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

Applied Algorithms

Nguyễn Khánh Phương

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

Contents

Chapter 1

1. Basic data types
2. Big integer
3. Data structure and algorithm libraries
4. Representation a set using Bitmask
5. Graph representation
6. Augmenting data structure

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- ĐHBK HN

Chapter 1

1. **Basic data types**
2. Large integer
3. Data structure and algorithm libraries
4. Representation a set using Bitmask
5. Graph representation
6. Augmenting data structure

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- ĐHBK HN

1. Basic data types

- Must-known data types
  - `bool`: Boolean variable (true/false)
  - `char`: 8-bit integer variable (usually used to represent ASCII characters)
  - `string`: character string variable
  - `int`: 32-bit integer variable
  - `long`: 32-bit integer variable
  - `long long`: 64-bit integer variable
  - `float`: 32-bit real variable
  - `double`: 64-bit real variable
  - `long double`: 128-bit real variable
- Modifiers
  - `signed`
  - `unsigned`
  - `short`
  - `long`
  - `unsigned char, signed char, unsigned int, singed int, unsigned long, signed long`
  - `short int, unsigned short int,`
  - `long int, unsigned long int, long double`
  - …

5

1. Basic data types

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

On compiler GCC 64 bit

6

Chapter 1

1. Basic data types
2. **Large integer**
3. Data structure and algorithm libraries
4. Representation a set using Bitmask
5. Graph representation
6. Augmenting data structure

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- ĐHBK HN

2. Large integer process

- How to perform calculations on extremely large integers that cannot be stored using `long long int` (64 bit)?
- Idea:
  - Store that big integer as `string`
  - Perform math on each digit from right to left, saving the memory as learned in mathematic

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- ĐHBK HN

Example 1: add 2 large integers

A1. ADD

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

Given 2 integers a and b. Calculate a + b

Input

2 lines: each line contains an integer

Output

One line: contain the result of a+b

Scoring

$0 \le a, b \le 9 \times 10^{18}$ 50% test with $a, b \le 10^9$

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

➔ $2^{63} \approx 9*10^{18}$~ long long int

Examples

input	Copy
1 1	

output	Copy
2	

Example 1: add 2 large integers

```
  12345678919   string a
+ 00000012345   string b
```

```
string Sum(string a, string b)
```

- Step 1: Since each of these integers is represented by a string, we will make these two strings of equal length by inserting the character "0" in front of the shorter string.

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

    while (b.length() < a.length())
        b = "0" + b;
```

Example 1: add 2 large integers

```
  12345678919   string a
+ 00000012345   string b
```

- Step 2: perform the addition: traverse each character of the two strings in order from right to left, add two corresponding characters together and note the "remember" part if any

a[i] + b[i] ???

(a[i]-'0') + (b[i]-'0')

(a[i]- 48) + (b[i]- 48)

48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9

```
string result;
int sum, mem = 0;
int numChar = a.length();
for (int i= numChar - 1; i>=0; i--)
{
    sum = (a[i] - '0') + (b[i] - '0') + mem;
    mem = sum/10;
    result = char(sum%10 + '0') + result;
}
if (mem == 1) result = "1" + result;
```

11

Example 1: Source code add 2 large integers

```
#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 result;
    int sum, mem = 0;
    int numChar = a.length();
    for (int i=numChar - 1; i>=0; i--)  {
        sum = (a[i] - '0') + (b[i] - '0') + mem; //sum = (a[i] - 48) + (b[i] - 48) + mem;
        mem = sum/10;
        result.push_back(sum%10 + '0'); //result = char(sum%10 +'0') + result;
    }
    if (mem == 1) result.push_back(mem+'0'); //if (mem == 1) result = "1"+result;
    reverse(result.begin(), result.end());
    return result;
}

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

Example 1: Source code add 2 large integers

```
#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 result;
    int sum, mem = 0;
    int numChar = a.length();
    for (int i=numChar - 1; i>=0; i--)  {
        sum = (a[i] - '0') + (b[i] - '0') + mem; //sum = (a[i] - 48) + (b[i] - 48) + mem;
        mem = sum/10;
        result.push_back(sum%10 + '0'); //result = char(sum%10 +'0') + result;
    }
    if (mem == 1) result.push_back(mem+'0'); //if (mem == 1) result = "1"+result;
    reverse(result.begin(), result.end());
    return result;
}

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


Example 1: add 2 large integers

```
  12345678911   string a
+ 00000012345   string b
```

`string Sum(string a, string b)`

- Method 2: Instead of entering "0" so that the two strings have the same length as we did in the previous slides, we will reverse the two strings

  1198765321
  54321

then, as in this example, we will only add the 5 rightmost characters of string a with string b in order from right to left, then concatenate the result with the remaining characters of string a (note: still have to add "remember" if any). Reverse the result string, we will get the solution of the problem

Example 1: Source code add 2 large integers Method 2

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

string Sum(string a, string b)
{
    //Length of string b > length of string a
    if (a.length() > b.length()) swap(a, b);
    string result = "";
    int n1 = a.length(), n2 = b.length();
    //Reverse string a and b:
    reverse(a.begin(), a.end());  reverse(b.begin(), b.end());
    int mem = 0;
    for (int i=0; i<n1; i++)
    {
        int sum = ((a[i]-'0')+(b[i]-'0')+ mem);
        result.push_back(sum%10 + '0');
        mem = sum/10;
    }
    //Add the remaining digits of b into result
    for (int i=n1; i<n2; i++)
    {
        int sum = ((b[i]-'0')+ mem);
        result.push_back(sum%10 + '0');
        mem = sum/10;
    }
    //Add mem if any:
    if (mem) result.push_back(mem+'0');
    //Reverse string containing the result:
    reverse(result.begin(), result.end());
    return result;
}
```


Example 2: 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
```

Sample Output

```
370370367037037036703703703670
```

Number of lines: max = 100
Each line contains an integer with ≤ 100 digits
The final line contain 0 to terminate the input

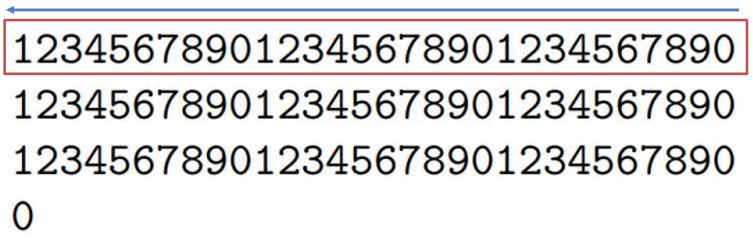
Sum of 100 integers (each integer contains 100 digits), is an integer containing how many digits

Sum is integer containing maximum of 100*2 = 200 digits

```
int Sum[201]={0};
```

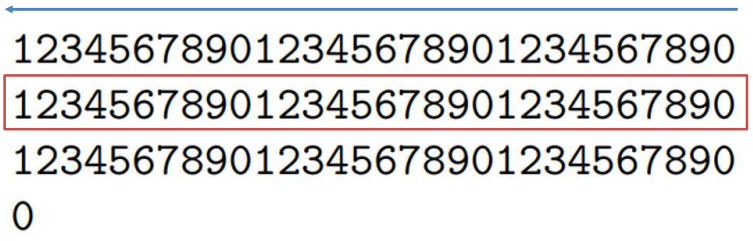
Sum[0]
Sum[1]
Sum[2]

Example 2: Integer Inquiry




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

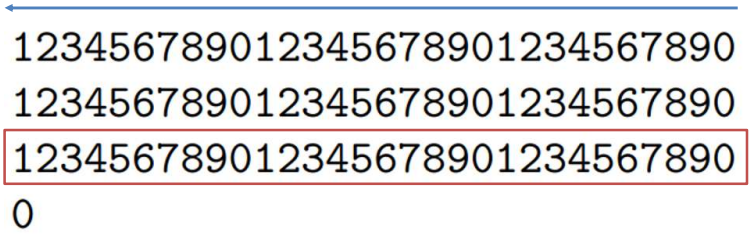

Example 2: Integer Inquiry




```
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
........
```

Example 2: Integer Inquiry




```
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
........
```


Example 2: Integer Inquiry

12345678901234567890123456789
12345678901234567890123456789
12345678901234567890123456789
0

```
Sum[0]+= 0 → Sum[0]= 0
Sum[1]+= 9 → Sum[1]= 18+9= 27
Sum[2]+= 8 → Sum[2]= 16+8= 24
........
```

Process "mem"

```
Sum[0]= 0
Sum[1]= 7; mem 2 → add it to Sum[2]
Sum[2]= 4 + 2 (mem from Sum[1]) = 6; mem
2 → add it to Sum[3]
........
```

```
#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';
    }
    //Fill in the "mem" process here

    return 0;
}
```

Example 2: 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';
    }
    //"mem" process:
    for(i = 0; i < 200; i++)
        if(Sum[i] >= 10) {
            Sum[i+1] += Sum[i]/10;
            Sum[i] %= 10;
        }
```

//Print result:

```
    for(i = 200; i >= 0; i--)
        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
........
```

"mem" process

```
Sum[0]= 0
Sum[1]= 7; mem 2 → add it to Sum[2]
Sum[2]= 4 + 2 (mem from Sum[1]) = 6;mem 2
→ add it to Sum[3]
........
```

??? Which number will be printed on the screen

Example 2: Integer Inquiry

??? How to remove all unnecessary digits "0" from result

```
1234567890
123456
1453444342444
0
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000001454679033790
```

//Print result:

```
    for(i = 200; i >= 0; i--)
        cout<<Sum[i];
```

Example 2: 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';
    }
    //"mem" process:
    for(i = 0; i < 200; i++)
        if(Sum[i] >= 10) {
            Sum[i+1] += Sum[i]/10;
            Sum[i] %= 10;
        }

    //Print result:
    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;
}
```


Chapter 1

1. Basic data types
2. Large integer
3. **Data structure and algorithm libraries**
4. Representation a set using Bitmask
5. Graph representation
6. Augmenting data structure

3. Data structures and algorithms libraries

Standard Template Library (STL) in C++ is divided into 4 parts:

- Containers Library : contains template data strucutres
  - Sequence containers (cấu trúc tuần tự): Vector, Deque, List
  - Containers adpators: Stack, Queue, Priority_queue
  - Associative containers (cấu trúc liên kết): Set, Multiset, Map, Multimap, Bitset
- Algorithms Library: some algorithms to manipulate data
- Iterator Library: like pointer, to access the elements of container
- Numeric library

- To use STL, we use "`using namespace std;`" after the library declarations ("`#include`", "`#define`",...)

25

3. Data structures and algorithms libraries

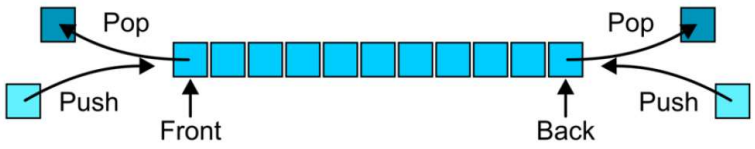
Some data structures:

- Static array `int arr[10]`
- Dynamic array `vector<int>`
- Linked list `list <int>`
- Stack `stack <int>`
- Queue `queue <int>`
- Priority queue `priority_queue <int>`
- Double-ended queue `deque <int>`
- Set (Tập hợp) `set <int>`
- Map (Ánh xạ) `map <int, int>`

26

Double-ended queue (deque)

- `#include <deque>` (Deque = Double-Ended Queue; đọc là deck): is a data structure obtaining the properties of both Stack and Queue, which allows adding and removing at both ends.
- Deque is similar to **vector** but we can manipulate the operations add and remove from two ends.
- Deque has the following advantages:
  - The element is accessible via its position index (similar to vector).
  - Insert or delete an element at the end or beginning of the list (the vector only inserts or deletes the element at the end of the list).



STL in C++: deque

Capacity	
size()	Return the number of elements in queue
empty()	True/False
Access element	
Operator [int i]	Access the ith element of queue, similar as in array
at(int i)	Access that ith element of deque
front()	Access the first element of deque
back()	Access the last element of deque
Query	
push_back(x)	Add element of value x at the end of deque.
push_front(x)	Add element of value x at the beginning of deque
pop_back()	Remove the last element from deque.
pop_front()	Remove the first element from deque.

STL in C++: deque

Query	
insert (iterator pos,const x)	Insert element with value x before position pos.
insert (iterator pos,int n, const x)	Insert n elements with value x before position pos.
insert (iterator pos, iterator a, itertator b)	Inserts before the pos position all the elements in the position of range [a,b) of another deque.
erase (iterator pos)	Remove element at position pos
erase (iterator first, iterator last)	remove all elements in the range [first,last), i.e. from the first element to the element (last-1)th
swap(vector v)	Swap the elements of the current deque and the vector v
clear()	Delete deque

For example, create a deque with 5 elements that all have the value 100. Create an int array with 3 elements of 200, 300, 400 respectively.

Then, step 1: Insert the first 2 elements of the array at the beginning of the deque.
Step 2: Insert the first 2 elements of the array before the second element of the 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

   //Step 1: Insert the first 2 elements of the array at the beginning of the deque:
    myDeque.insert(myDeque.begin(), myArray, myArray +2 ) ; // 200 300 100 100 100 100 100
  // print deque
    cout<<"Deque contains numbers: ";
    for(int i = 0 ; i < myDeque.size() ; i++) cout<<myDeque[i]<<" " ;

   //Step 2: Insert the first 2 elements of the array before the 2nd element of the deque:
    myDeque.insert(myDeque.begin()+1, myArray, myArray +2 ) ; // 200 200 300 300 100 100 100 100 100
  // print deque
    cout<<endl<<"Deque contains numbers: ";
    for(int i = 0 ; i < myDeque.size() ; i++) cout<<myDeque[i]<<" " ;

    return 0;
}
```

STL in C++: map

`#include <map>` each element of the map consists of two components: its key (key value) and its mapping (mapped value). The elements in the map do not contain the same key. The elements in the map are sorted in ascending order of the key value.

Declaration:

```
map <key_type, value_type> mapName;
```
//declare mapName where each element has key field with the type of `key_type`, value field with the type of `value_type`

Example: Declare map with key is string, value is integer:

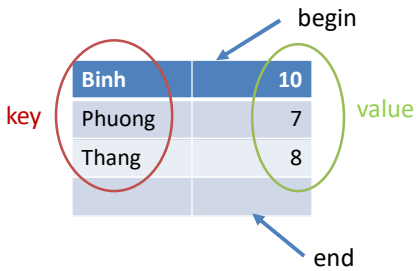
```
map <string, int> map1;
map1["Thang"] = 8; //Add element key = "Thang",value = 8
map1["Binh"] = 10; // Add element key = "Binh",value = 10
map1["Phuong"] = 7; // Add element key = "Phuong",value = 7
```


STL in 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
```



STL in 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;
}
```

?What will be displayed on screen

STL in C++: map

Capacity	
size()	Return number of elements in map
empty()	True/False
Truy cập phần tử	
begin()	Return the iterator of the first element in map
end()	Return the iterator of the last element in map
Query	
insert(x)	Insert element x (consists of key and value) into map (Note: if key of x exists already in map, then not insert x into map)
insert_or_assign()	Same as insert, but if the key of the element you want to add to the map is already in the map, its value will be changed by the value of the element you want to add.
erase(iterator pos)	Remove the element pointed by iterator `pos` from map
erase(iterator start, iterator end)	Remove elements pointing from iterator `pos` to `end` from map
clear()	Remove all elements from map

STL in C++: map

Truy vấn	
swap(map1)	Swap the elements of two maps
find(key1)	Returns iterator pointing to the element to be searched for with key = key1. If not found, point to the end of the map

35

Exercise

You are given two list 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 list 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 list 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 element in the first list and M ($1 \leq M \leq 10000$), the number of element in the second list. The next line will contain N integers representing the first list followed by another line having M elements representing the second list. Each integers 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
```

36

Exercise: source code

```
#include <iostream>
#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: key is the integer x in the list,
                               value is the number of times that x appears in the list */
        while(n--)
            scanf("%d", &x), r[x]++; //count on list1: increase value by 1 for each apperance of x in list
        while(m--)
            scanf("%d", &x), r[x]--; //count on list2: decrease value by 1 for each appearance of x in list

        int numDel = 0; // = su of value of all elements in the map
        for(map<int,int>::iterator it = r.begin(); it != r.end(); it++)
            numDel += abs(it->second);
        cout<<numDel<<endl;
        count++;
    }
}
```

37

3. Data structures and algorithms libraries

Sort and search function in <#include algorithm>

- Sort an array
- Search on an unsorted array
- Search on a sorted array
- Find min, max of 2 elements
- Find next/previous permutation

3. Data structures and algorithms libraries

- Sort an array

```
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>());
}
```


3. Data structures and algorithms libraries

- Sort an array

```
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;
}
```

What is the output?

3. Data structures and algorithms libraries

- Search on an unsorted array

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

Returns the index of the first element in [first, last) if it is equal to the value val. If no element is found, the function returns last .

Example: function `find`

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

int main () {
  //1. Find element with value 30 in the array:
  int myints[] = { 40, 30, 10, 20 };
  int *p;

  p = find (myints, myints+4, 30);
  if (p != myints+4)
    cout << "Value 30 is in the array myints at element: " << (p-myints+1) << '\n';
  else
    cout << "Value 30 is not in the array myints\n";
  return 0;
}
```

```
Value 30 is in the array myints at element: 2
```

Example: function `find`

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

int main () {
  int myints[] = { 40, 30, 10, 20 };
  //1. Find element with value 30 in the vector:
  vector<int> M(myints,myints+4);
  vector<int>::iterator it;

  it = find (M.begin(), M.end(), 30);
  if (it != M.end())
      cout << "Value 30 is in the vector M at element: " << (it - M.begin()+1) << '\n';
  else
    cout << "Value 30 is not in the vector M\n";
  return 0;
}
```

```
Value 30 is in the vector M at element: 2
```


3. Data structures and algorithms libraries

- Search on an unsorted array by condition

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

Returns the index of the first element in `[first, last)` that satisfies the `pred` condition. If no element is found, the function returns `last`.

There are also other functions:

- `find_if_not`
- `find_end`
- `find_first_of`
- `adjacent_find`

Example: function `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, 28, 38} ;

    int *p = find_if(myArray, myArray+4, IsOdd);
    if (p != myArray+4)
      cout << "The first odd element in myArray is " << *p <<"; at the position "<<(p-myArray+1)<<" in array";
    else
        cout<<"myArray does not have odd value";
    return 0;
}
```

```
myArray does not have odd value
```


3. Data structures and algorithms libraries

- Search on an unsorted array

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

Returns the index of the element with the smallest value in `[first, last)`.

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

Returns the index of the element with the largest value in `[first, last)`.

Example 1: function 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;
}
```


Example 1: function 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 smallest 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;
}
```

3. Data structures and algorithms libraries

- Search on a sorted array

```
template <class ForwardIterator, class T>
  ForwardIterator lower_bound (ForwardIterator first,
ForwardIterator last, const T& val);
```

Returns `iterator` pointing to the first element in `[first, last)` not less than (>=) `val`. If all elements are less than `val`, return `last`

```
template <class ForwardIterator, class T>
  ForwardIterator upper_bound (ForwardIterator first,
ForwardIterator last, const T& val);
```

Returns `iterator` pointing to the first element in `[first, last)` larger than `val`. If not exist, return `last`

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

Return `true` if `val` is in the array

49

Example: function `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 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 (first element in list >=20) at position " << (low- v.begin()+1) << '\n';
  cout << "upper_bound (first element in list  > 20) at position " << (up - v.begin()+1) << '\n';

  return 0;

}
```

```
lower_bound (first element in list >=20) at position 4
upper_bound (first element in list  > 20) at position 7
```

3. Data structures and algorithms libraries

- Search on a sorted array

```
(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);
```

Return `true` if `val` is in the array

When compare 2 values `a` and `b`:

At template (1): comparison function "<" is used by default; it means `a==b` if `(!(a<b) && !(b<a))`

At template (2): function `comp` is used; it means `a==b` if `(!comp(a,b) && !comp(b,a))`.

Example: function `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 << "Find whether value 3 is in the vector v ... ";
  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 << "Find whether value 6 is in the vector v ... ";
  if (binary_search (v.begin(), v.end(), 6, myfunction))
    cout << "found!\n";
  else cout << "not found.\n";

  return 0;
}
```

Chapter 1

1. Basic data types
2. Large integer
3. Data structure and algorithm libraries
4. **Representation a set using Bitmask**
5. Graph representation
6. Augmenting data structure

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- ĐHBK HN

4. Representation a set using bit mask

Given a set S consisting of n elements (n is small $n \leq 64$). We label all elements in S from 0, 1, .. n-1.

- Then, we can represent each element in the set S by a 64-bit integer.
- The ith element of the set S is represented by the integer X if the ith bit of X is 1

Example: Set S consists of 5 girls {Lan, Lê, Lành, Linh, Lam}, these girls are labeled in the order 0, 1, 2, 3, 4 → Set S is represented by integer 2^5 -1= 31 = $11111_{(2)}$

- Set S_1 consisting of 3 girls {Lan, Lành, Linh} corresponds to the element at position 0, 2, 3 of S, it means S_1={0, 2, 3} is a subset of S. Set S_1 could be represented by integer x = 13 because

```
x = (1<<0)| (1<<2)|(1<<3) = 1 | 4 | 8 = 01101(2) = 13

      00001  There is 0th element → 1 << 0 = 00001
OR    00100  There is 2nd element  → 1 << 2 = 00100
      01000  There is 3rd element → 1 << 3 = 01000
      -----
      01101  There are 0th, 2nd, 3rd elements
```

Left shift: x<<y = x*2^y
Right shift: x>>y = x/2^y

4. Representation a set using bit mask

Consider the universe set N with *n* elements and a subset S of N:

- Add the jth element of the set N to the set S:
  - Do S |= (1 << j)
- Remove the jth element of the set N from the set S:
  - Do S &= ~(1 << j)
- Change the state of the jth element of the set N in the set S (it means, if *jth* element is in the set S then remove it from *S*, if *jth* element is not in the set S then insert it into S):
  - Do S ^= (1 << j)
- Check whether the jth element of the set N is in the set S:
  - Do T = S & (1 << j)
    - If T = 0 : the jth element is not in the set S
    - If T ≠ 0: the jth element is in the set S

AND (&)

a	b	a&b
0	0	0
0	1	0
1	0	0
1	1	1

OR (|)

a	b	a|b
0	0	0
0	1	1
1	0	1
1	1	1

XOR (^)

a	b	a^b
0	0	0
0	1	1
1	0	1
1	1	0

NOT (~)

a	~a
0	1
1	0

4. Representation a set using bit mask

Consider the universe set N with *n* elements and a subset S of N:

- Find the jth element with the smallest order in the set N that appear in the set S:
  - Do T = (S & (-S))

```
 S   =  40 (base 10) =  000...000101000 (32 bits, base 2)
-S   = -40 (base 10) =  111...111011000 (two's complement)
                        ---------------- AND
 T   =   8 (base 10) =  000...000001000
```

3rd bit from the right = 1 ➔ j = 3

AND (&)

a	b	a&b
0	0	0
0	1	0
1	0	0
1	1	1

OR (|)

a	b	a|b
0	0	0
0	1	1
1	0	1
1	1	1

XOR (^)

a	b	a^b
0	0	0
0	1	1
1	0	1
1	1	0

NOT (~)

a	~a
0	1
1	0

4. Representation a set using bit mask

Consider the universe set N with *n* elements and a subset S of N:

Set	Integer
Empty set	0
Subset with one element that is the ith element	1 << i
Universe set N (i.e. set with all elements) with n elements	(1 <<n) -1
Union of subsets S1 and S2 where: • Subset S1 is represented by integer x, • Subset S2 is represented by integer y.	x | y
Intersection of subsets S1 and S2 where:: • Subset S1 is represented by integer x, • Subset S2 is represented by integer y.	x & y
The complement of subset S1	~x & ((1<<n)-1)

AND (&)

a	b	a&b
0	0	0
0	1	0
1	0	0
1	1	1

OR (|)

a	b	a|b
0	0	0
0	1	1
1	0	1
1	1	1

XOR (^)

a	b	a^b
0	0	0
0	1	1
1	0	1
1	1	0

NOT (~)

a	~a
0	1
1	0

4. Representation a set using bit mask

Why we should use bitmask rather than `set <int>` ?

- Save memory
- All subsets of the set N with *n* elements could be represented by integers in the range 0 .. $2^n - 1$.
- Easy to traverse through all subsets of the set N:

```
for ( x = 0; x < (1 << n); ++x )
```

- Easy to traverse through all subsets of the subset S (not include empty set) represented by integer y:

```
for ( x = y; x > 0; x = ( y & (x-1) ) )
```

- Easy to use set as an index of the array

Example

Problem: Let **S** be a set of integers. Count the number of subsets of **S** for which the sum of the numbers in each of these subsets is equal to the given integer *k*.

Answer: If |S| small, we can use bitmask to traverse all subsets of S and sum the numbers in each subset to see if it is equal to *k*:

```
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;
}
```

Complexity: $O(2^{|S|}\ |S|)$

Example

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 //gán công 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;
```

Some applications of data structures

Some applications of array and linked list:

- Problem that need to store data, need to enumerate many elements
- Example: Broken keyboard: https://onlinejudge.org/external/119/p11988.pdf

Some applications of Stack:

- Handle events in a last-in-first-out sequence
- De-recursion
- DFS in graph
- Reverse a string
- Check brackets in expression / source code

Some applications of Queue:

- Handle events in a first-in-first-out sequence
- BFS in graph
- Find the path going through the least number of edges

Some applications of data structures

Some applications of priority queue:

- Handle events in order of precedence
- Find the shortest path on the graph
- Some greedy algorithms

Some applications of mapping:

- Assign each value to a unique key
- Frequency table
- Storing memory when implementing dynamic programming

Chapter 1

1. Basic data types
2. Big integer
3. Data structure and algorithm libraries
4. Representation a set using Bitmask
5. **Graph representation**
6. Augmenting data structure

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- ĐHBK HN

5. Graph representation

There are different types of graphs:

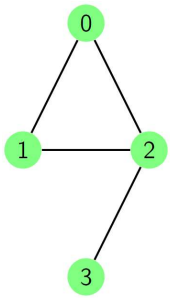
- Directional or non-directional
- Weighted or Unweighted
- Single Graph or Multigraph

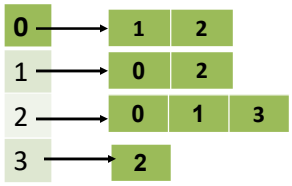
Some common graph representations:

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

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- ĐHBK HN

5.1. Represent graph by Adjacency lists (Danh sách kề)



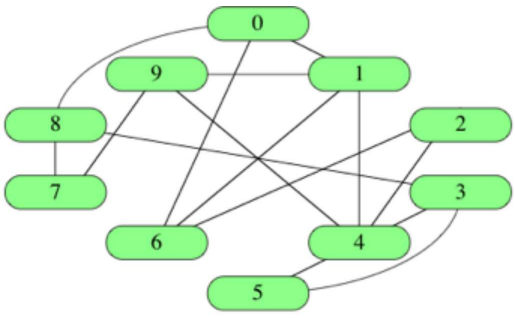


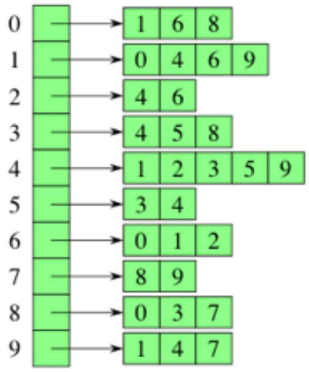

```
vector<int> adj[4];
adj[0].push_back(1);
adj[0].push_back(2);
adj[1].push_back(0);
adj[1].push_back(2);
adj[2].push_back(0);
adj[2].push_back(1);
adj[2].push_back(3);
adj[3].push_back(2);
```

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- ĐHBK HN

5.1. Represent graph by Adjacency lists (Danh sách kề)

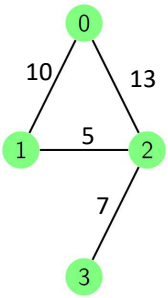
Exercise: write sourcecode to represent graph by using Adjacency list

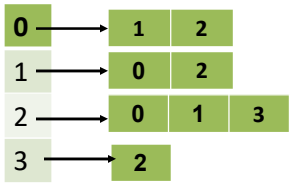




NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- ĐHBK HN

5.1. Represent graph by Adjacency lists (Danh sách kề)

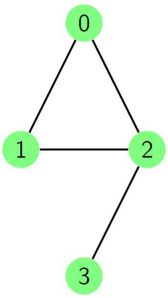





```
vector <pair <int, int> > adj[4];
//hoặc list <pair <int, int> > adj[4];
adj[0].push_back(make_pair(1, 10));
adj[1].push_back(make_pair(0, 10)); //undirected graph
adj[0].push_back(make_pair(2, 13));
adj[2].push_back(make_pair(0, 13)); // undirected graph
adj[1].push_back(make_pair(2, 5));
adj[2].push_back(make_pair(1,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

```
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. Represent graph by Adjacency matrix (Ma trận kề)

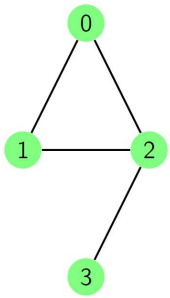



```
0 1 1 0
1 0 1 0
1 1 0 1
0 0 1 0
```

```
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;
```

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- ĐHBK HN

5.3. Represent graph by Edge lists (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

Compare cost analysis

	Adjacency list	Adjacency matrix	Edge list
Memory	$O(\|V\|+\|E\|)$	$O(\|V\|^2)$	$O(\|E\|)$
Add a vertex	$O(1)$	$O(\|V\|^2)$	$O(1)$
Add an edge	$O(1)$	$O(1)$	$O(1)$
Delete a vertex	$O(\|E\|)$	$O(\|V\|^2)$	$O(\|E\|)$
Delete an edge	$O(\|E\|)$	$O(1)$	$O(\|E\|)$
Query: whether vertices u and v are adjacency	$O(\|V\|)$	$O(1)$	$O(\|E\|)$

- Selection representation depends on the problem need to be solved
- Can use multiple representations at the same time

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- ĐHBK HN

Chapter 1

1. Basic data types
2. Big integer
3. Data structure and algorithm libraries
4. Representation a set using Bitmask
5. Graph representation
6. **Augmenting data structure**

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- ĐHBK HN

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

- **An augumenting data structure:** is an existing data structure (e.g. doubly linked list, binary search tree..) with some information added/modified to be able to solve the requirements of the problem faster.
- Normally this is not possible with data structures in the standard library. It need to be self-implemented for customization

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- ĐHBK HN

Example: Augmenting BST (Cây nhị phân tìm kiếm mở)

Given a binary search tree, we need to:

- Find the kth smallest element
- Count the number of elements with value $< x$ (it means: given a node with value x, determine rank(x))

Direct method: Traverse all the nodes on the tree $O(n)$

➔ Improve: build augmenting BST so that these 2 operations cost only $O(\log n)$

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- ĐHBK HN

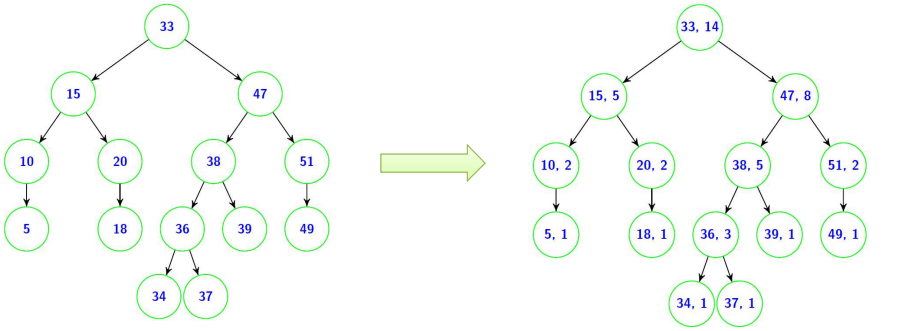
Example: Augmenting BST (Cây nhị phân tìm kiếm mở)

Given a binary search tree, we need to:

- Find the kth smallest element
- Count the number of elements with value $< x$

Direct method: Traverse all the nodes on the tree $O(n)$

➔ Improve: build augmenting BST so that these 2 operations cost only $O(\log n)$

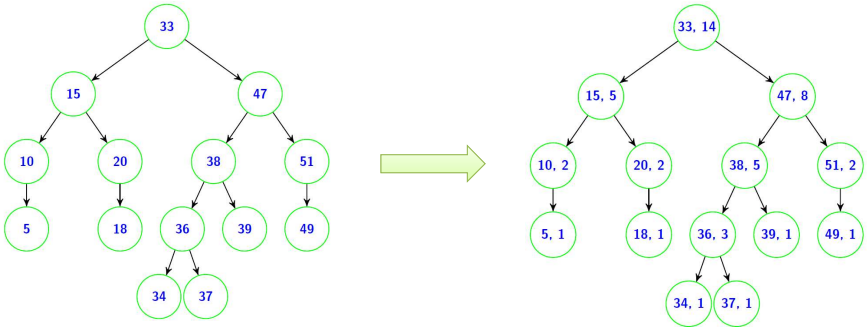


At each node, store additional information: the number of nodes in the tree it is the root

Example: Augmenting BST (Cây nhị phân tìm kiếm mở)

Augmenting BST:

- At each node, store additional information: the number of nodes in the tree it is the root
- This information will be updated every time a node is added/removed on the tree without affecting the overall complexity of the algorithm.



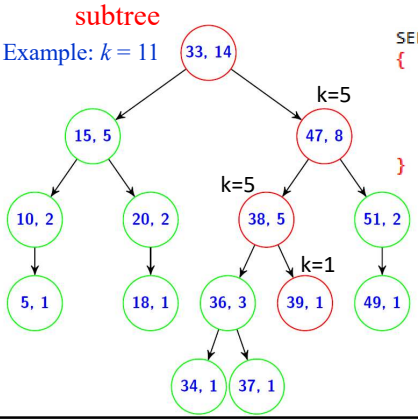
At each node, store additional information: the number of nodes in the tree it is the root

Example: Augmenting BST (Cây nhị phân tìm kiếm mở)

Q1: Find the kth smallest element on the tree with the root pointed by `root`:

- Derived from the root. At each node whose left subtree has size *r*:
  - If $k = r+1$, then the search element is the current node
  - If $k \leq r$, continue to find the kth smallest element in the left subtree
  - If $k > r + 1$, continue to find the $(k-r-1)$th smallest element in the right subtree

Example: $k = 11$

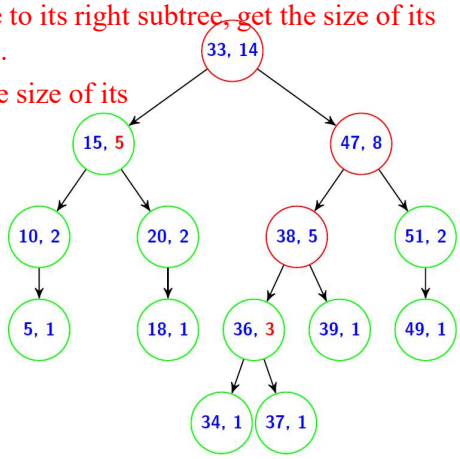



```
SELECTk(root, k)
{
   r =  size[left[root]];
   if (k == r +1) then return root;
   else if (k <= r) then return SELECTk(left[root], k);
   else return SELECTk(right[root], k - r - 1);
}
```

Example: Augmenting BST (Cây nhị phân tìm kiếm mở)

Q2: Count the number of nodes with value < 38 on the tree with roote node pointed by pointer `root`:

- Start from root, find value 38 on the tree:
  - Count the number of traversed nodes that are smaller than 38: if from a current node, we need to traverse to its right subtree, get the size of its left subtree +1, then add to result.
  - When traverse to node 38: get the size of its left subtree and add to the result.

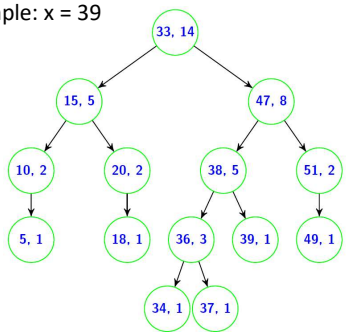


Example: Augmenting BST (Cây nhị phân tìm kiếm mở)

Q2 (ask in another form): Given a node pointed by *x*. Determine `rank(x)`

- Init : result stored in variable r = size of left subtree of node *x*
- Start from node *x*: set *y* = *x*, traverse back up to the root:
  - when traversing a node *y* where *y* is the right child of its parent, take the size of the left subtree of its parent plus 1, and add to result variable r, continue setting `y = parent[y];`

Example: x = 39




```
RANK(root, x)
{
    r = size[left[x]];
    y = x;
    while (y != root) //duyet tu nut y len den goc
    {
        if (y == right[parent[y]]) // if y is right child of its parent
            r += size[left[p[y]]] + 1;
        y = parent[y]; //set y = parent of y
    }
    return r;
}
```

Example 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.

Sample Input

```
8 4
1 3 2 2 4 3 2 1
1 3
2 4
3 2
4 2
```

Sample Output

```
2
0
7
0
```

Hint: Use 2D array **posOccur**:

For each integer value v (1≤v≤1,000,000) store all the positions where v appears in the input array:

posOccur[v][0..number of times that v appears in input array]

Example:for an array in Sample Input, then values of elements in array postOccur:

```
posOccur[1][0]=1; posOccur[1][1]=8
posOccur[2][0]=3; posOccur[2][1]=4
posOccur[2][2]=7
posOccur[3][0]=2; posOccur[3][1]=6
posOccur[4][0]=5
```

Then, if k = 1, v = 3 → see posOccur[3][0] = 2

if k = 3, v = 2 → see posOccur[2][2] = 7

if k = 2, v = 4 → see posOccur[4][1] = ???

79

Example 1: source code

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

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

80