copy

```cpp
#include <iostream>
#include <algorithm>
#include <vector>

    using namespace std;

    int main (){
    int myints[] = {10, 20, 30, 40, 50, 60, 70};
    vector<int> myvector;
    vector<int>::iterator it;

    myvector.resize(7);    // 为容器myvector分配空间

    //copy用法一:
    //将数组myints中的七个元素复制到myvector容器中
    copy ( myints, myints+7, myvector.begin() );

    cout << "myvector contains: ";
    for ( it = myvector.begin();  it != myvector.end();  ++it )
    {
        cout << " " << *it;
    }
    cout << endl;
    //copy用法二:
    //将数组myints中的元素向左移动一位
    int temp=myints[0];
    copy(myints + 1, myints + 7, myints);
    myints[6]=temp;
    cout << "myints contains: ";
    for ( size_t i = 0; i < 7; ++i )
    {
        cout << " " << myints[i];
    }
    cout << endl;

    return 0;
}
```

deque

```
deque概念
容器deque和vector非常相似，属于序列式容器。都是采用动态数组来管理元素，提供随机存取，并且有着和vector一样的接口。不同的是deque具有首尾两端进行快速插入、删除的能力。
创建deque
```

```
deque<Elem> c;                          创建一个空的deque
deque<Elem> c1(c2);                     复制deque,复制跟c1一模一样的队列c2
deque<Elem> c(n);                       创建一个deque,元素个数为n,且值均为0
deque<Elem> c(n,num);                   创建一个deque,元素个数为n,且值均为num
c.assign(n,num);                        初始化deque, 初始化后元素个数为n,且值均
为num
deque<int>::iterator it;                正向迭代器
deque<int>::reverse_iterator rit;       逆向迭代器
数据访问
c.at(idx);                              返回索引下标idx所指的数据（从0开始）
c.front();                              返回第一个数据
c.back();                               返回最后一个数据
c.begin();                              返回指向第一个数据的迭代器
c.end();                                返回指向最后一个数据的下一个位置的迭代器
c.rbegin();                             返回逆向队列的第一个数据
c.rend();                               返回指向逆向队列的最后一个数据的下一个位
置的迭代器
加入数据（pos、beg、end均为迭代器）
c.push_back(num);                       在尾部加入一个数据num
c.push_front(num);                      在头部插入一个数据num
c.insert(pos,num);                      在该pos位置的数前面插入一个num
c.insert(pos,n,num);                    在该pos位置的数前面插入n个num
c.insert(pos,beg,end);                  在该pos位置的数前插入在[beg,end)区间的
数据
删除数据（pos、beg、end均为迭代器）
c.pop_back();                           删除最后一个数据
c.pop_front();                          删除头部数据
c.erase(pos);                           删除pos位置的数据
c.erase(beg,end);                       删除[beg,end)区间的数据
其他操作
c.clear();                              销毁所有数据，释放内存
c.empty();                              判断容器是否为空
c.resize(n);                            deque队列的长度置为n，只保留队列前n个数
c.size();                               返回容器中实际数据的个数
swap(c1,c2);                            将c1和c2元素互换
```

```
//在deque中查找某元素
deque<int>::iterator pos=find(c.begin(),c.end(),num);
if(pos!=c.end()) printf("find success\n");
else printf("find failed\n");
//反向遍历队列中的元素：
deque<int>::reverse_iterator rit;
for(rit = c.rbegin();rit != c.rend(); rit++)
        printf("%d ",*rit);
#include <iostream>
#include <deque>
#include <cstdio>

using namespace std;
```

```cpp

int nums[4000001];

int main() {
    int n, m, left_result = 0, right_result = 0, sum_result = 0, left = 1, sum = 0;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        cin >> nums[i];
    }
    deque<int> queue;
    for (int right = 1; right <= n; right++) {
        //cout<<"right="<<right<<" left="<<left<<endl;
        queue.push_back(nums[right]);
        //cout<<"front="<<queue.front()<<endl;
        //cout<<"back="<<queue.back()<<endl;
        sum += nums[right];
        while (sum > m) {
            queue.pop_front();

            sum -= nums[right];

            if (sum > sum_result) {
                sum_result = sum;
                right_result = right - 1;
                left_result = left;
            }
            sum += nums[right];
            sum -= nums[left];

            left++;
        }
    }
    if (sum > sum_result) {
        sum_result = sum;
        right_result = n;
        left_result = left;
    }
    cout << left_result << " " << right_result << " " << sum_result;
    return 0;
}
```

list

```
Lst1.assign()           给list赋值
Lst1.back()             返回最后一个元素
Lst1.begin()            返回指向第一个元素的迭代器
Lst1.clear()            删除所有元素
Lst1.empty()            如果list是空的则返回true
```

```
Lst1.end()              返回末尾的迭代器
Lst1.erase()            删除一个元素
Lst1.front()            返回第一个元素
Lst1.get_allocator()    返回list的配置器
Lst1.insert()           插入一个元素到list中
Lst1.max_size()         返回list能容纳的最大元素数量
Lst1.merge()            合并两个list
Lst1.pop_back()         删除最后一个元素
Lst1.pop_front()        删除第一个元素
Lst1.push_back()        在list的末尾添加一个元素
Lst1.push_front()       在list的头部添加一个元素
Lst1.rbegin()           返回指向第一个元素的逆向迭代器
Lst1.remove()           从list删除元素
Lst1.remove_if()        按指定条件删除元素
Lst1.rend()             指向list末尾的逆向迭代器
Lst1.resize()           改变list的大小
Lst1.reverse()          把list的元素倒转
Lst1.size()             返回list中的元素个数
Lst1.sort()             给list排序
Lst1.splice()           合并两个list
Lst1.swap()             交换两个list
Lst1.unique()           删除list中重复的元素
```

```cpp
#include <iostream>
#include <cstdio>
#include <list>
#include <iterator> // std::advance distance
using namespace std;
list<int> arr;

// int* getHead(int* arr) { return arr; }

int main()
{
    int n, m, sum = 1;
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {
        arr.push_back(i);
    }
    // for(list<int>::const_iterator iter = arr.begin();iter != arr.end();iter++)
    // {
    //     cout<<*iter<<' ';
    // }

    // int arr2[] {1, 2, 3};
    // int* ptr = arr2;
    // int* ptr2 = &arr2[0];
```

```
    // int* ptr3 = getHead(arr2);
    // cout << *ptr << endl;
    // *(arr2 + 2)

    list<int>::iterator iter = arr.begin(); // bidirectional iterator 双向迭代器
    // iterator category 迭代器类型

    // forward iterator 前向迭代器  std::forward_list
    // bidirectional iterator 双向迭代器  std::list std::set std::map
    // random access iterator 随机访问迭代器  std::vector std::deque
    for (int i = m - 1; !arr.empty(); arr.size() != 0 && (i = (i + m - 1) % arr.size()))
    {
        auto itr = arr.begin();
        advance(itr, i);//advance for(int i=0;i<i;i++){ iter++}
        //distance(arr.begin(), itr);
        if (arr.size() != n) cout << ' ';
        cout << *itr;
        arr.erase(itr);
    }

    cout << endl;
    return 0;
}



#include <iostream>
#include <list>
#include <iterator>
#include <algorithm>
using namespace std;

int main()
{
    list<int> lst {1, 2, 3};
     auto begin = lst.begin();
    // begin++;
    // begin = lst.erase(begin);
    // cout << *begin << endl;

    // copy(lst.begin(), lst.end(), ostream_iterator<int>(cout, " "));
    // cout << endl;
    advance(begin, 1);
    cout<<*begin<<endl;

    return 0;
}
```

map

```cpp
#include<iostream>
#include<cstdio>
#include<map>
#include<string>
using namespace std;
map<string,string>fam;
string findd(string x){
    while(fam[x]!=x){
        x=fam[x];
    }
    return x;
}
int main(){
    char flag;
    string fri,name;
    while(cin>>flag){
        if(flag=='$')
            return 0;
        if(flag=='#'){
            cin>>fri;
            if(fam.count(fri)==0)
                fam[fri]=fri;
            //getchar();
        }
        if(flag=='+'){
            cin>>name;
            //cout<<name<<' '<<fri<<endl;
            fam[name]=fri;
        }
        if(flag=='?'){
            string seek,seek_fri;
            cin>>seek;
            seek_fri=findd(seek);
            cout<<seek<<' '<<seek_fri<<endl;
        }
    }
}
1. 简介
map 是 STL 的一个关联容器，它提供一对一的hash

第一个称为关键字(key)，每个关键字只能在map中出现一次；
第二个称为该关键字的值(value)；
map以模板(泛型)方式实现，可以存储任意类型的数据，包括使用者自定义的数据类型。Map主要用于资料一对一映射(one-to-one)的情况，map内部的实现自建一颗红黑树，这颗树具有对数据自动排序的功能。在map内部所有的数据都是有序的，后边我们会见识到有序的好处。比如一个班级中，每个学生的学号跟他的姓名就存在著一对一映射的关系。

```

标准卡提供 8 个关联容器

2. pair 类型

在介绍关联容器操作之前，先了解一下 pair 的标准库类型。pair类型是在有文件 utility 中定义的，pair是将2个数据组合成一组数据，当需要这样的需求时就可以使用pair，如STL中的map就是将key和value放在一起来保存。另一个应用是，当一个函数需要返回2个数据的时候，可以选择 pair。 pair的实现是一个结构体，主要的两个成员变量是 first ，second 因为是使用struct 不是 class，所以可以直接使用pair的成员变量。

2.1 pair 类型的定义和初始化

pair 类型包含了两个数据值，通常有以下的一些定义和初始化的一些方法：

`pair<T1, T2> p;`	定义了一个空的pair对象p，T1和T2的成员都进行了值初始化
`pair<T1, T2> p(v1, v2);`	p是一个成员类型为T1和T2的pair; first和second成员分别用v1和v2进行初始化。
`pair<T1, T2> p = {v1, v2}`	等价于p(v1, v2)
`make_pair(v1, v2)`	以v1和v2值创建的一个新的pair对象

2.2 pair 对象的一些操作,除此之外，pair对象还有一些方法，如取出pair对象中的每一个成员的值：

`p.first`	返回p的名为 first 的(公有)数据成员
`p.second`	返回p的名为second的(公有)数据成员
`p1 relop p2`	关系运算符 (<、>、<= 、>=) 按字典序定义。

例如，当 p1.first < p2.first 或 !(p2.first < p1.first) && p1.second < p2.second 成立时

p1 < p2 为 true。关系运算利用元素的 < 运算符来实现

`p1 == p2`	当 first 和 second 成员分别相等时，两个pair相等
`p1 != p2`	若不能达到以上要求，则不相等

例如：

```
#include <stdio.h>
#include <string.h>
#include <string>
#include <utility>
using namespace std;

int main(){
    pair<int, string> p1(0, "Hello");
    printf("%d, %s\n", p1.first, p1.second.c_str());
    pair<int, string> p2 = make_pair(1, "World");
    printf("%d, %s\n", p2.first, p2.second.c_str());
    return 0;
}
```

3. map基本操作

3.1 头文件

```
#include <map>
```

3.2 创建map对象

map是键-值对的组合，即map的元素是pair，其有以下的一些定义的方法：

```
map<k, v> m;              定义了一个名为m的空的map对象
map<k, v> m2(m);          创建了m的副本m2
map<k, v> m3(b, e);       创建了map对象m3，并且存储迭代器b和e范围内的所有元素的副本
```

map 的 value_type 是存储元素的键以及值的pair类型，键为 const。

```
map<int, char> m;                          // 定义了一个名为m的空的map
map<int, char> m2(m);                      // 创建了m的副本m2
map<int, char> m3(m.begin(), m.end());     // 创建了map对象m3，并且存储迭代器范围内的所有元素的副本
```

3.3 map元素访问

注意：下标[] 和 at() 操作，只使用与非 const 的 map 和 unordered_map

3.3.1 使用下标 [ ] 访问

```
#include <iostream>
#include <map>
#include <string>

int main() {
    std::map<char, std::string> mymap;

    mymap['a'] = "an element";
    mymap['b'] = "another element";
    mymap['c'] = mymap['b'];

    std::cout << "mymap['a'] is " << mymap['a'] << '\n';
    std::cout << "mymap['b'] is " << mymap['b'] << '\n';
    std::cout << "mymap['c'] is " << mymap['c'] << '\n';
    std::cout << "mymap['d'] is " << mymap['d'] << '\n';

    std::cout << "mymap now contains " << mymap.size() << " elements.\n";
    return 0;
}
/*
mymap['a'] is an element
mymap['b'] is another element
mymap['c'] is another element
mymap['d'] is                    // 下标访问不会进行下标检查
mymap now contains 4 elements.
*/
```

注意：下标访问不会做下标检查，如上第4行打印的语句不会报错，但打印结果为空，因为下标访问会插入不存在的key，对应的value为默认值

而使用 at() 访问则会做下标检查，若不存在该key会报错

3.3.2 使用 at() 方法访问

```
#include <iostream>
#include <map>
#include <string>

int main() {
    std::map<std::string, int> mymap = {
        {"alpha", 0}, {"beta", 0}, {"gamma", 0}};

    mymap.at("alpha") = 10;
    mymap.at("beta") = 20;
    mymap.at("gamma") = 30;

    for (auto& x : mymap) {
        std::cout << x.first << ": " << x.second << '\n';
    }

    return 0;
}
/*
alpha: 10
beta: 20
gamma: 30
*/
```

3.4 map中元素的插入

在map中元素有两种插入方法：1. 使用下标 [] 2. 使用 insert() 函数

3.4.1 使用下标[]插入

使用下标访问不存在的元素，将会在map容器中添加一个新的元素；

使用下标访问存在的元素，将会覆盖map容器中的该元素

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

int main() {
    map<int, char> mymap;
    mymap[0] = 'a';
    mymap[1] = 'b';
    mymap[2] = 'c';
    mymap[0] = 'x';
    for (map<int, char>::iterator iter = mymap.begin(); iter != mymap.end(); iter++)
        cout << iter->first << " ==> " << iter->second << endl;
    return 0;
}
```

3.4.2 使用insert()插入元素

insert函数的插入方法主要有如下：

```
pair<iterator,bool> insert (const value_type& val);
```

插入单个键值对，并返回插入位置和成功标志，插入位置已经存在值时，插入失败

```
iterator insert (const_iterator position, const value_type& val);
```

在指定位置插入，在不同位置插入效率是不一样的，因为涉及到重排

```
void insert (InputIterator first, InputIterator last);
```

插入迭代器范围内键值对

几种插入方法如下面的例子所示：

```
#include <iostream>
#include <map>

int main()
{
    std::map<char, int> mymap;

    // （1）插入单个值
    mymap.insert(std::pair<char, int>('a', 100));
    mymap.insert(std::pair<char, int>('z', 200));
    mymap.insert(std::make_pair('f', 300)); // pair方式和make_pair功能是一样的

    // 返回插入位置以及是否插入成功
    std::pair<std::map<char, int>::iterator, bool> ret;
    ret = mymap.insert(std::pair<char, int>('z', 500));
    if (ret.second == false) {
        std::cout << "element 'z' already existed";
        std::cout << " with a value of " << ret.first->second << '\n';
    }

    // （2）指定位置插入
    std::map<char, int>::iterator it = mymap.begin();
    mymap.insert(it, std::pair<char, int>('b', 300));  //效率更高
    mymap.insert(it, std::pair<char, int>('c', 400));  //效率非最高

    // （3）范围多值插入
    std::map<char, int> anothermap;
    anothermap.insert(mymap.begin(), mymap.find('c'));

    // （4）列表形式插入
    anothermap.insert({ { 'd', 100 }, {'e', 200} });

    return 0;
}
```

3.4 erase() 删除元素

从map中删除元素的函数是erase()，该函数有如下的三种形式：

```
size_t erase( const key_type& key );
根据key来进行删除， 返回删除的元素数量，在map里结果非0即1
iterator erase( iterator pos )
删除迭代器指向位置的键值对，并返回一个指向下一元素的迭代器
iterator erase( const_iterator first, const_iterator last );
删除一定范围内的元素，并返回一个指向下一元素的迭代器
#include <iostream>
#include <map>
using namespace std;

int main() {
    map<int, int> mymap;
    for (int i = 0; i < 20; i++) {
        mymap.insert(make_pair(i, i));
    }
    mymap.erase(0);               //  （1）删除key为0的元素
    mymap.erase(mymap.begin());   //  （2）删除迭代器指向的位置元素
    map<int, int>::iterator it;
    for (it = mymap.begin(); it != mymap.end(); it++) {
        cout << it->first << "==>" << it->second << endl;
    }
    return 0;
}
3.5 count(k) 查找关键字k出现的次数
size_type count (const key_type& k) const;
mymap.count(1);      // 查找关键字1在容器map中出现的次数，如果不存在则为0
1
3.6 find(k) 查找元素
iterator find (const key_type& k);
const_iterator find (const key_type& k) const;
若存在，返回指向该key的迭代器
若不存在，则返回迭代器的尾指针，即 mymap.end()，即 -1

#include <iostream>
#include <map>
using namespace std;

int main() {
    map<int, int> mp;
    for (int i = 0; i < 20; i++) {
        mp.insert(make_pair(i, i));
    }

    if (mp.count(0)) {
        cout << "yes!" << endl;
    } else {
        cout << "no!" << endl;
    }

```

```
    map<int, int>::iterator it_find;
    it_find = mp.find(0);
    if (it_find != mp.end()) {
        it_find->second = 20;
    } else {
        cout << "no!" << endl;
    }

    map<int, int>::iterator it;
    for (it = mp.begin(); it != mp.end(); it++) {
        cout << it->first << " ==> " << it->second;
    }
    return 0;
}
```

3.7 lower_bound(k) 返回关键字>=k的元素的第一个位置(是一个迭代器)

```
iterator lower_bound (const key_type& k);
const_iterator lower_bound (const key_type& k) const;
c.lower_bound(k)
```

1

3.8 upper_bound(k) 返回关键字>k的元素的第一个位置(是一个迭代器)

```
iterator upper_bound (const key_type& k);
const_iterator upper_bound (const key_type& k) const;
c.upper_bound(k)
```

1

注意：lower_bound 和 upper_bound 不适用与无序容器

3.9 equal_range() 返回一个迭代器pair，表示关键字 == k的元素的范围。若k不存在，pair的两个成员均等于c.end()

```
pair<const_iterator,const_iterator> equal_range (const key_type& k) const;
pair<iterator,iterator> equal_range (const key_type& k);
#include <iostream>
#include <map>
using namespace std;

int main() {
    map<char, int> mymap;
    mymap['a'] = 3;
    mymap['b'] = 4;
    mymap['c'] = 5;
    mymap['d'] = 6;

    cout << mymap.lower_bound('c')->first << endl;  // 返回key >= 'c'第一个元素的迭代器
    cout << mymap.upper_bound('c')->first << endl;  // 返回key > 'c'第一个元素的迭代器

    pair<map<char, int>::iterator, map<char, int>::iterator> ret;
    ret = mymap.equal_range('c');
```

```
    cout << "lower bound points to: ";
    cout << ret.first->first << " => " << ret.first->second << '\n';

    cout << "upper bound points to: ";
    cout << ret.second->first << " => " << ret.second->second << '\n';
    return 0;
}
/*
c
d
lower bound points to: c => 5
upper bound points to: d => 6
*/
3.10 empty() 容器是否为空
mymap.enpty();
1
3.11 clear() 清空容器
mymap.clear();
1
3.12 size() 容器的大小
mymap.size();
1
3.13 max_size() 容器可以容纳的最大元素个数
mymap.max_size();
1
3.14 swap() 交换两个map
A.swap(B);
1
3.15 begin()          返回指向map头部的迭代器
3.16 end()            返回指向map末尾的迭代器
3.17 rbegin()         返回一个指向map尾部的逆向迭代器
3.18 rend()           返回一个指向map头部的逆向迭代器
3.19 关联容器额外的类型别名


3.20 key_comp() 比较key_type值大小
// 比较两个关键字在map中位置的先后
key_compare key_comp() const;
1
2
map<char,int> mymap;
map<char,int>::key_compare mycomp = mymap.key_comp();

mymap['a']=100;
mymap['b']=200;
mycomp('a', 'b');  // a排在b前面，因此返回结果为true

3.21 value_comp() 比较value_type值大小
```

```cpp
#include <iostream>
#include <map>

int main() {
    std::map<char, int> mymap;

    mymap['x'] = 1001;
    mymap['y'] = 2002;
    mymap['z'] = 3003;

    std::cout << "mymap contains:\n";
    std::pair<char, int> highest = *mymap.rbegin();  // last element
    std::map<char, int>::iterator it = mymap.begin();
    do {
        std::cout << it->first << " => " << it->second << '\n';
    } while (mymap.value_comp()(*it++, highest));   // 注意这里只会比较value_type中的key

    return 0;
}
```

4. map遍历

4.1 使用迭代器遍历

```cpp
#include <iostream>
#include <string>
#include <map>
using namespace std;
int main() {
    map<string, int> word_count;
    string word;
    while (cin >> word && word != "-1")      // 统计每个单词出现的次数
        word_count[word]++;

    // 使用迭代器遍历
    map<string, size_t>::iterator iter;
    for (iter = word_count.begin(); iter != word_count.end(); iter++)
{
        cout << iter->first << " occurs " << iter->second
             << ((iter->second) > 1 ? " times" : " time") << endl;
    }

    // 当key是int类型的话，还可以使用下标迭代访问
    return 0;
}
```

4.2 使用下标访问

```cpp
// easy to understand
#include <map>
#include <vector>
#include <cstdio>
#include <cstring>
```

```cpp
#include <iostream>
#include <algorithm>
using namespace
std;const int maxn = 100001;
int n, m, num, cnt[maxn];
string s;
map<string,vector<int> >a;
int main() {
    std::ios::sync_with_stdio(false);
    cin>>n;
    for(int i = 1; i <= n; i++)
    {
        cin>>num;
        for(int j = 1; j <= num; j++)
        {
            cin>>s;
            a[s].push_back(i);
        }
    }
    cin>>m;
    for(int i = 1; i <= m; i++)
    {
        cin>>s;
        memset(cnt,0,sizeof(cnt));
        for(int j = 0; j < a[s].size(); j++)
        if(cnt[a[s][j]] == 0)
        {
            cout<<a[s][j]<<" ";
            //cout<<"  "<<cnt[a[s][j]]<<endl;
            cnt[a[s][j]]++;
            //cout<<"  "<<cnt[a[s][j]]<<endl;
        }
        cout<<endl;
    }
    return 0;
}
#include<iostream>
#include<cstdio>
#include<map>
using namespace std;
map<int,map<int ,int>> arr;
int main(){
    int n,q;
    cin>>n>>q;
    while(q--){
        int flag;
        cin>>flag;
        if(flag==1){
            int a,b;
```

```
464                 cin>>a>>b;
465                 int k;
466                 cin>>k;
467                 if(k==0)
468                     arr[a].erase(b);
469                 else
470                     arr[a][b]=k;
471                 //cout<<arr[a][b]<<endl;
472             }
473             if(flag==2){
474                 int a,b;
475                 cin>>a>>b;
476                 cout<<arr[a][b]<<endl;
477             }
478         }
479         return 0;
480     }
```

merge

```
1  #include<iostream>
2  #include<cstdio>
3  #include<algorithm>
4  #include<string>
5  #include<cstring>
6  #include<queue>
7  #include<stack>
8  #include<cmath>
9  #include<set>
10 #include<map>
11 using namespace std;
12 #define ll long long
13 #define lson l,m,rt<<1
14 #define rson m+1,r,rt<<1|1
15 typedef pair<int,int>P;
16 const int INF=0x3f3f3f3f;
17 const int N=100005;
18 vector<int>a,b,c;
19
20 int main(){
21     a.push_back(1);
22     a.push_back(3);
23     a.push_back(5);
24     a.push_back(7);
25
26     b.push_back(2);
27     b.push_back(4);
28     b.push_back(6);
29     b.push_back(8);
```

```
    c.resize(8);
    merge(a.begin()/*a*/,a.end()/*a+n*/,b.begin(),b.end(),c.begin());
    for(int i=0;i<c.size();i++){
        printf("%d ",c[i]);
    }
    printf("\n");
    for(int i=0;i<a.size();i++){
        printf("%d ",a[i]);
    }
    printf("\n");
    for(int i=0;i<b.size();i++){
        printf("%d ",b[i]);
    }
    printf("\n");
}
```

queue

```
#include<iostream>
#include<cstdio>
#include<queue>
#include<cmath>
using namespace std;
int arr[1024];
queue<int> ranki;
int main(){
    int n;
    cin>>n;
    for(int i=1;i<=pow(2,n);i++){
        cin>>arr[i];
        ranki.push(i);
    }
    while(ranki.size()>2){
        int a=ranki.front();
        ranki.pop();
        int b=ranki.front();
        ranki.pop();
        int mid=arr[a]>arr[b]?a:b;
        ranki.push(mid);
    }
    int a=ranki.front();
    ranki.pop();
    int b=ranki.front();
    ranki.pop();
    int mid=arr[a]<arr[b]?a:b;
    cout<<mid<<endl;
    return 0;
}

```

```
push(x)       将x压入队列的末端 pop() 弹出队列的第一个元素(队顶元素)，注意此函数并不返回任何值 front() 返回第一个元素(队顶元素)
back()        返回最后被压入的元素(队尾元素) empty() 当队列为空时，返回true
size() 返回队列的长度

#include <queue>
queue<int>q;
struct node { int x, y; }; queue<node>q;
```

set

```
#include<cstdio>
#include<iostream>
#include<set>
using namespace std;
set<int> s;
int main(){
    int t;
    cin>>t;
    while(t--){
        int n;
        int mid;
        cin>>n;
        s.clear();
        int sum=0;
        for(int i=0;i<n;i++){
            cin>>mid;
            s.insert(mid);
            if(sum!=s.size()){
                cout<<mid<<' ';
                sum=s.size();
            }
        }
        cout<<endl;
    }
    return 0;
}

begin()              返回set容器第一个元素的迭代器
end()                返回一个指向当前set末尾元素的下一位置的迭代器.
clear()              删除set容器中的所有的元素
empty()              判断set容器是否为空
max_size()           返回set容器可能包含的元素最大个数
size()               返回当前set容器中的元素个数
rbegin()             返回的值和end()相同
rend()               返回的值和begin()相同
find()               返回给定值值得定位器，如果没找到则返回end()。
count()              用来查找set中某个某个键值出现的次数。这个函数在set并不是很实用，因为一个键值在set只可能出现0或1次，
```

```
                    这样就变成了判断某一键值是否在set出现过了。
erase(iterator)     删除定位器iterator指向的值
erase(first,second) 删除定位器first和second之间的值
erase(key_value)    删除键值key_value的值

#include <iostream>
#include <set>
using namespace std;
int main(){
     set<int> s;
     set<int>::const_iterator iter;
     set<int>::iterator first;
     set<int>::iterator second;
     for(int i = 1 ; i <= 10 ; ++i)
     {
         s.insert(i);
     }
     //第一种删除
     s.erase(s.begin());
     //第二种删除
     first = s.begin();
     second = s.begin();
     second++;
     second++;
     s.erase(first,second);
     //第三种删除
     s.erase(8);
     cout<<"删除后 set 中元素是 : ";
     for(iter = s.begin() ; iter != s.end() ; ++iter)
     {
         cout<<*iter<<" ";
     }
     cout<<endl;
     return 0;
}
```

stack

```
#include<iostream>
#include<stack>
#include<cstdio>
using namespace std;
stack <int> sta;
int a[100005],b[100005];
int main(){
    int q;
    cin>>q;
    while(q--){
        int n;
```

```
        cin>>n;
        for(int i=1;i<=n;i++){
            cin>>a[i];
        }
        for(int i=1;i<=n;i++){
            cin>>b[i];
        }
        int count=1;
        for(int i=1;i<=n;i++){
            sta.push(a[i]);
            while(sta.top()==b[count]){
                sta.pop();
                count++;
                if(sta.empty()){
                    break;
                }
            }
        }
        if(sta.empty()){
            cout<<"Yes"<<endl;
        }
        else
            cout<<"No"<<endl;
        while(!sta.empty()){
            sta.pop();
        }
    }
    return 0;
}


s.push(10)      入栈
s.pop()         出栈
empty()         堆栈为空则返回真
size()          返回栈中元素数目
top()           返回栈顶元素
```

单调栈

```
#include<stack>
#include<iostream>
#include<cstdio>
using namespace std;
const int N=3e6+5;
int a[N],n,b[N];
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);//使cin 与 cout 加速，提高效率
```

```
    cin >> n;
    for(int i=1;i<=n;i++)
        cin >> a[i];
    stack<int> s;
    // if(s.empty()){
    //     cout<<"empty"<<endl;
    // }
    for(int i=n;i>=1;i--)
    {
        while(!s.empty()&&a[i]>=a[s.top()])
            s.pop();
        b[i]=s.empty()?0:s.top();
        s.push(i);
    }
    for(int i=1;i<=n;i++)
        printf("%d ",b[i]);
    return 0;
}



//std::ios::sync_with_stdio(false)加速
#include <map>
#include <vector>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
const int maxn = 100001;
int n, m, num, cnt[maxn];
string s;
map<string,vector<int> >a;
int main()
{
    std::ios::sync_with_stdio(false);
    cin>>n;
    for(int i = 1; i <= n; i++)
    {
        cin>>num;
        for(int j = 1; j <= num; j++)
        {
            cin>>s;
            a[s].push_back(i);
        }
    }
    cin>>m;
    for(int i = 1; i <= m; i++)
    {
```

```
        cin>>s;
        memset(cnt,0,sizeof(cnt));
        for(int j = 0; j < a[s].size(); j++)
        if(cnt[a[s][j]] == 0)
        {
            cout<<a[s][j]<<" ";
            //cout<<"  "<<cnt[a[s][j]]<<endl;
            cnt[a[s][j]]++;
            //cout<<"  "<<cnt[a[s][j]]<<endl;
        }
        cout<<endl;
    }
    return 0;
}
```

单源最短路径

```
单源最短路径（弱化）：
//SPFA
#include<cstdio>
using namespace std;
int dis[500010],n,m,f,g,w,t=1,s,q[200000],h,st[500010],tot;
bool vis[100010];
/*
st[i]            表示到达i点的最后一条边的编号；
dis[i]           表示从起点到i点目前为止的最短距离；
vis[i]=true      表示点i在队列里；
vis[i]=false     表示点i不在队列里；
数组q是队列；
h是指队列的头指针，t是指队列的尾指针；
*/
struct node //不开结构体，变量有点乱，所以开结构体；
{
    int to;//这条边连接的终点；
    int v;//这条边的长度；
    int last;//前一条边；
}e[500010];
void add(int from,int to,int val)
/*
val表示传入的这条边的权值；
from表示传入的这条边的起点；
to表示传入的这条边的终点；
*/
{
    tot++;//表示当前这条边的编号；
    e[tot].to=to;//更新当前这条边的终点；
    e[tot].v=val;//更新当前这条边的长度（权）；
    e[tot].last=st[from];
    /*
```

```
        e[tot].last表示的是当前读入的这条边的上一条边；
        st[from]表示的是 到当前读入的边为止（不包括这条边） 的上一条边的序号；
    */
    st[from]=tot;//更新为 当前读入的边为止（包括这条边） 的上一条边的序号；
}
void SPFA()//核心最短路；
{
    while(h<=t)//队列不为空；
    {
        h++;
        int u=q[h];//取出队首
        vis[u]=0;//队首出队
        for(int i=st[u]; i!=0; i=e[i].last)
        /*
        st[u]是指可以到达点u的上一条边，如果存在st[u]（也就是st[u]不为0，因为0是初始值），
        说明有一条边可以到达点u。所以i变成st[u]。e[i].last是指可以到达st[u]这条边的起点的边的编号
        */
        {
            int v=e[i].to;
            if(dis[v]>dis[u]+e[i].v)
            /*
            如果从起点到v的距离大于从起点到点i，再从点i到点u，再从点u到点v的距离，更新从起点
            到点v的最短路；
            */
            {
                dis[v]=dis[u]+e[i].v;
                if(vis[v]==0)//没有入过队就入队；
                {
                    vis[v]=1;//标志改为1，表示已经入队；
                    t++;
                    q[t]=v;
                }
            }
        }
    }
}



int main() //主程序；
{
    scanf("%d %d %d",&n,&m,&s);
    for(int i=1;i<=m;i++)
    {
        scanf("%d %d %d",&f,&g,&w);//输入这条边的起点f，终点g和长度w；
        add(f,g,w);//建图；
```

```
    }
    for(int i=1;i<=n;i++)//因为题目要我们求一个点到其余点的最短路;
        dis[i]=2147483647;//所以初始化全部赋为int的最大值;
    dis[s]=0;//起点到本身的的距离为0;
    q[t]=s;//起点入队;
    vis[s]=true;//标志改为true表示起点已入队;
    SPFA(); //运行最短路;
    for(int i=1;i<=n;i++)//循环输出答案;
        printf("%d ",dis[i]);
    return 0;
}
```

堆

```
堆：(stl模板)
#include <iostream>
#include <algorithm>
#include <vector>
#include <cstdio>
using namespace std;
// vector<int> get_input()
// {
//  vector<int> nums;
//  for(int i=0; i<n; i++)
//  {
//      int x;
//      cin>>x;
//      nums.insert(nums.end(), x);
//  }
//  return nums;
// }

// void out(vector<int> nums)
// {
//  for(int i=0; i<nums.size(); i++)
//  {
//      cout<<nums[i]<<" ";
//  }
//  cout<<endl;
// }

int main()
{
    int t;
    cin>>t;
    vector<int> nums;//创建堆
    while(t--){
        int n;
        cin>>n;
```

```
        if(n==1){
            int x;
            cin>>x;
            //make_heap(nums.begin(), nums.end(), greater<int>());//创建堆
            nums.push_back(x); //加入元素
            push_heap(nums.begin(), nums.end(), greater<int>());//将该元素插入堆
            //out(nums);
        }
        if(n==2){
            cout<<nums[0]<<endl;
            //out(nums);
        }
        if(n==3){
            //make_heap(nums.begin(), nums.end(), greater<int>());
            pop_heap(nums.begin(), nums.end(), greater<int>());//将元素交换到最后
            nums.pop_back();//移除该元素
            //out(nums);
        }
    }
    return 0;
}

判断序列是否为一个堆：
is_heap(param1, param2, param3);
param 1      序列式容器的起始地址
param 2      序列式容器的结束地址
param 3      比较函数，默认使用less<>，判断是否满足大顶堆
堆排序：
sort_heap(param1, param2, param3);
sort_heap(nums.begin(), nums.end(), greater<int>());
（小顶堆排序完为降序，大堆排序完为升序）
```

二叉树深度

```
#include <iostream>
#include <algorithm>
#include <cstdio>
using namespace std;
#define maxn 1000010
struct node
{
    int l, r;
} tree[maxn];
int ans;
void dfs(int u, int deep)
{
```

```
    //cout<<"u="<<u<<" deep="<<deep<<endl;
    if (u == 0)
        return;
    ans = max(ans, deep);
    //cout<<"ans="<<ans<<endl;
    //cout<<1<<endl;
    dfs(tree[u].l, deep + 1);
    //cout<<2<<endl;
    dfs(tree[u].r, deep + 1);
    return;
}

int main()
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> tree[i].l >> tree[i].r;
    dfs(1, 1);
    cout << ans;
    return 0;
}
```

归并排序

```
#include<iostream>
#include<cstdio>
using namespace std;
int arr[10000];

void arry_add(int arry[], int left, int mid, int right) {
    //cout << left << ' ' << mid << ' ' << right << endl;
    int tmp[right - left + 1];
    if(left >= right)
        return ;
    int i = left, j = mid + 1, k = 0;
    while(i <= mid && j <= right) {
        if(arry[i] <= arry[j]) {
            tmp[k++] = arry[i++];
        } else {
            tmp[k++] = arry[j++];
        }
    }
    while(i <= mid) {
        tmp[k++] = arry[i++];
    }
    while(j <= right) {
        tmp[k++] = arry[j++];
    }
```

```
    for(i = 0; i < k; i++) {
        arry[i + left] = tmp[i];
    }
}

void merge_sort(int arry[], int left, int right) {
    if(left >= right) return ;
    int mid = (left + right) >> 1;
    // cout << left << ' ' << mid << ' ' << right << endl;
    merge_sort(arry, left, mid);
    merge_sort(arry, mid + 1, right);
    arry_add(arry, left, mid, right);
}

int main() {
    int x;
    cin>>x;
    for(int i=1;i<=x;i++){
        cin>>arr[i];
    }
    merge_sort(arr,1,x);
    for(int i = 1; i <= x; i++) {
        cout << arr[i] << ' ';
    }
    cout << endl;
}
```

回溯模板（自然数拆分问题）

```
#include<cstdio>
#include<iostream>
using namespace std;
int n;
int ans;
int a[10];
int cnt;
void dfs(int c, int sum){
    //printf("c=%d sum=%d\n",c,sum);
    if (sum == n) {
        for (int i = 0; i < cnt; i++) {
            if (i) printf("+");
            printf("%d", a[i]);
        }
        printf("\n");
//        ans++;
        return;
    }
    for (int i = 1; i <= n; i++) {
        if (c <= i && sum + i <= n) {
```

```
            //printf("c=%d,sum=%d\n",c,sum);
            a[cnt++] = i;
            //printf("a[%d]=%d\n",cnt-1,a[cnt-1]);
            dfs(i, sum + i);
            //printf("1\n");
            cnt--;
        }
    }
}

int main() {
    cin>>n;
    for (int i = 1; i < n; i++) {
        cnt = 0;
        a[cnt++] = i;
        //printf("-dfs\n");
        //printf("a[%d]=%d\n",cnt-1,a[cnt-1]);
        dfs(i, i);
        //printf("dfs\n");
    }
//    printf("%d\n", ans);  //加法式子总数
    return 0;
}
```

快速幂

```
#include<iostream>
#include<cstdio>
using namespace std;
typedef long long LL;
int fastPow(LL a,LL n,LL p) {
    LL ans=1;
    while(n){
        if(n&1) ans=ans*a%p;
        a=a*a%p;
        n>>=1;
    }
    return ans;
}
int main() {
    LL b,p,k;
    cin>>b>>p>>k;
    cout<<b<<"^"<<p<<" mod "<<k<<"="<<fastPow(b,p,k)%k<<endl;
    return 0;
}
```

逆序对

```cpp
#include<bits/stdc++.h>
using namespace std;
long long a[500005],r[500005],ans=0,n;
void msort(int s,int t)
{
    if(s==t) return ;
    long long mid=(s+t)/2;
    msort(s,mid);
    msort(mid+1,t);
    int i=s,j=mid+1,k=s;
    //printf("i=%d,j=%d,k=%d\n",i,j,k);
    while(i<=mid&&j<=t){
        if(a[i]<=a[j])
            r[k]=a[i],k++,i++;
        else{
            r[k]=a[j],k++,j++;
            ans+=mid-i+1;
        }
        //printf("r[k]=%d\n",r[k-1]);
    }
    while(i<=mid)
        r[k]=a[i],k++,i++;
    while(j<=t)
        r[k]=a[j],k++,j++;
    for(int i=s;i<=t;i++)
        a[i]=r[i];//printf("a[i]=%d\n",a[i])
}
int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
        scanf("%lld",&a[i]);
    msort(1,n);
    printf("%lld\n",ans);
    return 0;
}
```

转换为二进制

```cpp
#include <iostream>
#include <bitset>
using namespace std;

void print_bin_aux(int a) {    //辅助函数
    if (a == 0) return ;
    print_bin_aux(a / 2);
    cout << (a % 2);
```

```
}

void print_bin(int a) {
    if (a == 0)
        cout << 0;
    else
        print_bin_aux(a);
    cout << endl;
}                           //转化为二进制函数；

class Solution {
public:
    int rangeBitwiseAnd(int left, int right) {
        cout << bitset<70>(left) << endl;      //C++中输出70位的二进制的方法，不过比赛不一定能用；

        print_bin(left);
        print_bin(0);
        print_bin(0xff);
        print_bin(0xaa);

        return -1;
    }
};

int main()
{
    int left = 4, right = 6;
    cout << Solution().rangeBitwiseAnd(left, right) << endl;
    return 0;
}
```

二叉树问题

```
#include<cmath>
#include<cstdio>
#include<cstring>
#include<iostream>
#include<algorithm>
using namespace std;

int n;
int fa[101],root[101],son[101];

int depth[101],width[101];
int lca(int x,int y){
    if(x == y){
        return x;
    }
```

```
    else if(depth[x] == depth[y]){
        return lca(fa[x],fa[y]);
    }
    else if(depth[x] < depth[y]){
        return lca(x,fa[y]);
    }
    else{
        return lca(fa[x],y);
    }
}

int main(){
    cin >> n;
    depth[1] = 1;
    for(int i = 1;i < n;i++){
        cin >> root[i] >> son[i];
        depth[son[i]] = depth[root[i]] + 1;
        fa[son[i]] = root[i];
    }
    int x,y;
    cin >> x >> y;
    int max_depth = 1;
    for(int i = 1;i <= n;i++){
        max_depth = max(max_depth,depth[i]);
        width[depth[i]]++;
    }
    cout << max_depth << endl;
    int max_width = 1;
    for(int i = 1;i <= n;i++){
        max_width = max(max_width,width[i]);
    }
    cout << max_width << endl;
    int k = lca(x,y);       //求 LCA 的结点序号
    cout << (depth[x] - depth[k]) * 2 + depth[y] - depth[k];       //求距离
    return 0;
}
```

algorithm常用的函数

```
//  max、min和abs函数，下面是具体的代码：

#include<stdio.h>
#include<algorithm>
using namespace std;
int main(void){
    int x = 1, y = -2;
    printf("%d %d\n", max(x, y), min(x, y));
    printf("%d %d\n", abs(x), abs(y));
```

```
    return 0;
}
// swap函数，下面是具体的代码：

#include<stdio.h>
#include<algorithm>
using namespace std;
int main(void){
    int x = 1, y = -2;
    swap(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
// reverse函数
// reverse（it，it2）可以将数组指针在[it，it2）之间的元素或者容器的迭代器在[it，it2）范围内的元素进行翻转，具体的代码如下：

#include<stdio.h>
#include<algorithm>
using namespace std;
int main(void){
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 7};
    reverse(arr, arr + 8);
    for(int i = 0; i < 8; ++i){
        printf("%d ", arr[i]);
    }
    return 0;
}

// next_permutation函数
// 使用这个函数可以生成给定序列在全排列中的下一个排列，并且生成的排列是从小到大进行排序的，具体的代码如下：


// next_permutation -> 升序
// prev_permutation -> 降序

#include<stdio.h>
#include<algorithm>
using namespace std;
int main(void){
    int arr[] = {1, 2, 3, 4};
    //使用C++中的net_permutation函数生成数组的全排列
    do{
        printf("%d%d%d%d\n", arr[0], arr[1], arr[2], arr[3]);
    }while(next_permutation(arr, arr + 4));
    return 0;
}

```

```
57
58  // fill函数
59  // fill函数可以将数组或者是容器中的某一区间赋予某个相同的值，与memset函数不同的是，这里的赋值可以是数组类型对应范围中的任意值，具体的代码如下：
60
61  #include<stdio.h>
62  #include<algorithm>
63  using namespace std;
64  int main(void){
65      int arr[] = {1, 2, 3, 4, 5, 6, 7, 7};
66      fill(arr, arr + 8, 100);
67      for(int i = 0; i < 8; ++i){
68          printf("%d ", arr[i]);
69      }
70      return 0;
71  }
```