

copy

```
1  #include <iostream>
2  #include <algorithm>
3  #include <vector>
4
5      using namespace std;
6
7      int main () {
8          int myints[] = {10, 20, 30, 40, 50, 60, 70};
9          vector<int> myvector;
10         vector<int>::iterator it;
11
12         myvector.resize(7);    // 为容器myvector分配空间
13
14         //copy用法一:
15         //将数组myints中的七个元素复制到myvector容器中
16         copy ( myints, myints+7, myvector.begin() );
17
18         cout << "myvector contains: ";
19         for ( it = myvector.begin(); it != myvector.end(); ++it )
20         {
21             cout << " " << *it;
22         }
23         cout << endl;
24         //copy用法二:
25         //将数组myints中的元素向左移动一位
26         int temp=myints[0];
27         copy(myints + 1, myints + 7, myints);
28         myints[6]=temp;
29         cout << "myints contains: ";
30         for ( size_t i = 0; i < 7; ++i )
31         {
32             cout << " " << myints[i];
33         }
34         cout << endl;
35
36         return 0;
37     }
```

deque

- 1 deque概念
- 2 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。
- 3 创建deque

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

```

1 //在deque中查找某元素
2 deque<int>::iterator pos=find(c.begin(),c.end(),num);
3 if(pos!=c.end()) printf("find success\n");
4 else printf("find failed\n");
5 //反向遍历队列中的元素:
6 deque<int>::reverse_iterator rit;
7 for(rit = c.rbegin();rit != c.rend(); rit++)
8     printf("%d ",*rit);
9 #include <iostream>
10 #include <deque>
11 #include <cstdio>
12
13 using namespace std;

```

```

14
15 int nums[4000001];
16
17 int main() {
18     int n, m, left_result = 0, right_result = 0, sum_result = 0, left
= 1, sum = 0;
19     cin >> n >> m;
20     for (int i = 1; i <= n; i++) {
21         cin >> nums[i];
22     }
23     deque<int> queue;
24     for (int right = 1; right <= n; right++) {
25         //cout<<"right="<<right<<" left="<<left<<endl;
26         queue.push_back(nums[right]);
27         //cout<<"front="<<queue.front()<<endl;
28         //cout<<"back="<<queue.back()<<endl;
29         sum += nums[right];
30         while (sum > m) {
31             queue.pop_front();
32
33             sum -= nums[right];
34
35             if (sum > sum_result) {
36                 sum_result = sum;
37                 right_result = right - 1;
38                 left_result = left;
39             }
40             sum += nums[right];
41             sum -= nums[left];
42
43             left++;
44         }
45     }
46     if (sum > sum_result) {
47         sum_result = sum;
48         right_result = n;
49         left_result = left;
50     }
51     cout << left_result << " " << right_result << " " << sum_result;
52     return 0;
53 }

```

list

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

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

```

1  #include <iostream>
2  #include <cstdio>
3  #include <list>
4  #include <iterator> // std::advance distance
5  using namespace std;
6  list<int> arr;
7
8  // int* getHead(int* arr) { return arr; }
9
10 int main()
11 {
12     int n, m, sum = 1;
13     cin >> n >> m;
14     for (int i = 1; i <= n; i++)
15     {
16         arr.push_back(i);
17     }
18     // for(list<int>::const_iterator iter = arr.begin(); iter !=
arr.end(); iter++)
19     // {
20     //     cout<<*iter<<' ';
21     // }
22
23     // int arr2[] {1, 2, 3};
24     // int* ptr = arr2;
25     // int* ptr2 = &arr2[0];

```

```

26     // int* ptr3 = getHead(arr2);
27     // cout << *ptr << endl;
28     // *(arr2 + 2)
29
30     list<int>::iterator iter = arr.begin(); // bidirectional iterator
双向迭代器
31     // iterator category 迭代器类型
32
33     // forward iterator 前向迭代器  std::forward_list
34     // bidirectional iterator 双向迭代器  std::list std::set std::map
35     // random access iterator 随机访问迭代器  std::vector std::deque
36     for (int i = m - 1; !arr.empty(); arr.size() != 0 && (i = (i + m -
1) % arr.size()))
37     {
38         auto itr = arr.begin();
39         advance(itr, i); // advance for(int i=0; i<i; i++){ iter++}
40         // distance(arr.begin(), itr);
41         if (arr.size() != n) cout << ' ';
42         cout << *itr;
43         arr.erase(itr);
44     }
45
46     cout << endl;
47     return 0;
48 }
49
50
51
52 #include <iostream>
53 #include <list>
54 #include <iterator>
55 #include <algorithm>
56 using namespace std;
57
58 int main()
59 {
60     list<int> lst {1, 2, 3};
61     auto begin = lst.begin();
62     // begin++;
63     // begin = lst.erase(begin);
64     // cout << *begin << endl;
65
66     // copy(lst.begin(), lst.end(), ostream_iterator<int>(cout, " "));
67     // cout << endl;
68     advance(begin, 1);
69     cout << *begin << endl;
70
71     return 0;
72 }

```

map

```
1  #include<iostream>
2  #include<cstdio>
3  #include<map>
4  #include<string>
5  using namespace std;
6  map<string,string>fam;
7  string findd(string x){
8      while(fam[x]!=x){
9          x=fam[x];
10     }
11     return x;
12 }
13 int main(){
14     char flag;
15     string fri,name;
16     while(cin>>flag){
17         if(flag=='$'){
18             return 0;
19         }
20         if(flag=='#'){
21             cin>>fri;
22             if(fam.count(fri)==0)
23                 fam[fri]=fri;
24             //getchar();
25         }
26         if(flag=='+'){
27             cin>>name;
28             //cout<<name<<' '<<fri<<endl;
29             fam[name]=fri;
30         }
31         if(flag=='?'){
32             string seek,seek_fri;
33             cin>>seek;
34             seek_fri=findd(seek);
35             cout<<seek<<' '<<seek_fri<<endl;
36         }
37     }
38 }
```

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。

45 标准卡提供 8 个关联容器

46 2. pair 类型

47 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。

48

49 2.1 pair 类型的定义和初始化

50 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:

51

52 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

53 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。

54 pair<T1, T2> p = {v1, v2} 等价于p(v1, v2)

55 make_pair(v1, v2) 以v1和v2值创建的一个新的pair对象

56

57 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
一个成员的值:

58

59 p.first 返回p的名为 first 的(公有)数据成员

60 p.second 返回p的名为second的(公有)数据成员

61 p1 relop p2 关系运算符 (<、>、<=、>=) 按字典序定义。

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

63 p1 < p2 为 true。关系运算利用元素的 < 运算符来实现

64

65 p1 == p2 当 first 和 second 成员分别相等时,两个pair相等

66 p1 != p2 若不能达到以上要求,则不相等

67 例如:

68

69 #include <stdio.h>

70 #include <string.h>

71 #include <string>

72 #include <utility>

73 using namespace std;

74

75 int main(){

76 pair<int, string> p1(0, "Hello");

77 printf("%d, %s\n", p1.first, p1.second.c_str());

78 pair<int, string> p2 = make_pair(1, "World");

79 printf("%d, %s\n", p2.first, p2.second.c_str());

80 return 0;

81 }

82

83

84 3. map基本操作

85 3.1 头文件

```

86 #include <map>
87
88 3.2 创建map对象
89 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：
90
91 map<k, v> m;           定义了一个名为m的空map对象
92 map<k, v> m2(m);       创建了m的副本m2
93 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
94 map 的 value_type 是存储元素的键以及值的pair类型，键为 const。
95
96 map<int, char> m;       // 定义了一个名为m的空map
97 map<int, char> m2(m);   // 创建了m的副本m2
98 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
99

```

100 3.3 map元素访问

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

102 103 3.3.1 使用下标 [] 访问

```

104 #include <iostream>
105 #include <map>
106 #include <string>
107
108 int main() {
109     std::map<char, std::string> mymap;
110
111     mymap['a'] = "an element";
112     mymap['b'] = "another element";
113     mymap['c'] = mymap['b'];
114
115     std::cout << "mymap['a'] is " << mymap['a'] << '\n';
116     std::cout << "mymap['b'] is " << mymap['b'] << '\n';
117     std::cout << "mymap['c'] is " << mymap['c'] << '\n';
118     std::cout << "mymap['d'] is " << mymap['d'] << '\n';
119
120     std::cout << "mymap now contains " << mymap.size() << "
elements.\n";
121     return 0;
122 }
123 /*
124 mymap['a'] is an element
125 mymap['b'] is another element
126 mymap['c'] is another element
127 mymap['d'] is           // 下标访问不会进行下标检查
128 mymap now contains 4 elements.
129 */

```

130 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:

```

226 size_t erase( const key_type& key );
227 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
228 iterator erase( iterator pos )
229 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
230 iterator erase( const_iterator first, const_iterator last );
231 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
232 #include <iostream>
233 #include <map>
234 using namespace std;
235
236 int main() {
237     map<int, int> mymap;
238     for (int i = 0; i < 20; i++) {
239         mymap.insert(make_pair(i, i));
240     }
241     mymap.erase(0); // (1) 删除key为0的元素
242     mymap.erase(mymap.begin()); // (2) 删除迭代器指向的位置元素
243     map<int, int>::iterator it;
244     for (it = mymap.begin(); it != mymap.end(); it++) {
245         cout << it->first << "==" << it->second << endl;
246     }
247     return 0;
248 }
249 3.5 count(k) 查找关键字k出现的次数
250 size_type count (const key_type& k) const;
251 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
252 1
253 3.6 find(k) 查找元素
254 iterator find (const key_type& k);
255 const_iterator find (const key_type& k) const;
256 若存在，返回指向该key的迭代器
257 若不存在，则返回迭代器的尾指针，即 mymap.end()，即 -1
258
259 #include <iostream>
260 #include <map>
261 using namespace std;
262
263 int main() {
264     map<int, int> mp;
265     for (int i = 0; i < 20; i++) {
266         mp.insert(make_pair(i, i));
267     }
268
269     if (mp.count(0)) {
270         cout << "yes!" << endl;
271     } else {
272         cout << "no!" << endl;
273     }
274

```

```

275     map<int, int>::iterator it_find;
276     it_find = mp.find(0);
277     if (it_find != mp.end()) {
278         it_find->second = 20;
279     } else {
280         cout << "no!" << endl;
281     }
282
283     map<int, int>::iterator it;
284     for (it = mp.begin(); it != mp.end(); it++) {
285         cout << it->first << " ==> " << it->second;
286     }
287     return 0;
288 }
289 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)
290 iterator lower_bound (const key_type& k);
291 const_iterator lower_bound (const key_type& k) const;
292 c.lower_bound(k)
293 1
294 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)
295 iterator upper_bound (const key_type& k);
296 const_iterator upper_bound (const key_type& k) const;
297 c.upper_bound(k)
298 1
299 注意: lower_bound 和 upper_bound 不适用与无序容器
300
301 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()
302 pair<const_iterator, const_iterator> equal_range (const key_type& k)
    const;
303 pair<iterator, iterator> equal_range (const key_type& k);
304 #include <iostream>
305 #include <map>
306 using namespace std;
307
308 int main() {
309     map<char, int> mymap;
310     mymap['a'] = 3;
311     mymap['b'] = 4;
312     mymap['c'] = 5;
313     mymap['d'] = 6;
314
315     cout << mymap.lower_bound('c')->first << endl; // 返回key >=
    'c'第一个元素的迭代器
316     cout << mymap.upper_bound('c')->first << endl; // 返回key >
    'c'第一个元素的迭代器
317
318     pair<map<char, int>::iterator, map<char, int>::iterator> ret;
319     ret = mymap.equal_range('c');

```

```

320     cout << "lower bound points to: ";
321     cout << ret.first->first << " => " << ret.first->second << '\n';
322
323     cout << "upper bound points to: ";
324     cout << ret.second->first << " => " << ret.second->second <<
'\n';
325     return 0;
326 }
327 /*
328 c
329 d
330 lower bound points to: c => 5
331 upper bound points to: d => 6
332 */
333 3.10 empty() 容器是否为空
334 mymap.empty();
335 1
336 3.11 clear() 清空容器
337 mymap.clear();
338 1
339 3.12 size() 容器的大小
340 mymap.size();
341 1
342 3.13 max_size() 容器可以容纳的最大元素个数
343 mymap.max_size();
344 1
345 3.14 swap() 交换两个map
346 A.swap(B);
347 1
348 3.15 begin()          返回指向map头部的迭代器
349 3.16 end()            返回指向map末尾的迭代器
350 3.17 rbegin()         返回一个指向map尾部的逆向迭代器
351 3.18 rend()           返回一个指向map头部的逆向迭代器
352 3.19 关联容器额外的类型别名
353
354
355 3.20 key_comp() 比较key_type值大小
356 // 比较两个关键字在map中位置的先后
357 key_compare key_comp() const;
358 1
359 2
360 map<char,int> mymap;
361 map<char,int>::key_compare mycomp = mymap.key_comp();
362
363 mymap['a']=100;
364 mymap['b']=200;
365 mycomp('a', 'b'); // a排在b前面, 因此返回结果为true
366
367 3.21 value_comp() 比较value_type值大小

```

```

368 #include <iostream>
369 #include <map>
370
371 int main() {
372     std::map<char, int> mymap;
373
374     mymap['x'] = 1001;
375     mymap['y'] = 2002;
376     mymap['z'] = 3003;
377
378     std::cout << "mymap contains:\n";
379     std::pair<char, int> highest = *mymap.rbegin(); // last element
380     std::map<char, int>::iterator it = mymap.begin();
381     do {
382         std::cout << it->first << " => " << it->second << '\n';
383     } while (mymap.value_comp()(*it++, highest)); // 注意这里只会比较
value_type中的key
384
385     return 0;
386 }
387 4. map遍历
388 4.1 使用迭代器遍历
389 #include <iostream>
390 #include <string>
391 #include <map>
392 using namespace std;
393 int main() {
394     map<string, int> word_count;
395     string word;
396     while (cin >> word && word != "-1") // 统计每个单词出现的次数
397         word_count[word]++;
398
399     // 使用迭代器遍历
400     map<string, size_t>::iterator iter;
401     for (iter = word_count.begin(); iter != word_count.end(); iter++)
402     {
403         cout << iter->first << " occurs " << iter->second
404             << ((iter->second) > 1 ? " times" : " time") << endl;
405     }
406
407     // 当key是int类型的话, 还可以使用下标迭代访问
408     return 0;
409 }
409 4.2 使用下标访问
410 // easy to understand
411 #include <map>
412 #include <vector>
413 #include <cstdio>
414 #include <cstring>

```

```

415 #include <iostream>
416 #include <algorithm>
417 using namespace
418 std;const int maxn = 100001;
419 int n, m, num, cnt[maxn];
420 string s;
421 map<string,vector<int> >a;
422 int main() {
423     std::ios::sync_with_stdio(false);
424     cin>>n;
425     for(int i = 1; i <= n; i++)
426     {
427         cin>>num;
428         for(int j = 1; j <= num; j++)
429         {
430             cin>>s;
431             a[s].push_back(i);
432         }
433     }
434     cin>>m;
435     for(int i = 1; i <= m; i++)
436     {
437         cin>>s;
438         memset(cnt,0,sizeof(cnt));
439         for(int j = 0; j < a[s].size(); j++)
440             if(cnt[a[s][j]] == 0)
441             {
442                 cout<<a[s][j]<<" ";
443                 //cout<<" "<<cnt[a[s][j]]<<endl;
444                 cnt[a[s][j]]++;
445                 //cout<<" "<<cnt[a[s][j]]<<endl;
446             }
447         cout<<endl;
448     }
449     return 0;
450 }
451 #include<iostream>
452 #include<cstdio>
453 #include<map>
454 using namespace std;
455 map<int,map<int ,int>> arr;
456 int main(){
457     int n,q;
458     cin>>n>>q;
459     while(q--){
460         int flag;
461         cin>>flag;
462         if(flag==1){
463             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);

```



```

30     c.resize(8);
31     merge(a.begin()/*a*/,a.end()/*a+n*/,b.begin(),b.end(),c.begin());
32     for(int i=0;i<c.size();i++){
33         printf("%d ",c[i]);
34     }
35     printf("\n");
36     for(int i=0;i<a.size();i++){
37         printf("%d ",a[i]);
38     }
39     printf("\n");
40     for(int i=0;i<b.size();i++){
41         printf("%d ",b[i]);
42     }
43     printf("\n");
44 }

```

queue

```

1  #include<iostream>
2  #include<cstdio>
3  #include<queue>
4  #include<cmath>
5  using namespace std;
6  int arr[1024];
7  queue<int> ranki;
8  int main(){
9      int n;
10     cin>>n;
11     for(int i=1;i<=pow(2,n);i++){
12         cin>>arr[i];
13         ranki.push(i);
14     }
15     while(ranki.size()>2){
16         int a=ranki.front();
17         ranki.pop();
18         int b=ranki.front();
19         ranki.pop();
20         int mid=arr[a]>arr[b]?a:b;
21         ranki.push(mid);
22     }
23     int a=ranki.front();
24     ranki.pop();
25     int b=ranki.front();
26     ranki.pop();
27     int mid=arr[a]<arr[b]?a:b;
28     cout<<mid<<endl;
29     return 0;
30 }
31

```

```

32 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)
33 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
34
35 #include <queue>
36 queue<int>q;
37 struct node { int x, y; }; queue<node>q;

```

set

```

1  #include<cstdio>
2  #include<iostream>
3  #include<set>
4  using namespace std;
5  set<int> s;
6  int main() {
7      int t;
8      cin>>t;
9      while(t--){
10         int n;
11         int mid;
12         cin>>n;
13         s.clear();
14         int sum=0;
15         for(int i=0;i<n;i++){
16             cin>>mid;
17             s.insert(mid);
18             if(sum!=s.size()){
19                 cout<<mid<<' ';
20                 sum=s.size();
21             }
22         }
23         cout<<endl;
24     }
25     return 0;
26 }

```

27
 28 begin() 返回set容器第一个元素的迭代器
 29 end() 返回一个指向当前set末尾元素的下一位置的迭代器。
 30 clear() 删除set容器中的所有元素
 31 empty() 判断set容器是否为空
 32 max_size() 返回set容器可能包含的元素最大个数
 33 size() 返回当前set容器中的元素个数
 34 rbegin() 返回的值和end() 相同
 35 rend() 返回的值和begin() 相同
 36 find() 返回给定值值得定位器，如果没找到则返回end()。
 37 count() 用来查找set中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，

```

38 |                                     这样就变成了判断某一键值是否在set出现过了。
39 | erase(iterator)                    删除定位器iterator指向的值
40 | erase(first,second)               删除定位器first和second之间的值
41 | erase(key_value)                  删除键值key_value的值
42 |
43 | #include <iostream>
44 | #include <set>
45 | using namespace std;
46 | int main() {
47 |     set<int> s;
48 |     set<int>::const_iterator iter;
49 |     set<int>::iterator first;
50 |     set<int>::iterator second;
51 |     for(int i = 1 ; i <= 10 ; ++i)
52 |     {
53 |         s.insert(i);
54 |     }
55 |     //第一种删除
56 |     s.erase(s.begin());
57 |     //第二种删除
58 |     first = s.begin();
59 |     second = s.begin();
60 |     second++;
61 |     second++;
62 |     s.erase(first,second);
63 |     //第三种删除
64 |     s.erase(8);
65 |     cout<<"删除后 set 中元素是 : ";
66 |     for(iter = s.begin() ; iter != s.end() ; ++iter)
67 |     {
68 |         cout<<*iter<<" ";
69 |     }
70 |     cout<<endl;
71 |     return 0;
72 | }

```

stack

```

1 | #include<iostream>
2 | #include<stack>
3 | #include<cstdio>
4 | using namespace std;
5 | stack <int> sta;
6 | int a[100005],b[100005];
7 | int main() {
8 |     int q;
9 |     cin>>q;
10 |     while(q--){
11 |         int n;

```

```

12     cin>>n;
13     for(int i=1;i<=n;i++){
14         cin>>a[i];
15     }
16     for(int i=1;i<=n;i++){
17         cin>>b[i];
18     }
19     int count=1;
20     for(int i=1;i<=n;i++){
21         sta.push(a[i]);
22         while(sta.top()==b[count]){
23             sta.pop();
24             count++;
25             if(sta.empty()){
26                 break;
27             }
28         }
29     }
30     if(sta.empty()){
31         cout<<"Yes"<<endl;
32     }
33     else
34         cout<<"No"<<endl;
35     while(!sta.empty()){
36         sta.pop();
37     }
38 }
39 return 0;
40 }
41
42
43 s.push(10)    入栈
44 s.pop()      出栈
45 empty()      堆栈为空则返回真
46 size()       返回栈中元素数目
47 top()        返回栈顶元素

```

单调栈

```

1  #include<stack>
2  #include<iostream>
3  #include<cstdio>
4  using namespace std;
5  const int N=3e6+5;
6  int a[N],n,b[N];
7  int main()
8  {
9      ios::sync_with_stdio(false);
10     cin.tie(0);//使cin 与 cout 加速，提高效率

```

```

11     cin >> n;
12     for(int i=1;i<=n;i++)
13         cin >> a[i];
14     stack<int> s;
15     // if(s.empty()){
16     //     cout<<"empty"<<endl;
17     // }
18     for(int i=n;i>=1;i--)
19     {
20         while(!s.empty() && a[i]>=a[s.top()])
21             s.pop();
22         b[i]=s.empty()?0:s.top();
23         s.push(i);
24     }
25     for(int i=1;i<=n;i++)
26         printf("%d ",b[i]);
27     return 0;
28 }
29
30
31
32 //std::ios::sync_with_stdio(false) 加速
33 #include <map>
34 #include <vector>
35 #include <cstdio>
36 #include <cstring>
37 #include <iostream>
38 #include <algorithm>
39 using namespace std;
40 const int maxn = 100001;
41 int n, m, num, cnt[maxn];
42 string s;
43 map<string,vector<int> >a;
44 int main()
45 {
46     std::ios::sync_with_stdio(false);
47     cin>>n;
48     for(int i = 1; i <= n; i++)
49     {
50         cin>>num;
51         for(int j = 1; j <= num; j++)
52         {
53             cin>>s;
54             a[s].push_back(i);
55         }
56     }
57     cin>>m;
58     for(int i = 1; i <= m; i++)
59     {

```

```

60         cin>>s;
61         memset(cnt,0,sizeof(cnt));
62         for(int j = 0; j < a[s].size(); j++)
63             if(cnt[a[s][j]] == 0)
64             {
65                 cout<<a[s][j]<<" ";
66                 //cout<<" "<<cnt[a[s][j]]<<endl;
67                 cnt[a[s][j]]++;
68                 //cout<<" "<<cnt[a[s][j]]<<endl;
69             }
70         cout<<endl;
71     }
72     return 0;
73 }

```

单源最短路径

```

1  单源最短路径（弱化）：
2  //SPFA
3  #include<cstdio>
4  using namespace std;
5  int dis[500010],n,m,f,g,w,t=1,s,q[200000],h,st[500010],tot;
6  bool vis[100010];
7  /*
8  st[i]          表示到达i点的最后一条边的编号；
9  dis[i]         表示从起点到i点目前为止的最短距离；
10 vis[i]=true    表示点i在队列里；
11 vis[i]=false   表示点i不在队列里；
12 数组q是队列；
13 h是指队列的头指针，t是指队列的尾指针；
14 */
15 struct node //不开结构体，变量有点乱，所以开结构体；
16 {
17     int to;//这条边连接的终点；
18     int v;//这条边的长度；
19     int last;//前一条边；
20 }e[500010];
21 void add(int from,int to,int val)
22 /*
23 val表示传入的这条边的权值；
24 from表示传入的这条边的起点；
25 to表示传入的这条边的终点；
26 */
27 {
28     tot++;//表示当前这条边的编号；
29     e[tot].to=to;//更新当前这条边的终点；
30     e[tot].v=val;//更新当前这条边的长度（权）；
31     e[tot].last=st[from];
32     /*

```

```

33         e[tot].last表示的是当前读入的这条边的上一条边;
34         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;
35     */
36     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;
37 }
38 void SPFA() //核心最短路;
39 {
40     while(h<=t) //队列不为空;
41     {
42         h++;
43         int u=q[h]; //取出队首
44         vis[u]=0; //队首出队
45         for(int i=st[u]; i!=0; i=e[i].last)
46             /*
47             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），
48             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
49             */
50             {
51                 int v=e[i].to;
52                 if(dis[v]>dis[u]+e[i].v)
53                     /*
54                     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
55                     到点v的最短路;
56                     */
57                     {
58                         dis[v]=dis[u]+e[i].v;
59                         if(vis[v]==0) //没有入过队就入队;
60                         {
61                             vis[v]=1; //标志改为1，表示已经入队;
62                             t++;
63                             q[t]=v;
64                         }
65                     }
66             }
67     }
68 }
69
70
71
72 int main() //主程序;
73 {
74     scanf("%d %d %d", &n, &m, &s);
75     for(int i=1; i<=m; i++)
76     {
77         scanf("%d %d %d", &f, &g, &w); //输入这条边的起点f，终点g和长度w;
78         add(f, g, w); //建图;

```

```

79     }
80     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;
81         dis[i]=2147483647;//所以初始化全部赋为int的最大值;
82     dis[s]=0;//起点到本身的距离为0;
83     q[t]=s;//起点入队;
84     vis[s]=true;//标志改为true表示起点已入队;
85     SPFA(); //运行最短路;
86     for(int i=1;i<=n;i++)//循环输出答案;
87         printf("%d ",dis[i]);
88     return 0;
89 }

```

堆

```

1 堆: (stl模板)
2  #include <iostream>
3  #include <algorithm>
4  #include <vector>
5  #include <cstdio>
6  using namespace std;
7  // vector<int> get_input()
8  // {
9  //     vector<int> nums;
10 //     for(int i=0; i<n; i++)
11 //     {
12 //         int x;
13 //         cin>>x;
14 //         nums.insert(nums.end(), x);
15 //     }
16 //     return nums;
17 // }
18
19 // void out(vector<int> nums)
20 // {
21 //     for(int i=0; i<nums.size(); i++)
22 //     {
23 //         cout<<nums[i]<<" ";
24 //     }
25 //     cout<<endl;
26 // }
27
28 int main()
29 {
30     int t;
31     cin>>t;
32     vector<int> nums;//创建堆
33     while(t--){
34         int n;
35         cin>>n;

```



```

36         if (n==1) {
37             int x;
38             cin>>x;
39             //make_heap(nums.begin(), nums.end(), greater<int>()); //创
建堆
40             nums.push_back(x); //加入元素
41             push_heap(nums.begin(), nums.end(), greater<int>()); //将该
元素插入堆
42             //out(nums);
43         }
44         if (n==2) {
45             cout<<nums[0]<<endl;
46             //out(nums);
47         }
48         if (n==3) {
49             //make_heap(nums.begin(), nums.end(), greater<int>());
50             pop_heap(nums.begin(), nums.end(), greater<int>()); //将元素
交换到最后
51             nums.pop_back(); //移除该元素
52             //out(nums);
53         }
54     }
55     return 0;
56 }
57
58 判断序列是否为一个堆:
59 is_heap(param1, param2, param3);
60 param 1    序列式容器的起始地址
61 param 2    序列式容器的结束地址
62 param 3    比较函数, 默认使用less<>, 判断是否满足大顶堆
63 堆排序:
64 sort_heap(param1, param2, param3);
65 sort_heap(nums.begin(), nums.end(), greater<int>());
66 (小顶堆排序完为降序, 大堆排序完为升序)

```

二叉树深度

```

1  #include <iostream>
2  #include <algorithm>
3  #include <cstdio>
4  using namespace std;
5  #define maxn 1000010
6  struct node
7  {
8      int l, r;
9  } tree[maxn];
10 int ans;
11 void dfs(int u, int deep)
12 {

```

```

13     //cout<<"u="<<u<<" deep="<<deep<<endl;
14     if (u == 0)
15         return;
16     ans = max(ans, deep);
17     //cout<<"ans="<<ans<<endl;
18     //cout<<1<<endl;
19     dfs(tree[u].l, deep + 1);
20     //cout<<2<<endl;
21     dfs(tree[u].r, deep + 1);
22     return;
23 }
24
25 int main()
26 {
27     int n;
28     cin >> n;
29     for (int i = 1; i <= n; i++)
30         cin >> tree[i].l >> tree[i].r;
31     dfs(1, 1);
32     cout << ans;
33     return 0;
34 }

```

归并排序

```

1  #include<iostream>
2  #include<cstdio>
3  using namespace std;
4  int arr[10000];
5
6  void array_add(int array[], int left, int mid, int right) {
7      //cout << left << ' ' << mid << ' ' << right << endl;
8      int tmp[right - left + 1];
9      if(left >= right)
10         return ;
11     int i = left, j = mid + 1, k = 0;
12     while(i <= mid && j <= right) {
13         if(array[i] <= array[j]) {
14             tmp[k++] = array[i++];
15         } else {
16             tmp[k++] = array[j++];
17         }
18     }
19     while(i <= mid) {
20         tmp[k++] = array[i++];
21     }
22     while(j <= right) {
23         tmp[k++] = array[j++];
24     }

```

```

25     for(i = 0; i < k; i++) {
26         array[i + left] = tmp[i];
27     }
28 }
29
30 void merge_sort(int array[], int left, int right) {
31     if(left >= right) return ;
32     int mid = (left + right) >> 1;
33     // cout << left << ' ' << mid << ' ' << right << endl;
34     merge_sort(array, left, mid);
35     merge_sort(array, mid + 1, right);
36     array_add(array, left, mid, right);
37 }
38
39 int main() {
40     int x;
41     cin>>x;
42     for(int i=1;i<=x;i++){
43         cin>>arr[i];
44     }
45     merge_sort(arr,1,x);
46     for(int i = 1; i <= x; i++) {
47         cout << arr[i] << ' ';
48     }
49     cout << endl;
50 }

```

回溯模板（自然数拆分问题）

```

1  #include<cstdio>
2  #include<iostream>
3  using namespace std;
4  int n;
5  int ans;
6  int a[10];
7  int cnt;
8  void dfs(int c, int sum){
9      //printf("c=%d sum=%d\n",c,sum);
10     if (sum == n) {
11         for (int i = 0; i < cnt; i++) {
12             if (i) printf("+");
13             printf("%d", a[i]);
14         }
15         printf("\n");
16         // ans++;
17         return;
18     }
19     for (int i = 1; i <= n; i++) {
20         if (c <= i && sum + i <= n) {

```

```

21         //printf("c=%d,sum=%d\n",c,sum);
22         a[cnt++] = i;
23         //printf("a[%d]=%d\n",cnt-1,a[cnt-1]);
24         dfs(i, sum + i);
25         //printf("1\n");
26         cnt--;
27     }
28 }
29 }
30
31 int main() {
32     cin>>n;
33     for (int i = 1; i < n; i++) {
34         cnt = 0;
35         a[cnt++] = i;
36         //printf("-dfs\n");
37         //printf("a[%d]=%d\n",cnt-1,a[cnt-1]);
38         dfs(i, i);
39         //printf("dfs\n");
40     }
41     // printf("%d\n", ans); //加法式子总数
42     return 0;
43 }

```

快速幂

```

1  #include<iostream>
2  #include<cstdio>
3  using namespace std;
4  typedef long long LL;
5  int fastPow(LL a,LL n,LL p) {
6      LL ans=1;
7      while(n){
8          if(n&1) ans=ans*a%p;
9          a=a*a%p;
10         n>>=1;
11     }
12     return ans;
13 }
14 int main() {
15     LL b,p,k;
16     cin>>b>>p>>k;
17     cout<<b<<"^"<<p<<" mod "<<k<<"="<<fastPow(b,p,k)%k<<endl;
18     return 0;
19 }

```

逆序对

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  long long a[500005],r[500005],ans=0,n;
4  void msort(int s,int t)
5  {
6      if(s==t) return ;
7      long long mid=(s+t)/2;
8      msort(s,mid);
9      msort(mid+1,t);
10     int i=s,j=mid+1,k=s;
11     //printf("i=%d,j=%d,k=%d\n",i,j,k);
12     while(i<=mid&&j<=t){
13         if(a[i]<=a[j])
14             r[k]=a[i],k++,i++;
15         else{
16             r[k]=a[j],k++,j++;
17             ans+=mid-i+1;
18         }
19         //printf("r[k]=%d\n",r[k-1]);
20     }
21     while(i<=mid)
22         r[k]=a[i],k++,i++;
23     while(j<=t)
24         r[k]=a[j],k++,j++;
25     for(int i=s;i<=t;i++)
26         a[i]=r[i];//printf("a[i]=%d\n",a[i])
27 }
28 int main()
29 {
30     scanf("%d",&n);
31     for(int i=1;i<=n;i++)
32         scanf("%lld",&a[i]);
33     msort(1,n);
34     printf("%lld\n",ans);
35     return 0;
36 }
```

转换为二进制

```
1  #include <iostream>
2  #include <bitset>
3  using namespace std;
4
5  void print_bin_aux(int a) {    //辅助函数
6      if (a == 0) return ;
7      print_bin_aux(a / 2);
8      cout << (a % 2);
```

```

9   }
10
11  void print_bin(int a) {
12      if (a == 0)
13          cout << 0;
14      else
15          print_bin_aux(a);
16      cout << endl;
17  }                                     //转化为二进制函数;
18
19  class Solution {
20  public:
21      int rangeBitwiseAnd(int left, int right) {
22          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;
23
24          print_bin(left);
25          print_bin(0);
26          print_bin(0xff);
27          print_bin(0xaa);
28
29          return -1;
30      }
31  };
32
33  int main()
34  {
35      int left = 4, right = 6;
36      cout << Solution().rangeBitwiseAnd(left, right) << endl;
37      return 0;
38  }

```

二叉树问题

```

1   #include<cmath>
2   #include<cstdio>
3   #include<cstring>
4   #include<iostream>
5   #include<algorithm>
6   using namespace std;
7
8   int n;
9   int fa[101], root[101], son[101];
10
11  int depth[101], width[101];
12  int lca(int x, int y) {
13      if (x == y) {
14          return x;
15      }

```

```

16     else if(depth[x] == depth[y]){
17         return lca(fa[x],fa[y]);
18     }
19     else if(depth[x] < depth[y]){
20         return lca(x,fa[y]);
21     }
22     else{
23         return lca(fa[x],y);
24     }
25 }
26
27 int main(){
28     cin >> n;
29     depth[1] = 1;
30     for(int i = 1;i < n;i++){
31         cin >> root[i] >> son[i];
32         depth[son[i]] = depth[root[i]] + 1;
33         fa[son[i]] = root[i];
34     }
35     int x,y;
36     cin >> x >> y;
37     int max_depth = 1;
38     for(int i = 1;i <= n;i++){
39         max_depth = max(max_depth,depth[i]);
40         width[depth[i]]++;
41     }
42     cout << max_depth << endl;
43     int max_width = 1;
44     for(int i = 1;i <= n;i++){
45         max_width = max(max_width,width[i]);
46     }
47     cout << max_width << endl;
48     int k = lca(x,y);          //求 LCA 的结点序号
49     cout << (depth[x] - depth[k]) * 2 + depth[y] - depth[k];          //求
    距离
50     return 0;
51 }

```

algorithm常用的函数

```

1 // max、min和abs函数，下面是具体的代码：
2
3 #include<stdio.h>
4 #include<algorithm>
5 using namespace std;
6 int main(void){
7     int x = 1, y = -2;
8     printf("%d %d\n", max(x, y), min(x, y));
9     printf("%d %d\n", abs(x), abs(y));

```

```
10     return 0;
11 }
12 // swap函数，下面是具体的代码：
13
14 #include<stdio.h>
15 #include<algorithm>
16 using namespace std;
17 int main(void) {
18     int x = 1, y = -2;
19     swap(x, y);
20     printf("%d %d\n", x, y);
21     return 0;
22 }
23 // reverse函数
24 // 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：
25
26 #include<stdio.h>
27 #include<algorithm>
28 using namespace std;
29 int main(void) {
30     int arr[] = {1, 2, 3, 4, 5, 6, 7, 7};
31     reverse(arr, arr + 8);
32     for(int i = 0; i < 8; ++i){
33         printf("%d ", arr[i]);
34     }
35     return 0;
36 }
37
38 // next_permutation函数
39 // 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：
40
41
42 // next_permutation -> 升序
43 // prev_permutation -> 降序
44
45 #include<stdio.h>
46 #include<algorithm>
47 using namespace std;
48 int main(void) {
49     int arr[] = {1, 2, 3, 4};
50     //使用C++中的next_permutation函数生成数组的全排列
51     do{
52         printf("%d%d%d%d\n", arr[0], arr[1], arr[2], arr[3]);
53     }while(next_permutation(arr, arr + 4));
54     return 0;
55 }
56
```



```
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 }
```