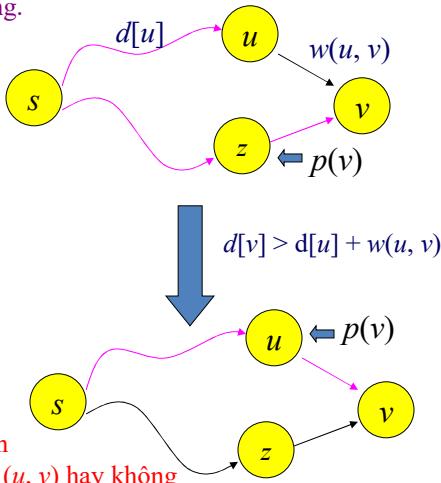
```

if ( $d[v] > d[u] + w(u, v)$ )
{
     $d[v] \leftarrow d[u] + w(u, v)$ 
     $p[v] \leftarrow u$ 
}

```

Relax(u, v) : đã tìm được ddnn từ s đến u ;

Do đó, kiểm tra xem có thể làm giảm ddnn hiện biết từ s đến v bằng cách đi qua cạnh (u, v) hay không



73

Thuật toán Dijkstra

- Thuật toán lặp lại các thao tác sau đây cho đến khi tất cả các đỉnh được khảo sát xong (nghĩa là $k[u] = \text{true}$ với mọi u):

- Trong tập đỉnh với $k[u] = \text{false}$: chọn đỉnh u có $d[u]$ là nhỏ nhất.

- Đặt $k[u] = \text{true}$.

- For each $v \in \text{Adj}[u]$

- if ($k[v] = \text{false}$): gọi Relax(u, v)

- if ($d[v] > d[u] + w(u, v)$): thì đặt lại $d[v] = d[u] + w(u, v)$ và

- $p[v] = u$

74

Ví dụ

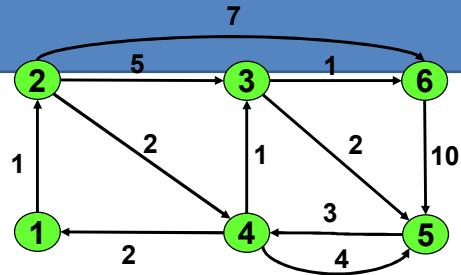
Tìm đường đi ngắn nhất từ đỉnh

1 đến tất cả các đỉnh còn lại

$d[v] = \delta(s, v)$;

$p[v]$ - đỉnh đi trước v trong đđnnn từ s đến v

$k[v] = \text{true}$ nếu đã tìm được đđnnn từ s đến v



	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6
Khởi tạo	[0, -, F]	[\infty, -, F]				
1						
2						
3						
4						
5						

75

Ví dụ

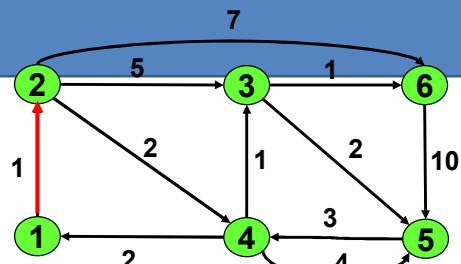
Tìm đường đi ngắn nhất từ đỉnh

1 đến tất cả các đỉnh còn lại

$d[v] = \delta(s, v)$;

$p[v]$ - đỉnh đi trước v trong đđnnn từ s đến v

$k[v] = \text{true}$ nếu đã tìm được đđnnn từ s đến v



Cập nhật lại nhãn cho các đỉnh còn lại

	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6
Khởi tạo	[0, -, F]*	[\infty, -, F]				
1		[1, 1, F]	[\infty, -, F]	[\infty, -, F]	[\infty, -, F]	[\infty, -, F]
2						
3						
4						
5						

76

Ví dụ

$d[2] = 1$
 $p[2] = 1$

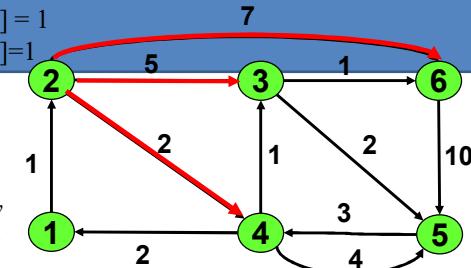
Tìm đường đi ngắn nhất từ đỉnh

1 đến tất cả các đỉnh còn lại

$d[v] = \delta(s, v)$;

$p[v]$ - đỉnh đi trước v trong đđnn từ s đến v

$k[v] = \text{true}$ nếu đã tìm được đđnn từ s đến v



Cập nhật lại nhãn cho các đỉnh còn lại

	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6
Khởi tạo	[0, -, F]*	[∞, -, F]	[∞, -, F]	[∞, -, F]	[∞, -, F]	[∞, -, F]
1		[1, 1, T]*	[∞, -, F]	[∞, -, F]	[∞, -, F]	[∞, -, F]
2			[6, 2, F]	[3, 2, F]	[∞, -, F]	[8, 2, F]
3			Xét đỉnh 3: if ($\infty > 1+5$) → cập nhật lại nhãn cho đỉnh 3			
4			Xét đỉnh 4: if ($\infty > 1+2$) → cập nhật lại nhãn cho đỉnh 4			
5			Xét đỉnh 5: if ($\infty > 1+\infty$)			

Xét đỉnh 6: if ($\infty > 1+7$) → cập nhật lại nhãn cho đỉnh 6 77

Ví dụ

$d[2] = 1$
 $p[2] = 1$

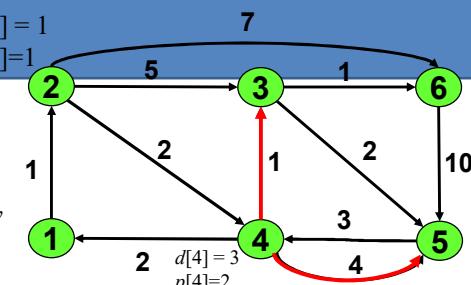
Tìm đường đi ngắn nhất từ đỉnh

1 đến tất cả các đỉnh còn lại

$d[v] = \delta(s, v)$;

$p[v]$ - đỉnh đi trước v trong đđnn từ s đến v

$k[v] = \text{true}$ nếu đã tìm được đđnn từ s đến v



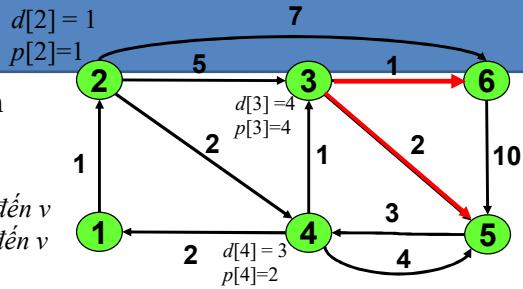
	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6
Khởi tạo	[0, -, F]*	[∞, -, F]	[∞, -, F]	[∞, -, F]	[∞, -, F]	[∞, -, F]
1		[1, 1, T]*	[∞, -, F]	[∞, -, F]	[∞, -, F]	[∞, -, F]
2			[6, 2, F]	[3, 2, T]*	[∞, -, F]	[8, 2, F]
3			[4, 4, F]		[7, 4, F]	[8, 2, F]
4			Xét đỉnh 3: if ($6 > 3+1$) → cập nhật lại nhãn cho đỉnh 3			
5						

Xét đỉnh 5: if ($\infty > 3+4$) → cập nhật lại nhãn cho đỉnh 5

Xét đỉnh 6: if ($8 > 3+\infty$)

Ví dụ

Tìm đường đi ngắn nhất từ đỉnh 1 đến tất cả các đỉnh còn lại
 $d[v] = \delta(s, v)$;
 $p[v]$ - đỉnh đi trước v trong đđnnn từ s đến v
 $k[v] = \text{true}$ nếu đã tìm được đđnnn từ s đến v



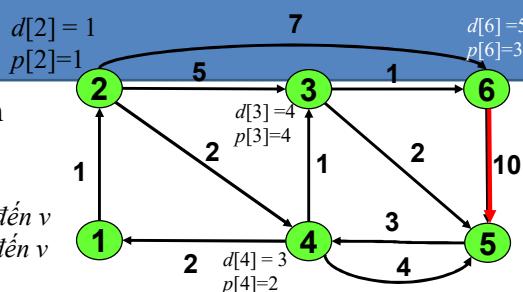
	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6
Khởi tạo	[0, -, F]*	[\infty, -, F]				
1		[1, 1, T]*	[\infty, -, F]	[\infty, -, F]	[\infty, -, F]	[\infty, -, F]
2			[6, 2, F]	[3, 2, T]*	[\infty, -, F]	[8, 2, F]
3				[4, 4, T]*	[7, 4, F]	[8, 2, F]
4					[6, 3, F]	[5, 3, F]
5						

Xét đỉnh 5: if ($7 > 4 + 2$) → cập nhật lại nhãn cho đỉnh 5

Xét đỉnh 6: if ($8 > 4 + 1$) → cập nhật lại nhãn đỉnh 6

Ví dụ

Tìm đường đi ngắn nhất từ đỉnh 1 đến tất cả các đỉnh còn lại
 $d[v] = \delta(s, v)$;
 $p[v]$ - đỉnh đi trước v trong đđnnn từ s đến v
 $k[v] = \text{true}$ nếu đã tìm được đđnnn từ s đến v

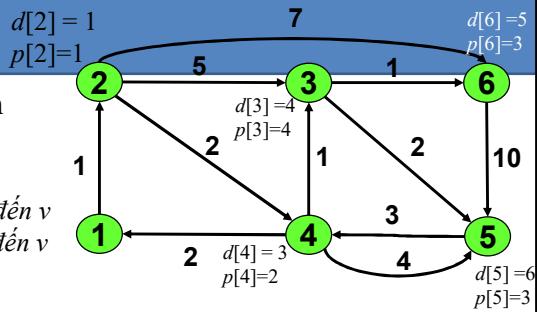


	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6
Khởi tạo	[0, -, F]*	[\infty, -, F]				
1		[1, 1, T]*	[\infty, -, F]	[\infty, -, F]	[\infty, -, F]	[\infty, -, F]
2			[6, 2, F]	[3, 2, T]*	[\infty, -, F]	[8, 2, F]
3				[4, 4, T]*	[7, 4, F]	[8, 2, F]
4					[6, 3, F]	[5, 3, F]*
5					[6, 3, F]	

Xét đỉnh 5: if ($6 > 5 + 10$)

Ví dụ

Tìm đường đi ngắn nhất từ đỉnh 1 đến tất cả các đỉnh còn lại
 $d[v] = \delta(s, v)$;
 $p[v]$ - đỉnh đi trước v trong đđnn từ s đến v
 $k[v] = \text{true}$ nếu đã tìm được đđnn từ s đến v



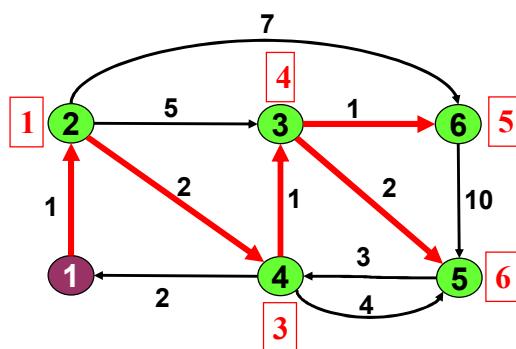
	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6
Khởi tạo	[0, -, F]*	[\infty, -, F]				
1		[1, 1, T]*	[\infty, -, F]	[\infty, -, F]	[\infty, -, F]	[\infty, -, F]
2			[6, 2, F]	[3, 2, T]*	[\infty, -, F]	[8, 2, F]
3				[4, 4, T]*	[7, 4, F]	[8, 2, F]
4					[6, 3, F]	[5, 3, F]*
5					[6, 3, F]*	

Kết luận: **đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại của đồ thị ????**

Cây đường đi ngắn nhất

- Tập cạnh $\{(p(v), v) : v \in V \setminus \{s\}\}$ tạo thành cây có gốc tại đỉnh nguồn s được gọi là cây đđnn xuất phát từ đỉnh s .

- Các cạnh màu đỏ tạo thành cây đđnn xuất phát từ đỉnh 1
 - Số màu đỏ viết bên cạnh mỗi đỉnh là độ dài đường đi ngắn nhất từ 1 đến nó.



Cài đặt thuật toán với các cấu trúc dữ liệu

Để cài đặt thuật toán Dijkstra chúng ta sử dụng bộ nhãn của các đỉnh:

- Nhãn của mỗi đỉnh v gồm 3 thành phần cho biết các thông tin:
 - $k[v]$ - đã tìm được đường đi ngắn nhất từ đỉnh nguồn đến v hay chưa,
 - $d[v]$ - khoảng cách (độ dài đường đi) từ s đến v hiện biết
 - $p[v]$ - đỉnh đi trước đỉnh v trong đường đi tốt nhất hiện biết.
- Các thành phần này sẽ được cất giữ tương ứng trong các biến $k[v]$, $d[v]$ và $p[v]$.

83

(1) Cài đặt trực tiếp

```
Dijkstra_Table(G, s)
1.   for u ∈ V do {
2.       d[u] ← infinity;
3.       p[u] ← NIL;
4.       k[u] ← FALSE;
5.   }
6.   d[s] ← 0;
    // s là đỉnh nguồn
7.   T = V;
```

```
8.   while T ≠ ∅ do {
9.       u ← đỉnh có d[u] là nhỏ nhất trong T;
10.      k[u]=TRUE;
11.      T = T-{u};
12.      for (v ∈ Adj(u) && !k[v]) do
13.          if (d[v] > d[u] + c[u, v]) {
14.              d[v] = d[u] + c[u, v];
15.              p[v] = u;
16.          }
17.      }
```

Để dàng nhận thấy rằng **Dijkstra_Table(G, s)** đòi hỏi thời gian $O(|V|^2+|E|)$.

Sử dụng hàng đợi có ưu tiên

- Để chứa thông tin gì ?
- Priority Queue min hay Priority Queue max ?

84

(2) Cài đặt thuật toán Dijkstra sử dụng hàng đợi có ưu tiên

Do tại mỗi bước ta cần tìm ra đỉnh với nhãn khoảng cách nhỏ nhất, nên để thực hiện thao tác này một cách hiệu quả ta sẽ sử dụng **hàng đợi có ưu tiên min Q** lưu trữ nhãn d với mỗi đỉnh của đồ thị.

NGUYỄN KHÁNH PHƯƠNG 85
Bộ môn KHMT – ĐHBKHN

(2) Cài đặt thuật toán Dijkstra sử dụng hàng đợi có ưu tiên

Dijkstra_Heap(G, s)

```
1.   for  $u \in V$  do {
2.        $d[u] \leftarrow \text{infinity}$ ;
3.        $p[u] \leftarrow \text{NIL}$ ;
4.        $k[u] \leftarrow \text{FALSE}$ ;
5.   }
6.    $d[s] \leftarrow 0$ ; //  $s$  là đỉnh nguồn
7.    $Q \leftarrow \text{Build\_Min\_Heap}(d[V])$ ; // Khởi tạo hàng đợi có ưu tiên  $Q$  từ  $d[v], v \in V$ 
8.   while Not Empty( $Q$ ) do {
9.        $u \leftarrow \text{Extract-Min}(Q)$ ; // loại bỏ gốc của  $Q$  và đưa vào  $u$ 
10.       $k[u]=\text{TRUE}$ ;
11.      for ( $v \in \text{Adj}(u) \ \&& \ !k[v]$ ) do
12.          if  $d[v] > d[u] + c[u, v]$  {
13.               $d[v] = d[u] + c[u, v]$ ;
14.               $p[v] = u$ ;
15.              Decrease_Key( $Q, v, d[v]$ );
16.          }
17.      }
```

NGUYỄN KHÁNH PHƯƠNG 86
Bộ môn KHMT – ĐHBKHN

Phân tích thời gian tính của thuật toán: sử dụng priority queue cài đặt bằng min heap

- Vòng lặp for ở dòng 1-5 đòi hỏi thời gian $O(|V|)$.
 - Việc khởi tạo min-heap (đống min) dòng 7 đòi hỏi thời gian $O(|V|)$.
 - Vòng lặp while ở dòng 8 lặp $|V|$ lần, do đó thao tác Extract-Min thực hiện $|V|$ lần, vậy đòi hỏi thời gian tổng cộng là $O(|V| \log |V|)$.
 - Thao tác Decrease_Key ở dòng 15 phải thực hiện không quá $O(|E|)$ lần. Do đó, thời gian thực hiện thao tác này trong thuật toán là $O(|E| \log |V|)$.
 - Vậy tổng cộng thời gian tính của thuật toán là $O((|E| + |V|) \log |V|)$
- $\sim O(|E| \log |V|)$ vì coi $|V| = O(|E|)$ trong trường hợp đồ thị là liên thông (đồ thị dày)

Dijkstra_Heap(G, s)

```

1.   for u ∈ V do {
2.       d[u] ← infinity;
3.       p[u] ← NIL;
4.       k[u] ← FALSE;
5.   }
6.   d[s] ← 0; // s là đỉnh nguồn
7.   Q ← Build_Min_Heap(d[V]); // Khởi tạo hàng đợi có ưu tiên Q từ d[v], v ∈ V
8.   while Not Empty(Q) do {
9.       u ← Extract-Min(Q); // loại bỏ gốc của Q và đưa vào u
10.      k[u]=TRUE;
11.      for (v ∈ Adj(u) && !k[v]) do
12.          if d[v] > d[u] + c[u, v] {
13.              d[v] = d[u] + c[u, v];
14.              p[v] = u;
15.              Decrease_Key(Q, v, d[v]);
16.          }
17.      }

```

Phân tích thời gian tính của thuật toán

- Chú ý: Nếu cài đặt bằng priority queue trong STL, ta sẽ không dùng thao tác Decrease_Key ở dòng 15 vì thư viện này không hỗ trợ. Vậy mỗi khi cần cập nhật (giảm) giá trị d[v] cho đỉnh v: ta sẽ bổ sung thêm (v, d[v]) mới này vào hàng đợi, tức là lúc này hàng đợi có thể có 1 đỉnh v xuất hiện nhiều lần với các giá trị d[v] khác nhau. Dù có nhiều phiên bản d[v] trong hàng đợi, nhưng với mỗi đỉnh v, ta sẽ chỉ làm việc với giá trị d[v] nhỏ nhất và bỏ qua các giá trị còn lại.

Dijkstra_Heap(G, s)

Khi đó, thời gian tính sẽ là

```

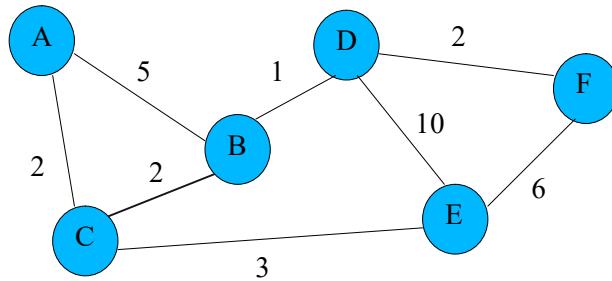
1.   for u ∈ V do {
2.       d[u] ← infinity;
3.       p[u] ← NIL;
4.       k[u] ← FALSE;
5.   }
6.   d[s] ← 0; // s là đỉnh nguồn
7.   Q ← Build_Min_Heap(d[V]); // Khởi tạo hàng đợi có ưu tiên Q từ d[v], v ∈ V
8.   while Not Empty(Q) do {
9.       u ← Extract-Min(Q); // loại bỏ gốc của Q và đưa vào u
10.      k[u]=TRUE;
11.      for (v ∈ Adj(u) && !k[v]) do
12.          if d[v] > d[u] + c[u, v] {
13.              d[v] = d[u] + c[u, v];
14.              p[v] = u;
15.              Decrease_Key(Q, v, d[v]);
16.          }
17.      }

```

if (!k[u])

Ví dụ: Dijkstra algorithm

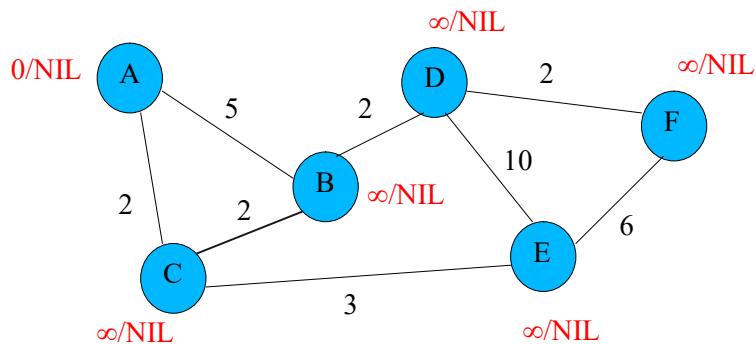
Tìm đường đi ngắn nhất từ đỉnh A đến tất cả các đỉnh còn lại



NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - ĐHBK HN

Thuật toán Dijkstra: Khởi tạo

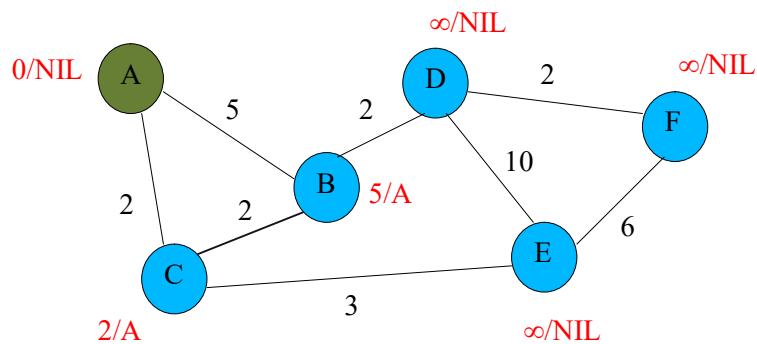
Priority queue: A, B, C, D, E, F



NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - ĐHBK HN

Thuật toán Dijkstra: Có định nhãn đỉnh A

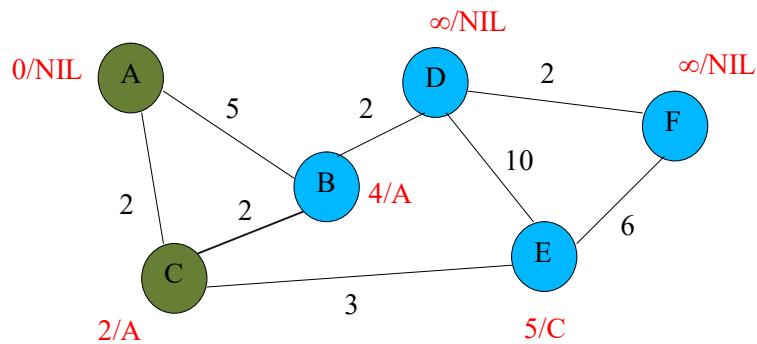
Priority queue: C, B, D, E, F



NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - ĐHBK HN

Thuật toán Dijkstra: Có định nhãn đỉnh C

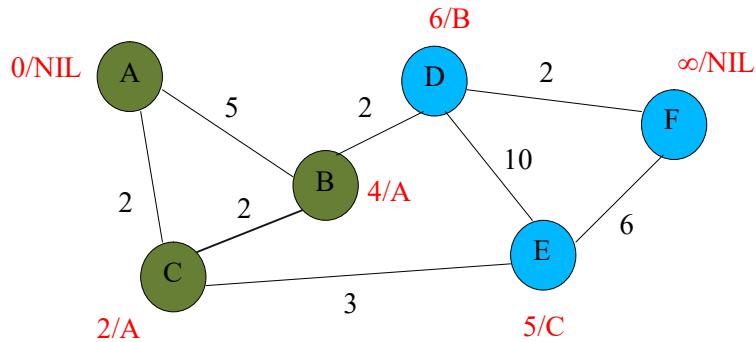
Priority queue: B, E, D, F



NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - ĐHBK HN

Thuật toán Dijkstra: Có định nhãn đỉnh B

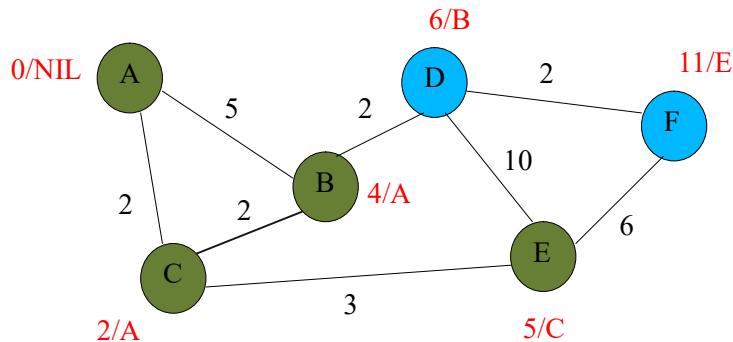
Priority queue: E, D, F



NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

Thuật toán Dijkstra: Có định nhãn đỉnh E

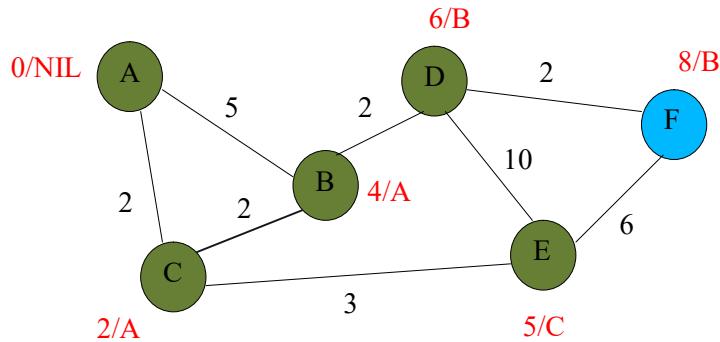
Priority queue: D, F



NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

Thuật toán Dijkstra: Có định nhãn đỉnh D

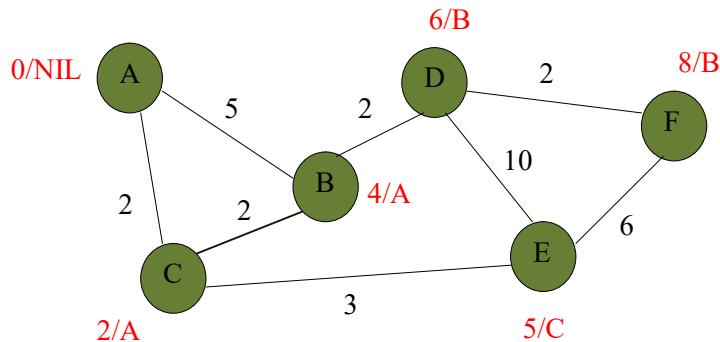
Priority queue: F



NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

Thuật toán Dijkstra: Có định nhãn đỉnh F

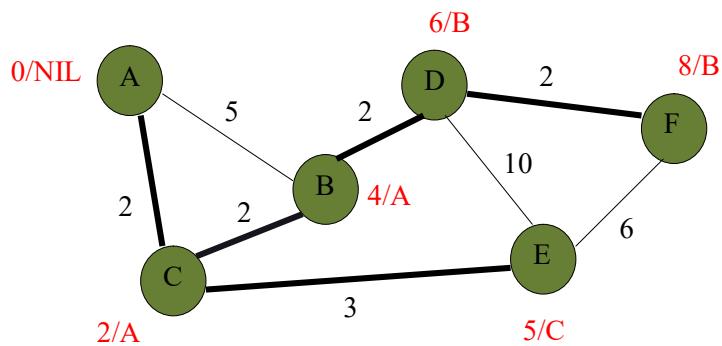
Priority queue:



NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

Kết quả

Cây đường đi ngắn nhất xuất phát từ A:



NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-DHBK HN

Cài đặt Dijkstra theo 2 cách

```
#include<bits/stdc++.h>
using namespace std;
# define INF 0x3f3f3f3f

typedef pair<int, int> iPair;

class Graph
{
    int V; // No. of vertices
    // In a weighted graph, we need to store vertex and weight pair for every edge
    list< pair<int, int> > *adj;
    //vector <pair<int, int> > *adj1;

public:
    Graph(int V); // Constructor
    ~Graph() { delete [] adj; } // To avoid memory leak
    //Them canh (u, v) co trong so = weight vao do thi:
    void addEdge(int u, int v, int weight);

    //Thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh s đến tất cả các đỉnh con lâm của đồ thị
    void Dijkstra(int s);

    //Thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh s đến tất cả các đỉnh con lâm của đồ thị
    void Dijkstra_PriorityQueue(int s);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<iPair> [V];
    //adj1 = new vector <iPair> [V];
}

void Graph::addEdge(int u, int v, int weight)
{
    adj[u].push_back(make_pair(v, weight));
    adj[v].push_back(make_pair(u, weight)); //đo thị vô hướng
}
```

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-DHBK HN

```

//Dijkstra cai dat truc tiep
void Graph::Dijkstra(int s)
{
    vector<int> d(V, INF);
    vector<int> p(V, s);
    vector <bool> k(V, 0);
    d[s] = 0;

    int count=1;
    int mindistance,i,j,u;
    while(count < V-1)
    {
        mindistance = INF;
        for(i=0; i<V; i++) //Tim dinh u co d[u] min
            if(!k[i] && d[i]<mindistance)
            {
                mindistance = d[i];
                u=i;
            }
        k[u]=1;
        list< pair<int, int> >::iterator i;
        //Duyet qua danh sach ke cua dinh u:
        for (i = adj[u].begin(); i != adj[u].end(); ++i)
        {
            // Get vertex label and weight of current adjacent of u.
            int v = (*i).first;
            int weight = (*i).second; //trong so tren cung (u, v)

```

99

```

//Goi Relax(u, v):
if (d[v] > d[u] + weight)
{
    d[v] = d[u] + weight; //cap nhat d[v]
    p[v] = u;
}
count++;
}

cout<<" Dinh      Do dai DDNN tu dinh "<<s<<"      DDNN \n";
for(i=0; i<V; i++)
    if(i != s)
    {
        cout<<i<<" \t \t "<<d[i]<<"\t\t\t";
        cout<<i;
        j=i;
        do {
            j=p[j];
            cout<<" <- "<<j;
        }while(j != s);
        cout<<endl;
    }
}

```

100

```

//Dijkstra cai dat priority queue:
void Graph::Dijkstra_PriorityQueue(int s)
{
    /* Create a priority queue to store vertices*/
    priority_queue< iPair, vector <iPair> , greater<iPair> > Q;
    vector<int> d(V, INF);
    vector<int> p(V, s);
    vector <bool> k(V,0);

    // Insert source itself in priority queue and initialize its distance as 0.
    d[s] = 0; Q.push(make_pair(d[s], s));

    /* Looping till priority queue becomes empty (or all
       distances are not finalized) */
    while (!Q.empty())
    {
        int u = Q.top().second; //dinh u co d[u] min
        int temp = Q.top().first;
        Q.pop();

        if (!k[u])
        {
            k[u] = true;
            list< pair<int, int> ::iterator i;
            for (i = adj[u].begin(); i != adj[u].end(); ++i)
            {
                // Get vertex label and weight of current adjacent of u.
                int v = (*i).first;
                int weight = (*i).second; //trong so tren cung (u, v)
                if (d[v] > d[u] + weight)
                {
                    d[v] = d[u] + weight;
                    Q.push(make_pair(d[v], v));
                    p[v]=u;
                }
            }
        }
    }
}

```

1

```

//Goi Relax(u, v):
if (d[v] > d[u] + weight)
{
    d[v] = d[u] + weight;
    Q.push(make_pair(d[v], v));
    p[v]=u;
}

int i, j;
cout<<" Dinh      Do dai DDNN tu dinh "<<s<<"      DDNN \n";
for(i=0; i<V; i++)
{
    if(i != s)
    {
        cout<<i<<" \t \t "<<d[i]<<"\t\t\t";
        cout<<i;
        int j=i;
        do {
            j=p[j];
            cout<<" <- "<<j;
        }while(j != s);
        cout<<endl;
    }
}

```

102

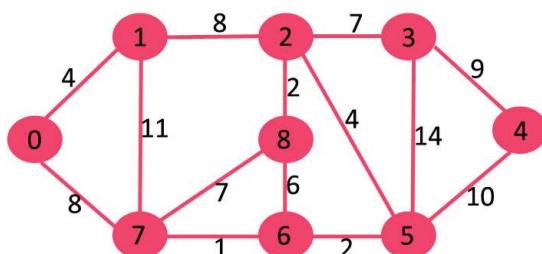
```

int main()
{
    int V = 9;
    Graph g(V);
    g.addEdge(0, 1, 4); g.addEdge(0, 7, 8);
    g.addEdge(1, 2, 8); g.addEdge(1, 7, 11);
    g.addEdge(2, 3, 7); g.addEdge(2, 8, 2);
    g.addEdge(2, 5, 4); g.addEdge(3, 4, 9);
    g.addEdge(3, 5, 14); g.addEdge(4, 5, 10);
    g.addEdge(5, 6, 2); g.addEdge(6, 7, 1);
    g.addEdge(6, 8, 6); g.addEdge(7, 8, 7);

    cout<<"__ KET QUA CAI DAT TRUC TIEP THUAT TOAN DIJKSTRA_____ "<<endl;
    g.Dijkstra(0);
    cout<<"__ KET QUA CAI DAT DIJKSTRA DUNG PRIORITY QUEUE _____ "<<endl;
    g.Dijkstra_PriorityQueue(0);

    return 0;
}

```



NGUYỄN KHÁNH PHƯƠNG 103
KHMT – SOICT-ĐHBK HN

3.8. Thư viện STL trong C++: map

`#include <map>` map có dữ liệu kiểu cấu trúc liên kết, mỗi phần tử của map gồm 2 thành phần: khóa (key value) và ánh xạ của nó (mapped value). Các phần tử trong map không chứa khóa giống nhau. Các phần tử trong map được sắp xếp theo thứ tự tăng dần của giá trị key.

Khai báo:

`map <key_type, value_type> mapName;` //khai báo mapName trong đó mỗi phần tử có trường key có kiểu key_type, trường value có kiểu value_type

Ví dụ: Khai báo map có keys là xâu kí tự, value là kiểu số nguyên:

```

map <string, int> map1;
map1["Thang"] = 8; //Thêm phần tử có key = "Thang", value = 8
map1["Binh"] = 10; //Thêm phần tử có key = "Binh", value = 10
map1["Phuong"] = 7; //Thêm phần tử có key = "Phuong", value = 7

```

NGUYỄN KHÁNH PHƯƠNG 104
KHMT – SOICT-ĐHBK HN

3.8. Thư viện STL trong C++: map

```
#include<bits/stdc++.h>
using namespace std;

int main ()
{
    map <string,int> map1;
    map1["Thang"] = 8;
    map1["Binh"] = 10;
    map1["Phuong"] = 7;
    //In map1:
    for (map<string,int>::iterator it = map1.begin(); it != map1.end(); it++)
        cout << it->first << ":" << it->second << endl;
}
```

Binh: 10
Phuong: 7
Thang: 8

	begin
Binh	10
Phuong	7
Thang	8

NGUYỄN KHÁNH PHƯƠNG 105
KHMT – SOICT-ĐHBK HN

3.8. Thư viện STL trong C++: map

```
#include<bits/stdc++.h>
using namespace std;

int main ()
{
    map <string,int> map1;
    map1["Thang"] = 8;
    map1["Binh"] = 10;
    map1["Phuong"] = 7;
    map1["Binh"] = 9;
    //In map1:
    for (map<string,int>::iterator it = map1.begin(); it != map1.end(); it++)
        cout << it->first << ":" << it->second << endl;
}
```

? Kết quả in ra màn hình

NGUYỄN KHÁNH PHƯƠNG 106
KHMT – SOICT-ĐHBK HN

3.8. Thư viện STL trong C++: map

Capacity	
size()	Trả về số lượng phần tử có trong map
empty()	True/False
Truy cập phần tử	
begin()	Trả về iterator trả tới phần tử đầu tiên của map
end()	Trả về iterator trả tới vị trí sau phần tử cuối cùng của map
Truy vấn	
insert(x)	Chèn phần tử x (gồm key và value) vào map (Chú ý: nếu key của x đã có trong map, thì không chèn x vào map)
insert (iterator start, iterator end)	Chèn các phần tử trả từ trả từ iterator pos đến end của một map khác vào map
insert_or_assign()	Chức năng giống như insert, nhưng khác là: nếu key của phần tử muốn thêm vào map đã có trong map, thì value của nó sẽ bị thay đổi bởi value của phần tử muốn thêm
erase(iterator pos)	Xóa phần tử trả bởi iterator pos ra khỏi map
erase(iterator start, iterator end)	Xóa các phần tử trả từ iterator pos đến end ra khỏi map
clear()	Xóa tất cả các phần tử khỏi map

3.8. Thư viện STL trong C++: map

Truy vấn	
swap(map1)	Hoán đổi các phần tử của hai map cho nhau
find(key)	Trả về iterator trả đến phần tử cần tìm kiếm có khóa = key. Nếu không tìm thấy thì trả về end của map

Bài tập

You are given two lists of integers. You can remove any number of elements from any of them. You have to ensure that after removing some elements both of the lists will contain same elements, but not necessarily in same order. For example consider the following two lists

List #1	1 2 3 2 1
List #2	1 2 5 2 3

After removing 1 from first list and 5 from second list, both lists will contain same elements. We will find the following lists after removing two elements.

List #1	1 2 3 2
List #2	1 2 2 3

What is the minimum number of elements to be removed to obtain two lists having same elements?

Input

The first line of the input file contains an integer T ($T \leq 100$) which denotes the total number of test cases. The description of each test case is given below:

First line will contain two integers N ($1 \leq N \leq 10000$), the number of elements in the first list and M ($1 \leq M \leq 10000$), the number of elements in the second list. The next line will contain N integers representing the first list followed by another line having M integers representing the second list. Each integer in the input is 32 bit signed integer.

Output

For each test case output a single line containing the number of elements to be removed. See sample output for exact format.

Sample Input

```
1
5 5
1 2 3 2 1
1 2 5 2 3
```

Sample Output

```
2
```

109

Bài tập

```
#include <stdio.h>
#include <stdlib.h>
#include <map>
using namespace std;
int main() {
    int numTest, count=0, n, m, x;

    scanf("%d", &numTest);
    while (count != numTest)
    {
        scanf("%d %d", &n, &m);
        map<int, int> r; /*map r co key la cac gia tri so nguyen x trong mang,
                           value la so lan xuat hien x trong mang */
        while(n--)
            scanf("%d", &x), r[x]++;
        while(m--)
            scanf("%d", &x), r[x]--;

        int sopheptinh = 0; // tong value cua tat ca cac phan tu trong map
        for(map<int,int>::iterator it = r.begin(); it != r.end(); it++)
            sopheptinh += abs(it->second);
        printf("%d\n", sopheptinh);
        count++;
    }
}
```

110

3. Thư viện cấu trúc dữ liệu và thuật toán

Một số hàm sắp xếp và tìm kiếm thông dụng trong `<#include algorithm>`

- Sắp xếp một mảng
- Tìm kiếm trên một mảng chưa sắp xếp
- Tìm kiếm trên một mảng đã sắp xếp
- Tìm min, max giữa hai phần tử
- Tìm hoán vị kế tiếp sau/trước

NGUYỄN KHÁNH PHƯƠNG 111
KHMT – SOICT-ĐHBK HN

3. Thư viện cấu trúc dữ liệu và thuật toán

- Sắp xếp một mảng

```
void sort(InputIterator first, InputIterator last, Compare comp);

#include<iostream>
#include<algorithm>
using namespace std;
int main()
{
    int demo[5] = {5, 4, 3, 2, 1};
    int len = sizeof(demo)/sizeof(demo[0]);

    //Sorting demo array in ascending order:
    std::sort(demo, demo + len);
    //Sorting demo array in descending order:
    std::sort(demo, demo + len, greater<int>());
}
```

NGUYỄN KHÁNH PHƯƠNG 112
KHMT – SOICT-ĐHBK HN

3. Thư viện cấu trúc dữ liệu và thuật toán

- Sắp xếp 1 mảng

```
void sort(InputIterator first, InputIterator last, Compare comp);  
#include<iostream>  
#include<algorithm>  
using namespace std;  
//our function  
bool My_func( int a, int b)  
{  
    return (a%10)<(b%10);  
}  
int main()  
{  
    int demo[5] = {18, 45, 34, 92, 21};  
    int len = sizeof(demo)/sizeof(demo[0]);  
    cout<<"Before sorting array : "  
    for(int i=0; i<len; i++) cout<<" "<<demo[i];  
  
    std::sort(demo, demo + len, My_func); //Sorting demo array  
  
    cout<<"\n\nAfter sorting array : "  
    for(int i=0; i<len; i++) cout<<" "<<demo[i];  
    return 0;  
}
```

output?

NGUYỄN KHÁNH PHƯƠNG 113
KHMT – SOICT-ĐHBKHN

3. Thư viện cấu trúc dữ liệu và thuật toán

- Tìm kiếm trên một mảng chưa sắp xếp

```
template <class InputIterator, class T>  
InputIterator find (InputIterator first, InputIterator last,  
                   const T& val);
```

Trả về chỉ số của phần tử đầu tiên trong [first, last) nếu nó bằng với giá trị val. Nếu không tìm được phần tử nào, hàm trả về giá trị last.

NGUYỄN KHÁNH PHƯƠNG 114
KHMT – SOICT-ĐHBKHN

Ví dụ: hàm find

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std ;

int main () {
    //1. Tìm phần tử có giá trị 30 trong mảng
    int myints[] = { 40, 30, 10, 20 };
    int *p;

    p = find (myints, myints+4, 30);
    if (p != myints+4)
        cout << "Giá trị 30 xuất hiện trong mảng myints tại phần tử thứ: " << (p-myints+1) << '\n';
    else
        cout << "Giá trị 30 không xuất hiện trong mảng myints\n";

    //2. Tìm phần tử có giá trị 30 trong vector:
    vector<int> myvector (myints,myints+4);
    vector<int>::iterator it;

    it = find (myvector.begin(), myvector.end(), 30);
    if (it != myvector.end())
        cout << "Giá trị 30 xuất hiện trong vector myvector tại phần tử thứ: " << (it - myvector.begin()+1) << '\n';
    else
        cout << "Giá trị 30 không xuất hiện trong vector myvector\n";

    return 0;
}

Gia tri 30 xuat hien trong mang myints tai phan tu thu: 2
Gia tri 30 xuat hien trong vector myvector tai phan tu thu: 2
```

NGUYỄN KHÁNH PHƯƠNG 115
KHMT – SOICT-ĐHBK HN

3. Thư viện cấu trúc dữ liệu và thuật toán

- Tìm kiếm trên mảng chưa sắp xếp theo điều kiện

InputIterator **find_if** (InputIterator first,
InputIterator last, UnaryPredicate pred);

Trả về chỉ số của phần tử đầu tiên trong [first, last) thỏa mãn điều kiện pred. Nếu không tìm được phần tử nào, hàm trả về giá trị last.

Ngoài ra, còn có các hàm:

- **find_if_not**
- **find_end**
- **find_first_of**
- **adjacent_find**

NGUYỄN KHÁNH PHƯƠNG 116
KHMT – SOICT-ĐHBK HN

Ví dụ: hàm find_if

```
#include <bits/stdc++.h>
using namespace std;

// Returns true if argument is odd
bool IsOdd(int i) {return i % 2; }

int main()
{
    int myArray[4] = {50, 32, 27, 25} ;
    int *p = find_if(myArray, myArray+4, IsOdd);
    if (p != myArray+4)
        cout << "Phan tu co gia tri le dau tien trong mang la " << *p <<"; tai vi tri thu "<<(p-myArray+1)<<'\n';
    else
        cout<<"Mang myArray ko co gia tri le\n";
    // Iterator to store the position of element found
    vector<int> vec;
    //Copy cac phan tu cua myArray sang vector vec:
    vec.insert(vec.begin(), myArray, myArray +4 );
    vector<int>::iterator it;
    it = find_if(vec.begin(), vec.end(), IsOdd);
    if (it != vec.end())
        cout << "Phan tu co gia tri le dau tien trong vector la " << *it <<"; tai vi tri thu "<<(it-vec.begin())+1)<< '\n';
    else
        cout<<"Vector vec ko co gia tri le\n";
    return 0;
}
```

NGUYỄN KHÁNH PHƯƠNG 117
KHMT – SOICT-ĐHBK HN

3. Thư viện cấu trúc dữ liệu và thuật toán

- Tìm kiếm trên mảng chưa sắp xếp

```
template <class ForwardIterator>
ForwardIterator min_element (ForwardIterator first,
ForwardIterator last);
```

Trả về chỉ số của phần tử có giá trị nhỏ nhất trong [first, last].

```
template <class ForwardIterator>
ForwardIterator max_element (ForwardIterator first,
ForwardIterator last);
```

Trả về chỉ số của phần tử có giá trị lớn nhất trong [first, last].

NGUYỄN KHÁNH PHƯƠNG 118
KHMT – SOICT-ĐHBK HN

Ví dụ 1: hàm max_element, min_element

```
#include<bits/stdc++.h>
using namespace std;

bool myfn(int i, int j) { return i<j; }

struct myclass {
    bool operator() (int i,int j) { return i<j; }
} myobj;

int main () {
    int myints[] = {3,7,2,5,6,4,9};

    // using default comparison:
    cout << "The smallest element is " << *min_element(myints,myints+7) << '\n';
    cout << "The largest element is " << *max_element(myints,myints+7) << '\n';

    // using function myfn as comp:
    cout << "The smallest element is " << *min_element(myints,myints+7,myfn) << '\n';
    cout << "The largest element is " << *max_element(myints,myints+7,myfn) << '\n';

    // using object myobj as comp:
    cout << "The smallest element is " << *min_element(myints,myints+7,myobj) << '\n';
    cout << "The largest element is " << *max_element(myints,myints+7,myobj) << '\n';

    return 0;
}
```

NGUYỄN KHÁNH PHƯƠNG |19
KHMT – SOICT-DHBK HN

Ví dụ 2: hàm max_element, min_element

```
#include<bits/stdc++.h>
using namespace std;

bool myfn(int i, int j) { return i<j; }

struct myclass {
    bool operator() (int i,int j) { return i<j; }
} myobj;

int main () {
    int myints[] = {1,2,3,4,5,4,3,2,1};
    vector<int> v(myints,myints+9);

    // using default comparison:
    vector<int>::iterator itmin = min_element(v.begin(),v.end());
    cout << "The smallest element is " << *itmin << '\n';
    vector<int>::iterator itmax = max_element(v.begin(),v.end());
    cout << "The largest element is " << *itmax << '\n';

    // using function myfn as comp:
    cout << "The smallest element is " << *min_element(v.begin(), v.end(),myfn) << '\n';
    cout << "The largest element is " << *max_element(v.begin(), v.end(),myfn) << '\n';

    // using object myobj as comp:
    cout << "The smallest element is " << *min_element(v.begin(),v.end(),myobj) << '\n';
    cout << "The largest element is " << *max_element(v.begin(),v.end(),myobj) << '\n';
    return 0;
}
```

NGUYỄN KHÁNH PHƯƠNG |120
KHMT – SOICT-DHBK HN

3. Thư viện cấu trúc dữ liệu và thuật toán

- Tìm kiếm trên một mảng đã sắp xếp

```
template <class ForwardIterator, class T>
    ForwardIterator lower_bound (ForwardIterator first,
    ForwardIterator last, const T& val);
Trả về iterator trả đến phần tử đầu tiên trong [first, last) không nhỏ hơn
val. Nếu tất cả các phần tử nhỏ hơn val, trả về last
```

```
template <class ForwardIterator, class T>
    ForwardIterator upper_bound (ForwardIterator first,
    ForwardIterator last, const T& val);
Trả về iterator trả đến phần tử đầu tiên trong [first, last) lớn hơn val.
Nếu không tồn tại, trả về last
```

```
template <class ForwardIterator, class T, class Compare> bool binary_search (ForwardIterator first,
    ForwardIterator last, const T& val);
Trả về true nếu val xuất hiện trong mảng
```

121

Ví dụ: hàm lower_bound, upper_bound

```
#include<bits/stdc++.h>
using namespace std;

int main () {
    int myints[] = {10,20,30,30,20,10,10,20};
    vector<int> v(myints,myints+8);           // 10 20 30 30 20 10 10 20
    sort (v.begin(), v.end());                // sorted arr: 10 10 10 20 20 20 20 30 30

    vector<int>::iterator low,up;
    low=lower_bound (v.begin(), v.end(), 20); //          ^
    up= upper_bound (v.begin(), v.end(), 20); //          ^
    cout << "lower_bound (phan tu dau tien trong day >=20) tai phan tu thu " << (low- v.begin())+1 << '\n';
    cout << "upper_bound (phan tu dau tien trong day > 20) tai phan tu thu " << (up - v.begin())+1 << '\n';

    return 0;
}

lower_bound (phan tu dau tien trong day >=20) tai phan tu thu 4
upper_bound (phan tu dau tien trong day > 20) tai phan tu thu 7
```

3. Thư viện cấu trúc dữ liệu và thuật toán

- Tìm kiếm trên một mảng đã sắp xếp

(1) template <class ForwardIterator, class T, class Compare> bool binary_search (ForwardIterator first, ForwardIterator last, const T& val);

(2) template <class ForwardIterator, class T, class Compare> bool binary_search (ForwardIterator first, ForwardIterator last, const T& val, Compare comp);

Trả về true nếu val xuất hiện trong mảng

Khi so sánh hai giá trị a và b:

Ở dạng (1): hàm so sánh “<” được mặc định sử dụng; tức là a==b nếu (! (a<b) && ! (b<a))

Ở dạng (2): hàm so sánh comp được sử dụng; tức là a==b nếu (! comp (a,b) && ! comp (b,a)).

NGUYỄN KHÁNH PHƯƠNG 123
KHMT – SOICT-ĐHBK HN

Ví dụ: hàm binary_search

```
#include<bits/stdc++.h>
using namespace std;

bool myfunction (int i,int j) { return (i<j); }

int main () {
    int myints[] = {1,2,3,4,5,4,3,2,1};
    vector<int> v(myints,myints+9); // 1 2 3 4 5 4 3 2 1

    // using default comparison:
    sort (v.begin(), v.end());

    cout << "Tim gia tri 3 có xuất hiện trong vector v hay không... ";
    if (binary_search (v.begin(), v.end(), 3, myfunction))
        cout << "found!\n";
    else cout << "not found.\n";

    // using myfunction as comp:
    sort (v.begin(), v.end(), myfunction);

    cout << "Tim gia tri 6 có xuất hiện trong vector v hay không... ";
    if (binary_search (v.begin(), v.end(), 6, myfunction))
        cout << "found!\n";
    else cout << "not found.\n";

    return 0;
}
```

NGUYỄN KHÁNH PHƯƠNG 124
KHMT – SOICT-ĐHBK HN

Nội dung chương 1

1. Các kiểu dữ liệu cơ bản
2. Xử lý số nguyên lớn
3. Thư viện cấu trúc dữ liệu và thuật toán
- 4. Biểu diễn tập hợp bằng Bit mask**
5. Biểu diễn đồ thị
6. Cấu trúc dữ liệu mở

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - ĐHBK HN

4. Biểu diễn tập hợp bằng bit mask

Cho tập S gồm n phần tử (n nhỏ, $n \leq 64$). Ta sẽ đánh thứ tự các phần tử trong tập S từ 0, 1, .. $n-1$.

- Khi đó, ta có thể biểu diễn mỗi phần tử trong tập S bởi một số nguyên 64-bit.
- Phần tử thứ i của tập S được biểu diễn bởi số nguyên X nếu bit thứ i của X là 1

Ví dụ: Cho tập S gồm 5 bạn gái {Lan, Lê, Lành, Linh, Lam}, các bạn gái này được đánh số theo thứ tự 0, 1, 2, 3, 4 \rightarrow Tập S được biểu diễn bởi số nguyên 31 = $11111_{(2)}$

- Tập S_1 gồm 3 bạn gái {Lan, Lành, Linh} sẽ tương ứng với phần tử thứ 0, 2, 3 của tập S, tức là $S_1 = \{0, 2, 3\}$ là tập con của tập S. Tập S_1 này có thể được biểu diễn bởi số nguyên x = 13 vì

$$x = (1 \ll 0) | (1 \ll 2) | (1 \ll 3) = 1 | 4 | 8 = 01101_{(2)} = 13$$

00001 Có phần tử thứ 0 $\rightarrow 1 \ll 0 = 00001$

OR 00100 Có phần tử thứ 2 $\rightarrow 1 \ll 2 = 00100$

01000 Có phần tử thứ 3 $\rightarrow 1 \ll 3 = 01000$

01101 Có phần tử thứ 0, 2, 3

Left shift: $x \ll y = x \cdot 2^y$

Right shift: $x \gg y = x / 2^y$

126

4. Biểu diễn tập hợp bằng bit mask

Xét tập vũ trụ N gồm n phần tử và tập con S của N:

- Bổ sung thêm phần tử thứ j của tập N vào tập S:
 - Thực hiện $S |= (1 << j)$
- Loại phần tử thứ j của tập N khỏi tập S:
 - Thực hiện $S &= \sim(1 << j)$
- Thay đổi trạng thái của phần tử thứ j của tập N trong tập S (tức là, nếu j đang có trong S thì loại j , nếu j không có trong S thì bổ sung j vào S):
 - Thực hiện $S ^= (1 << j)$
- Kiểm tra xem phần tử thứ j của tập N có trong S hay không:
 - Thực hiện $T = S \& (1 << j)$
 - Nếu $T = 0$: phần tử thứ j không có trong S
 - Nếu $T \neq 0$: phần tử thứ j có trong S

AND (&)	OR ()	XOR (^)	NOT (~)
a b a&b 0 0 0 0 1 0 1 0 0 1 1 1	a b a b 0 0 0 0 1 1 1 0 1 1 1 1	a b a^b 0 0 0 0 1 1 1 0 1 1 1 0	a ~a 0 1 1 0

NGUYỄN KHÁNH PHƯƠNG 127
KHMT – SOICT-ĐHBK HN

4. Biểu diễn tập hợp bằng bit mask

Xét tập vũ trụ N gồm n phần tử và tập con S của N:

- Tìm phần tử thứ j có thứ tự nhỏ nhất trong tập N có trong tập S:
 - Thực hiện $T = (S \& (-S))$

$$\begin{aligned}
 S &= 40 \text{ (base 10)} = 000\dots000101000 \text{ (32 bits, base 2)} \\
 -S &= -40 \text{ (base 10)} = 111\dots111011000 \text{ (two's complement)} \\
 &\quad \text{----- AND} \\
 T &= 8 \text{ (base 10)} = 000\dots000001000 \text{ bit thứ 3 từ phải sang = 1} \Rightarrow j = 3
 \end{aligned}$$

AND (&)	OR ()	XOR (^)	NOT (~)
a b a&b 0 0 0 0 1 0 1 0 0 1 1 1	a b a b 0 0 0 0 1 1 1 0 1 1 1 1	a b a^b 0 0 0 0 1 1 1 0 1 1 1 0	a ~a 0 1 1 0

NGUYỄN KHÁNH PHƯƠNG 128
KHMT – SOICT-ĐHBK HN

4. Biểu diễn tập hợp bằng bit mask

Xét tập vũ trụ N gồm n phần tử và các tập con của N:

Tập hợp	Số nguyên
Tập rỗng	0
Tập con có một phần tử thứ i	$1 \ll i$
Tập vũ trụ N có tất cả n phần tử	$(1 \ll n) - 1$
Hợp của S1 và S2 với:	$x y$
• Tập con S1 biểu diễn bởi số nguyên x, • Tập con S2 biểu diễn bởi số nguyên y.	
Giao của S1 và S2 với:	$x \& y$
• Tập con S1 biểu diễn bởi số nguyên x, • Tập con S2 biểu diễn bởi số nguyên y.	
Phần bù của tập S1	$\sim x \& ((1 \ll n) - 1)$

AND (&)	OR ()	XOR (^)	NOT (~)																																																			
<table border="1"> <tr><td>a</td><td>b</td><td>a&b</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	a	b	a&b	0	0	0	0	1	0	1	0	0	1	1	1	<table border="1"> <tr><td>a</td><td>b</td><td>a b</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	a	b	a b	0	0	0	0	1	1	1	0	1	1	1	1	<table border="1"> <tr><td>a</td><td>b</td><td>a^b</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </table>	a	b	a^b	0	0	0	0	1	1	1	0	1	1	1	0	<table border="1"> <tr><td>a</td><td></td></tr> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </table>	a		0	1	1	0
a	b	a&b																																																				
0	0	0																																																				
0	1	0																																																				
1	0	0																																																				
1	1	1																																																				
a	b	a b																																																				
0	0	0																																																				
0	1	1																																																				
1	0	1																																																				
1	1	1																																																				
a	b	a^b																																																				
0	0	0																																																				
0	1	1																																																				
1	0	1																																																				
1	1	0																																																				
a																																																						
0	1																																																					
1	0																																																					

NGUYỄN KHÁNH PHƯƠNG 129
KHMT – SOICT-ĐHBK HN

4. Biểu diễn tập hợp bằng bit mask

Tại sao nên dùng bit mask mà không dùng `set <int>` ?

- Tiết kiệm bộ nhớ
 - Tất cả các tập con của tập n phần tử N có thể biểu diễn bởi các số nguyên trong khoảng $0 .. 2^n - 1$.
 - Dễ dàng duyệt tất cả các tập con của N:
- ```
for (x = 0; x < (1 << n); ++x)
```
- Dễ dàng duyệt tất cả các tập con của tập con S (không bao gồm tập rỗng) biểu diễn bởi số nguyên y:
- ```
for ( x = y; x > 0; x = ( y & (x-1) ) )
```
- Dễ dàng sử dụng 1 tập hợp như một chỉ số của một mảng

NGUYỄN KHÁNH PHƯƠNG 130
KHMT – SOICT-ĐHBK HN

Ví dụ

Cho tập S gồm các số nguyên. Hãy đếm số tập con của S mà tổng các số trong mỗi tập con này bằng số nguyên k cho trước.

Nếu $|S|$ nhỏ ta có thể dùng bitmask để duyệt qua tất cả các tập con của S và tính tổng các số trong mỗi tập con xem có bằng k hay không:

```
mask = 0; //tuong ung voi tap rong
soluong = 0;
while (mask < (1 << |S|)) //duyet qua lan Luot tung tap con cua S
{
    sum = 0;
    for (i=0; i <= |S|-1; i++) //duyet qua lan Luot tung phan tu cua tap S
        if (mask & (1<<i)) //check phan tu thu i cua S co trong tap con hay ko
            sum += S[i];
        if (sum == K) soluong++;
    mask += 1;
}
```

Độ phức tạp: $O(2^{|S|} \cdot |S|)$

NGUYỄN KHÁNH PHƯƠNG 131
KHMT – SOICT - ĐHBK HN

Ví dụ

Có N công việc (0,1,..,N-1) cần phân cho N người (0,1,..,N-1), mỗi công việc chỉ được thực hiện bởi 1 người, mỗi người chỉ thực hiện 1 công việc. Cho trước ma trận chi phí kích thước NxN với $cost[i][j]$ là chi phí phải trả cho người thứ i nếu người này thực hiện công việc j . Hãy thực hiện phân công công việc cho mỗi người sao cho tổng chi phí phải trả cho N người này là nhỏ nhất.

Duyệt toàn bộ: có $N!$ cách gán N công việc cho N người thỏa mãn yêu cầu → thời gian tính $O(N!)$

```
for i = 0 to N-1
    assignment[i] = i //gan cong viec i cho nguoi i
result = INFINITY
for j = 1 to N! //duyet lan Luot N! hoan vi
{
    total_cost = 0
    for i = 0 to N-1
        total_cost += cost[i][assignment[i]];
    result = min(res, total_cost);
    generate_next_permutation(assignment);
}
return result;
```

NGUYỄN KHÁNH PHƯƠNG 132
KHMT – SOICT - ĐHBK HN

Một số ứng dụng các cấu trúc dữ liệu

Một số ứng dụng của Mảng và Danh sách liên kết:

- Bài toán cần lưu trữ dữ liệu, có nhiều phần tử cần liệt kê
- Ví dụ: Bài toán Broken keyboard:
<https://onlinejudge.org/external/119/p11988.pdf>

Một số ứng dụng của ngăn xếp:

- Xử lý các sự kiện theo trình tự vào sau – ra trước
- Khử đệ quy
- Tìm kiếm theo chiều sâu trên đồ thị
- Đảo ngược xâu
- Kiểm tra dãy ngoặc trong biểu thức / source code

Một số ứng dụng của hàng đợi:

- Xử lý các sự kiện theo trình tự vào trước – ra trước
- Tìm kiếm theo chiều rộng trên đồ thị

NGUYỄN KHÁNH PHƯƠNG 133
KHMT – SOICT - DHBK HN

Một số ứng dụng các cấu trúc dữ liệu

Một số ứng dụng của hàng đợi ưu tiên:

- Xử lý các sự kiện theo trình tự ưu tiên
- Tìm đường đi ngắn nhất trên đồ thị
- Một số thuật toán tham lam

Một số ứng dụng của tập hợp:

- Giữ vết của các phần tử phân biệt
- Nếu cài đặt như một cây nhị phân tìm kiếm:
 - Tìm cha của một phần tử (phần tử nhỏ nhất mà lớn hơn nó)
 - Đếm xem có bao nhiêu phần tử nhỏ hơn một phần tử cho trước
 - Đếm xem có bao nhiêu phần tử nằm giữa hai phần tử cho trước
 - Tìm phần tử lớn thứ k

Một số ứng dụng của kiểu ánh xạ:

- Gắn một giá trị với một khóa
- Giống như bảng tần suất
- Giống như phần lưu trữ khi thực hiện thuật toán quy hoạch động

134

Nội dung chương 1

1. Các kiểu dữ liệu cơ bản
2. Xử lý số nguyên lớn
3. Thư viện cấu trúc dữ liệu và thuật toán
4. Biểu diễn tập hợp bằng Bit mask
- 5. Biểu diễn đồ thị**
6. Cấu trúc dữ liệu mở

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

5. Biểu diễn đồ thị

Có nhiều dạng đồ thị:

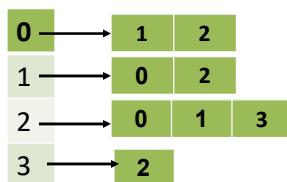
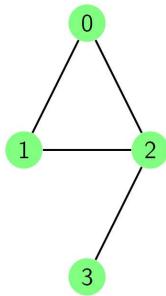
- Có hướng hoặc vô hướng
- Có trọng số hoặc Không có trọng số
- Đơn đồ thị hoặc Đa đồ thị

Một số cách biểu diễn đồ thị thông dụng:

- Danh sách kè (Adjacency lists)
- Ma trận kè (Adjacency matrix)
- Danh sách cạnh (Edge lists)

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

5.1. Biểu diễn đồ thị bởi danh sách kề

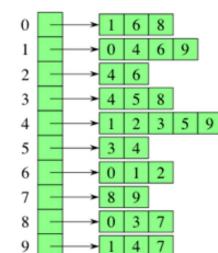
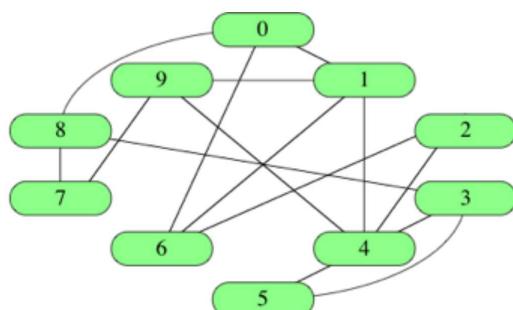


```
vector<int> adj[4];
adj[0].push_back(1);
adj[0].push_back(2);
adj[1].push_back(0);
adj[1].push_back(2);
adj[2].push_back(0);
adj[2].push_back(1);
adj[2].push_back(3);
adj[3].push_back(2);
```

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - ĐHBK HN

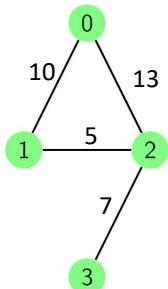
5.1. Biểu diễn đồ thị bởi danh sách kề

Bài tập: viết sourcecode biểu diễn đồ thị sau bởi danh sách kề



NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - ĐHBK HN

5.1. Biểu diễn đồ thị bởi danh sách kề

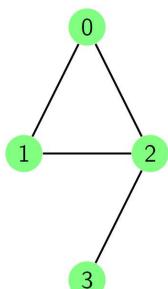


0	→	1	2
1	→	0	2
2	→	0	1
3	→	2	

```
vector <pair <int, int> > adj[4];
//hoặc list <pair <int, int> > adj[4];
adj[0].push_back(make_pair(1, 10));
adj[1].push_back(make_pair(0, 10)); //đồ thị vô hướng
adj[0].push_back(make_pair(2, 13));
adj[2].push_back(make_pair(0, 13)); //đồ thị vô hướng
adj[1].push_back(make_pair(2, 5));
adj[2].push_back(make_pair(1, 5)); //đồ thị vô hướng
adj[2].push_back(make_pair(3, 7));
adj[3].push_back(make_pair(2, 7)); //đồ thị vô hướng

typedef pair <int, int> iPair;
vector <pair <int, int> > *adj; //hoặc list <pair <int, int> > *adj;
adj = new vector <iPair> [4]; // hoặc adj = new list <iPair> [4];
```

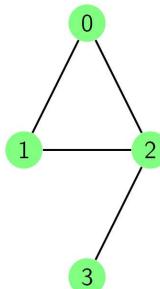
5.2. Biểu diễn đồ thị bởi ma trận kề



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

```
bool adj [4] [4];
adj [0] [1] = true;
adj [0] [2] = true;
adj [1] [0] = true;
adj [1] [2] = true;
adj [2] [0] = true;
adj [2] [1] = true;
adj [2] [3] = true;
adj [3] [2] = true;
```

5.3. Biểu diễn đồ thị bởi danh sách cạnh



(0, 1),
(0, 2),
(1, 2),
(2, 3)

```
vector<pair<int , int> > edges;
edges.push_back(make_pair(0, 1));
edges.push_back(make_pair(0, 2));
edges.push_back(make_pair(1, 2));
edges.push_back(make_pair(2, 3));
```

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

So sánh phân tích chi phí

	Danh sách kề	Ma trận kề	Danh sách cạnh
Bộ nhớ lưu trữ	$O(V + E)$	$O(V ^2)$	$O(E)$
Thêm đỉnh	$O(1)$	$O(V ^2)$	$O(1)$
Thêm cạnh	$O(1)$	$O(1)$	$O(1)$
Xóa đỉnh	$O(E)$	$O(V ^2)$	$O(E)$
Xóa cạnh	$O(E)$	$O(1)$	$O(E)$
Truy vấn: u, v có kề nhau không	$O(V)$	$O(1)$	$O(E)$

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

Nội dung chương 1

1. Các kiểu dữ liệu cơ bản
2. Xử lý số nguyên lớn
3. Thư viện cấu trúc dữ liệu và thuật toán
4. Biểu diễn tập hợp bằng Bit mask
5. Biểu diễn đồ thị
- 6. Cấu trúc dữ liệu mở**

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

6. Cấu trúc dữ liệu mở (Augmenting Data Structures)

Cấu trúc dữ liệu mở là cấu trúc dữ liệu sẵn có nào đó (ví dụ: danh sách liên kết đôi, cây nhị phân tìm kiếm..) được bổ sung/sửa đổi một số thông tin để có thể giải quyết các yêu cầu của bài toán được nhanh hơn.

Ví dụ: Cây nhị phân tìm kiếm mở (Augmenting Binary Search tree)

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT-ĐHBK HN

Ví dụ: Cây nhị phân tìm kiếm mở (Augmenting BST)

Cho cây nhị phân tìm kiếm, yêu cầu thực hiện:

- Tìm phần tử lớn thứ k trên cây
 - Đếm số lượng phần tử có giá trị $< x$ (hoặc xác định nút có giá trị x là phần tử lớn thứ bao nhiêu trên cây; hay nói cách khác: xác định **rank** của x)

Phương pháp trực tiếp: Duyệt qua tất cả các nút trên cây $O(n)$

→ Cải tiến: xây dựng cây nhị phân tìm kiếm mở, để hai thao tác trên mất $O(\log n)$

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- ĐHBK HN

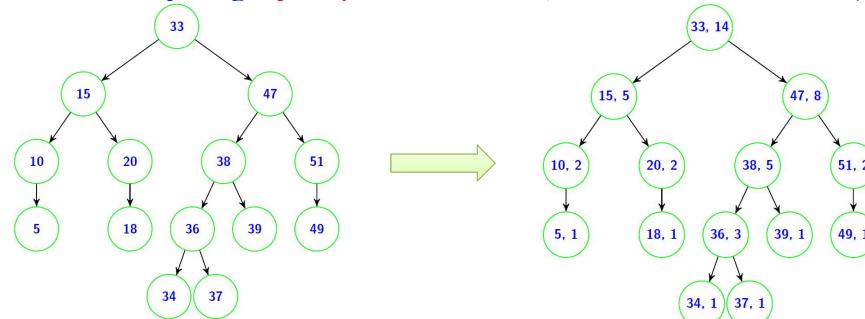
Ví dụ: Cây nhị phân tìm kiếm mở (Augmenting BST)

Cho cây nhị phân tìm kiếm, yêu cầu thực hiện:

- Tìm phần tử lớn thứ k trên cây
 - Đếm số lượng phần tử có giá trị $< x$ (hoặc xác định nút có giá trị x là phần tử lớn thứ bao nhiêu trên cây; hay nói cách khác: xác định **rank** của x)

Phương pháp trực tiếp: Duyệt qua tất cả các nút trên cây $O(n)$

→ Cải tiến: xây dựng cây nhị phân tìm kiếm mở, để hai thao tác trên mất $O(\log n)$

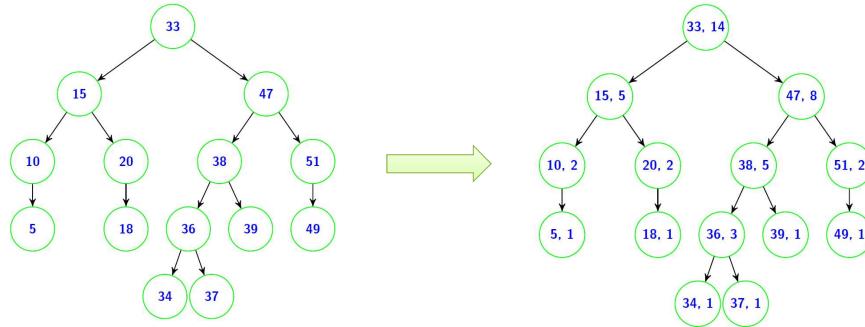


Tại mỗi nút, lưu thêm thông tin: số nút trên cây mà nó làm gốc

Ví dụ: Cây nhị phân tìm kiếm mở (Augmenting BST)

Cây nhị phân tìm kiếm mở:

- Tại mỗi nút, lưu thêm thông tin: số nút trên cây mà nó làm gốc
- Thông tin này sẽ được cập nhật mỗi khi thêm/xóa các nút trên cây mà không ảnh hưởng đến độ phức tạp chung của thuật toán



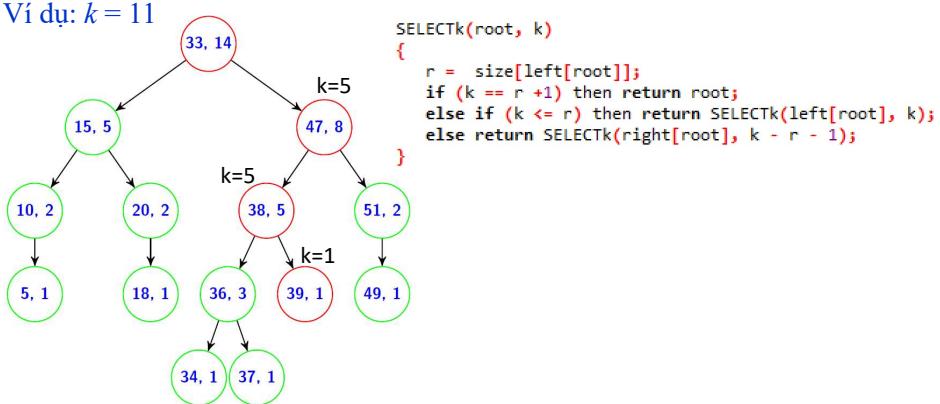
Tại mỗi nút, lưu thêm thông tin: số nút trên cây mà nó làm gốc

Ví dụ: Cây nhị phân tìm kiếm mở (Augmenting BST)

Câu hỏi 1: Tìm phần tử lớn thứ k trên cây có nút gốc trả bởi `root`:

- Xuất phát từ gốc. Tại mỗi nút mà cây con trái của nó có kích thước là r :
 - Nếu $k = r+1$, thì phần tử cần tìm chính là nút hiện tại
 - Nếu $k \leq r$, tiếp tục tìm phần tử lớn thứ k trong cây con trái
 - Nếu $k > r + 1$, tiếp tục tìm phần tử lớn thứ $k-r-1$ trong cây con phải

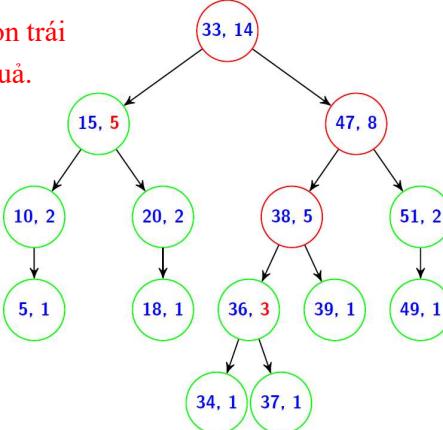
Ví dụ: $k = 11$



Ví dụ: Cây nhị phân tìm kiếm mở (Augmenting BST)

Câu hỏi 2: Đếm số lượng phần tử có giá trị < 38 trên cây có nút gốc trả bởi `root`:

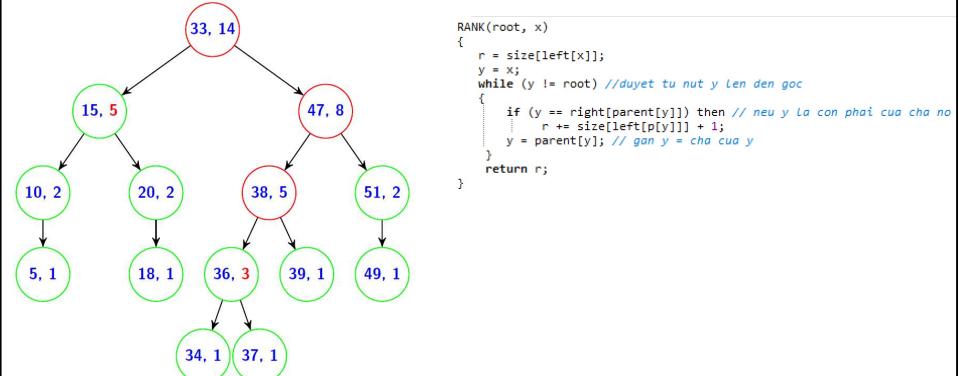
- Xuất phát từ gốc, tìm vị trí 38 trên cây:
 - Đếm số nút duyệt qua mà nhỏ hơn 38: khi duyệt đến một nút mà tiếp theo sẽ phải duyệt sang cây con phải của nó, lấy kích thước cây con trái + 1, rồi cộng vào biến đếm kết quả.
 - Gặp nút 38: lấy kích thước cây con trái của nó, rồi cộng vào biến đếm kết quả.



Ví dụ: Cây nhị phân tìm kiếm mở (Augmenting BST)

Câu hỏi 2 (cách hỏi khác): Cho nút trả bởi con trỏ x trên cây. Hãy xác định `rank(x)`

- Khởi tạo : biến đếm kết quả = kích thước cây con trái của nút x
- Xuất phát từ nút x : đặt $y = x$, duyệt ngược lên đến gốc:
 - khi duyệt đến một nút y mà y là con phải của cha nó, lấy kích thước cây con trái của cha nó cộng 1, và cộng vào biến đếm kết quả,



Bài tập 1

Given an array, your task is to find the k -th occurrence (from left to right) of an integer v . To make the problem more difficult (and interesting!), you'll have to answer m such queries.

Input

The first line of each test case contains two integers n, m ($1 \leq n, m \leq 100,000$), the number of elements in the array, and the number of queries. The next line contains n positive integers not larger than $1,000,000$. Each of the following m lines contains two integer k and v ($1 \leq k \leq n, 1 \leq v \leq 1,000,000$). The input is terminated by end-of-file (EOF).

Output

For each query, print the 1-based location of the occurrence. If there is no such element, output '0' instead.

Gợi ý: Dùng mảng hai chiều **vitrixuathien**:

Sample Input Với mỗi số nguyên v ($1 \leq v \leq 1,000,000$) lưu trữ các vị trí xuất hiện của v trong mảng đã cho:

8 4
1 3 2 2 4 3 2 1 **vitrixuathien[v][0..số lần xuất hiện v trong mảng]**

1 3

2 4

3 2

4 2

Ví dụ cho mảng như ở Sample Input, thì các giá trị của mảng là:

Vitrixuathien[1][0]=1; Vitrixuathien[1][1]=8

Vitrixuathien[2][0]=3; Vitrixuathien[2][1]=4

Vitrixuathien[2][2]=7

Vitrixuathien[3][0]=2; Vitrixuathien[3][1]=6

Vitrixuathien[4][0]=5

Sample Output

2
0
7
0
Khi đó, nếu $k = 1, v = 3 \Rightarrow$ tra vitrixuathien[3][0] = 2
nếu $k = 3, v = 2 \Rightarrow$ tra vitrixuathien[2][2] = 7
nếu $k = 2, v = 4 \Rightarrow$ tra vitrixuathien[4][1] = ???

151

Bài tập 1

```
#include <stdio.h>
#include <vector>
using namespace std;
vector<int> vitrixuathien[1000001];

int main() {
    int n, m, i, k, v;
    scanf("%d %d", &n, &m);
    for(i = 0; i < n; i++) {
        scanf("%d", &v);
        vitrixuathien[v].push_back(i+1);
    }
    while(m--) {
        scanf("%d %d", &k, &v);
        if(k > vitrixuathien[v].size())
            printf("0\n");
        else
            printf("%d\n", vitrixuathien[v][k-1]);
    }
    return 0;
}
```

152