ACM算法与程序设计
（二）
基础数据结构

杜育根

ygdu@sei.ecnu.edu.cn

基础数据结构

- 在这章里，你将会学习如何借助 C++/Java 标准库中的工具快速实现一些常用的数据结构和算法，如动态数组、集合、映射、栈、队列、优先队列等，并介绍一个在竞赛中很常见的数据结构——并查集的原理和用法。

动态数组

- 有些时候想开一个数组，但是却不知道应该开多大长度的数组合适，因为我们需要用到的数组可能会根据情况变动。这时候我们就需要用到动态数组。所谓动态数组，也就是不定长数组，数组的长度是可以根据我们的需要动态改变的。动态数组的实现也不难，但是在 C++ 和 Java 里面有已经写好的标准模板库（Standard Template Library），就是我们常说的 STL 库，实现了集合、映射表、栈、队列等数据结构和排序、查找等算法。我们可以很方便地调用标准库来减少我们的代码量。
- C++ 中动态数组写作vector，Java 中写作ArrayList，C 语言中没有标准库，因此非常建议 C/C++ 组的选手使用 C++ 进行比赛，即使你之前只用过 C 也没关系，因为 C 语言的代码几乎都可以被 C++ 编译器正确理解。用 C 语言同学尽管写的是 C 代码也尽量用 C++ 的方式编译。

引用库和方法

- C++ 中vector的实现在一个<vector>头文件中，在代码开头引入这个头文件，并在引入所有头文件之后加上一句using namespace std。构造一个vector的语句为：vector<T> vec。这样我们定义了一个名为vec的储存T类型数据的动态数组。其中T是我们数组要储存的数据类型，可以是int、float、double、或者其他自定义的数据类型等等。初始的时候vec是空的。
- 通过push_back()方法在数组最后面插入一个新的元素。通过size()方法获取vector的长度，通过[]操作直接访问vector中的元素，这一点和数组是一样的。修改vector中某个元素很简单，只需要用=给它赋值就好了，比如vec[1] = 3。C++ 调用clear()方法就可清空vector。
- C++ 中vector的clear()只是清空vector，并不会清空开的内存。用一种方法可以清空vector的内存。先定义一个空的vector x, 然后用需要清空的vector和x交换，因为x是局部变量，所以会被系统回收内存（注意 大括号一定不能去掉）。
- vector<int> v;
- { vector<int> x;
- v.swap(x); }

C++ vector应用

```
1 #include <vector>
2 #include <stdio.h>
3 using namespace std;
4 int main() {
5     vector<int> vec;  // []
6     vec.push_back(1); // [1]
7     vec.push_back(2); // [1, 2]
8     vec.push_back(3); // [1, 2, 3]
9     vec[1] = 3; // [1, 3, 3]
10     vec[2] = 2; // [1, 3, 2]
11     for (int i = 0; i < vec.size(); ++i) {
12         printf("%d\n", vec[i]);
13     }
14     return 0;
15 }
```

C++ vector方法总结

方法	功能
push_back	在末尾加入一个元素
pop_back	在末尾弹出一个元素
size	获取长度
clear	清空

Java ArrayList方法总结

Java `ArrayList`方法总结：

方法	功能
add	增加元素
get	获取元素
set	修改元素
size	获取长度
clear	清空

练习题：打印锯齿矩阵

- 锯齿矩阵是指各行包含的元素个数不相同的矩阵。现读入若干对整数（x,y），表示在第 x 行的末尾加上一个元素 y。输出最终的锯齿数组。初始时矩阵为空。
- 输入格式：第一行输入两个整数 n,m(1≤n,m≤10000)，其中 n表示锯齿数组的行数，m 表示插入的元素总数。接下来一共 m行，每行两个整数 x,y(1≤x≤n,0≤y≤10000)，表示在第x 行的末尾插入一个元素 y。
- 输出格式：一共输出 n 行，每行若干个用空格分隔的整数。如果某行没有任何元素，则输出一个空行。
- 样例输入
- 3 12
- 1 3
- 2 2
- 2 3
- 2 4
- 3 1
- 3 6
- 1 5
- 1 2

1 6
3 2
3 7
1 1
样例输出
3 5 2 6 1
2 3 4
1 6 2 7

练习题：堆积木

- 小明有 n 块积木，编号分别为 1 到 n。一开始，蒜头把第 i 块积木放在位置 i。小明进行 m 次操作，每次操作，蒜头把位置 b 上的积木整体移动到位置 a 上面。比如 1 位置的积木是 1，2 位置的积木是 2，那么把位置 2 的积木移动到位置 1 后，位置 1 上的积木从下到上依次为 1,2。
- 输入格式：第一行输入 2 个整数 n,m(1≤n≤10000,0≤m≤10000)。接下来 m 行，每行输入 2 个整数 a,b(1≤a,b≤n)，如果a，b 相等则本次不需要移动。
- 输出格式：输出 n 行，第 i 行输出位置 i 从下到上的积木编号，如果该行没有积木输出一行空行。
- 样例输入1
- 2 2
- 1 2
- 1 2
- 样例输出1
- 1 2

样例输入2

4 4
3 1
4 3
2 4
2 2

样例输出2

2 4 3 1

集合

- 集合是数学中的一个基本概念，通俗地理解，集合是由一些不重复的数据组成的。比如 {1,2,3} 就是一个有 1,2,3 三个元素的集合。C++ 和 Java 的标准库中的集合支持高效的插入、删除和查询操作，这 3 个操作的时间复杂度都是 O(lgn)，其中 n是当前集合中元素的个数。如果用数组，虽然插入的时间复杂度是 O(1)，但是删除和查询都是 O(n)，时间效率太低。
- 在 C++ 中我们常用的集合是set，在 Java 中常用集合是HashSet。

引用库

- C++ 中set的实现在一个<set>头文件中，在代码开头引入这个头文件，并且同样加上一句using namespace std。构造一个set的语句为：set<T> s。这样我们定义了一个名为s的、储存T类型数据的集合，其中T是集合要储存的数据类型。初始的时候s是空集合。通过iterator（迭代器）可以访问集合中的每个元素，迭代器就好比指向集合中的元素的指针。 C++ 中遍历set是从小到大进行的。调用clear()方法就可清空set。

C++ set集合例子insert、erase方法

用insert()方法向集合中插入一个新的元素。注意如果集合中已经存在了某个元素，再次插入不会产生任何效果，集合中是不会出现重复元素的。通过erase()方法删除集合中的一个元素，如果集合中不存在这个元素，不进行任何操作。

```
1  #include <set>
2  #include <string>
3  using namespace std;
4  int main() {
5      set<string> country;  // {}
6      country.insert("China"); // {"China"}
7      country.insert("America"); // {"China", "America"}
8      country.insert("France"); // {"China", "America", "France"}
9      country.erase("America"); // {"China", "France"}
10     country.erase("England"); // {"China", "France"}
11     return 0;
12 }
```

C++ set集合例子count方法

查找元素 C++ 中如果你想知道某个元素是否在集合中出现，你可以直接用count()方法。如果集合中存在我们要查找的元素，返回 1，否则会返回 0 。

```
#include <set>
#include <string>
#include <stdio.h>
using namespace std;
int main() {
    set<string> country;  // {}
    country.insert("China"); // {"China"}
    country.insert("America"); // {"China", "America"}
    country.insert("France"); // {"China", "America", "France"}
    if (country.count("China")) {
        printf("China belong to country");
    }
    return 0;
}
```

C++ set集合例子-遍历

通过iterator（迭代器）可以访问集合中的每个元素，迭代器就好比指向集合中的元素的指针。 C++ 中遍历set是从小到大进行的。

```cpp
#include <set>
#include <string>
#include <iostream>
using namespace std;
int main() {
    set<string> country;  // {}
    country.insert("China"); // {"China"}
    country.insert("America"); // {"China", "America"}
    country.insert("France"); // {"China", "America", "France"}
    for (set<string>::iterator it = country.begin(); it != country.end(); ++it) {
        cout << (*it) << endl;
    }
    return 0;
}
```

C++set方法总结

方法	功能
insert	插入一个元素
erase	删除一个元素
count	判断元素是否在`set`中
size	获取元素个数
clear	清空

Java HashSet方法总结

方法	功能
add	插入元素
remove	删除元素
contains	判断一个元素是否在`HashSet`中
size	获取长度
clear	清空

练习题：计算集合的并

- 给你两个集合，计算其并集，即{A}+{B}。注：{A}+{B}中不允许出现重复元素，但是{A}与{B}之间可能存在相同元素。
- 输入格式：输入数据分为三行，第一行有两个数字n,m(0<n,m≤10000)，分别表示集合A和集合B的元素个数。后两行分别表示集合A和集合B。每个元素为不超出 int 范围的整数，每个元素之间用一个空格隔开。
- 输出格式：输出一行数据，表示合并后的集合，要求从小到大输出，每个元素之间用一个空格隔开。
- 样例输入1　　样例输入2
- 1 2　　1 2
- 1　　1
- 2 3　　1 2
- 样例输出1　　样例输出2
- 1 2 3　　1 2

练习题：小明学英语

- 小明快要考托福了，这几天，小明每天早上都起来记英语单词。花椰妹时不时地来考一考小明：花椰妹会询问小明一个单词，如果小明背过这个单词，小明会告诉花椰妹这个单词的意思，不然小明会跟花椰妹说还没有背过。单词是由连续的大写或者小写字母组成。注意单词中字母大小写是等价的。比如You和you是一个单词。
- 输入格式：首先输入一个 n（1≤n≤100000）表示事件数。接下来 n 行，每行表示一个事件。每个事件输入为一个整数 d 和一个单词 word（单词长度不大于 20），用空格隔开。如果 d=0，表示小明记住了 word 这个单词，如果 d=1，表示这是一个 测试，测试小明是否认识单词 word（花椰妹永远不会告诉小明这个单词的意思）。事件的输入是按照时间先后顺序输入的。
- 输出格式：对于花椰妹的每次测试，如果小明认识这个单词，输出一行Yes, 否则输出一行No
- 样例输入

5	样例输出
0 we	No
0 are	Yes
1 family	
0 Family	
1 Family	

映射表

- 映射是指两个集合之间的元素的相互对应关系。通俗地说，就是一个元素对应另外一个元素。比如有一个姓名的集合 {"Tom"，"Jone"，"Mary"}，班级集合{1,2}。姓名与班级之间可以有如下的映射关系：classclass("Tom") =1，classclass("Jone") =2，classclass("Mary") =1　我们称其中的姓名集合为 关键字集合（key），班级集合为 值集合（value）。　在 C++ 中我们常用的映射是map，在 Java 中常用映射是HashMap。
- 引用库　在 C++ 中map的实现在一个<map>头文件中，在代码开头引入这个头文件，并且加上一句using namespace std。

```
1 #include <map>
2 using namespace std;
```

映射方法

- 在 C++ 中，我们构造一个map的语句为：map<T1, T2> m;。这样我们定义了一个名为m的从T1类型到T2类型的映射。初始的时候m是空映射。
- 通过insert()方法向集合中插入一个新的映射，参数是一个pair类型的结构。这里需要用到另外一个 STL 模板——元组（pair），pair<int, char>(1, 'a') 定义了一个整数 1 和字符a的pair。我们向映射中加入新映射对的时候就是通过加入pair来实现的。如果插入的 key 之前已经有了 value，不会用插入的新的 value 替代原来的 value，也就是此次插入是无效的。

访问映射

- 在 C++ 中访问映射和数组一样，直接用[]就能访问。比如dict[“Tom”]就可以获取“Tom”的班级了。而这里有一个比较神奇的地方，如果没有对“Tom”做过映射的话，此时你访问dict[“Tom”]，系统将会自动为“Tom”生成一个映射，其value为对应类型的默认值。并且我们可以之后再给映射赋予新的值，比如dict[“Tom”] = 3，这样为我们提供了另一种方便的插入手段。当然有些时候，我们不希望系统自动为我们生成映射，这时候我们需要检测“Tom”是否已经有映射了，如果已经有映射再继续访问。这时就需要用count()函数进行判断，判断某个关键字是否被映射过，你可以直接用count()方法。如果被映射过，返回 1，否则会返回0。

```cpp
#include <map>
#include <string>
#include <stdio.h>
using namespace std;
int main() {
    map<string, int> dict;  // {}
    dict["Tom"] = 1; // {"Tom"->1}
    dict["Jone"] = 2; // {"Tom"->1, "Jone"->2}
    dict["Mary"] = 1; // {"Tom"->1, "Jone"->2, "Mary"->1}
    if (dict.count("Mary")) {
        printf("Mary is in class %d\n", dict["Mary"]);
    } else {
        printf("Mary has no class");
    }
    return 0;
}
```

遍历映射

在 C++ 中，通过迭代器可以访问映射中的每对映射，每个迭代器的 first 值对应 key，second 值对应 value。调用 clear() 方法就可清空map。

```cpp
#include <map>
#include <string>
#include <iostream>
using namespace std;
int main() {
    map<string, int> dict;  // {}
    dict["Tom"] = 1; // {"Tom"->1}
    dict["Jone"] = 2; // {"Tom"->1, "Jone"->2}
    dict["Mary"] = 1; // {"Tom"->1, "Jone"->2, "Mary"->1}
    for (map<string, int>::iterator it = dict.begin(); it != dict.end(); ++it) {
        cout << it->first << " is in class " << it->second << endl;
    }
    return 0;
}
```

C++map常用方法总结

方法	功能
insert	插入一对映射
count	查找关键字
erase	删除关键字
size	获取映射对个数
clear	清空

练习题：小明面试

- 小明去面试的时候，曾经遇到这样一个面试题：
- 给定 n 个整数，求里面出现次数最多的数，如果有多个重复出现的数，求出值最大的一个。当时可算是给小明难住了。现在小明来考考你。
- 输入格式 ：第一行输入一个整数n(1≤n≤100000)，接下来一行输入n个int 范围内的整数。
- 输出格式 ：输出出现次数最多的数和出现的次数，中间用一个空格隔开，如果有多个重复出现的数，输出值最大的那个。
- 样例输入
- 10
- 9 10 27 4 9 10 3 1 2 6
- 样例输出
- 10 2

练习题：水果店

- 小明经营着一个不大的水果店。他认为生存之道就是经营最受顾客欢迎的水果。现在他想要一份水果销售情况的明细表，这样就可以很容易掌握所有水果的销售情况了。小明告诉你每一笔销售记录的水果名称，产地和销售的数量，请你帮他生成明细表。
- 输入格式 ： 第一行是一个整数 N(0 < N≤1000)，表示工有N 次成功的交易。其后有 N 行数据，每行表示一次交易，由水果名称(小写字母组成，长度不超过 100)，水果产地(小写字母组成，长度不超过 100)和交易的水果数目(正整数，不超过 1000)组成.
- 输出格式 ： 请你输出一份排版格式正确(请分析样本输出)的水果销售情况明细表。这份明细表包括所有水果的产地、名称和销售数目的信息。水果先按产地分类，产地按字母顺序排列；同一产地的水果按照名称排序，名称按字母顺序排序。
- 样例输入
- 5
- apple shandong 3
- pineapple guangdong 1
- sugarcane guangdong 1
  pineapple guangdong 3
  pineapple guangdong 1

样例输出

guangdong
|—-pineapple(5)
|—-sugarcane(1)
shandong
|—-apple(3)

栈（stack）

- 如果你和程序员接触的比较多，你可能会经常听到他们说一个词——“栈溢出（Stack Overflow）”。那么栈到底是什么？为什么会溢出这？下面我们就一一解答这些问题。
- 栈（stack），又名堆栈，是一种运算受限制的线性表类型的数据结构。其限制是只允许在栈的一端进行插入和删除运算。这一端被成为栈顶，相对地，把另一端称为栈底。 可以想象往子弹夹中装子弹的情形，正常情况下只能往子弹夹入口那段端入子弹，这一步就好比向栈中压入元素，称为push，射击的时候，弹夹会从顶端弹出子弹，这一步就好比从栈顶弹出元素，称为pop，可以发现，从栈顶弹出的子弹是最后一个压进去的子弹，这也是栈的一个重要性质，先进后出（FILO——first in last out）。另外，用一个top指针来标示当前栈顶的位置。

栈的实现（C语言手动方法）

- 关于栈的实现，一种方法是利用数组手动实现，需要固定缓存大小，也是就是数组大小。用stack表示储存栈的空间，top表示当栈顶的指针位置，方法push()压入一个数x到栈顶，方法pop()从栈顶弹出一个元素，方法topval()获取栈顶元素。

```
1 int stack[maxsize], top = 0;
2 void push(int x) {
3     stack[top++] = x;
4 }
5 void pop() {
6    --top;
7 }
8 int topval() {
9     return stack[top - 1];
10 }
11 int empty() {
12     return top > 0;
13 }
```

栈的实现和应用

在 C++ 和 Java 中有已经写好的栈对象，可以直接用。

```
1 #include <stack>
2 #include <iostream>
3 using namespace std;
4 stack<int> S;
5 int main() {
6     S.push(1);
7     S.push(10);
8     S.push(7);
9     while (!S.empty()) {
10        cout << S.top() << endl;
11        S.pop();
12    }
13    return 0;
14 }
```

上面代码是 C++ 里面的 stack 的用法。push，pop分别是压栈和出栈，top是栈顶元素，empty判断栈是否为空。

练习题：括号匹配

- 小明在纸上写了一个串，只包含‘（‘和‘）’。一个‘（‘能唯一匹配一个‘）’，但是一个匹配的‘（‘必须出现在‘）’之前。请判断小明写的字符串能否括号完全匹配，如果能，输出配对的括号的位置（匹配的括号不可以交叉，只能嵌套）。
- 输入格式
- 一行输入一个字符串只含有‘（‘和‘）’，输入的字符串长度不大于50000。
- 输出格式
- 如果输入括号不能匹配，输出一行”No”，否则输出一行”Yes”，接下里若干行每行输出 2 个整数，用空格隔开，表示所有匹配对的括号的位置（下标从 1 开始）。你可以按照任意顺序输出。
- 本题答案不唯一，符合要求的答案均正确
- 样例输入
- ()()
- 样例输出
- Yes
- 1 2
- 3 4

练习题：网页跳转

小明每天都在用一款浏览器软件。在这个浏览器中，一共三种操作：打开页面、回退和前进。它们的功能如下：

- 打开页面：在地址栏中输入网址，并跳转到网址对应的页面；
- 回退：返回到上一次访问的页面；
- 前进：返回到上次回退前的页面，如果上一次操作是打开页面，那么将无法前进。

现在，小明打开浏览器，进行了一系列操作，你需要输出他每次操作后所在页面的网址。

输入格式：第一行输入一个整数n(0<n≤100000)，表示小明的操作次数。接下来一共n行，每行首先输入一个字符串，如果是VISIT，后面接着输入一个不含有空格和换行的网址（网址长度小于100），表示小明在浏览器地址栏中输入的网址；如果是BACK，表示小明点击了回退按钮；如果是FORWARD，表示小明点击了前进按钮。

- 输出格式：对于每次操作，如果小明能操作成功，输出小明操作之后的网址，否则输出Ignore。假设小明输入的所有网址都是合法的。

样例输入
10
VISIT https://www.jisuanke.com/course/476
VISIT https://www.taobao.com/
BACK
BACK
FORWARD
FORWARD
BACK
VISIT https://www.jisuanke.com/course/429
FORWARD
BACK

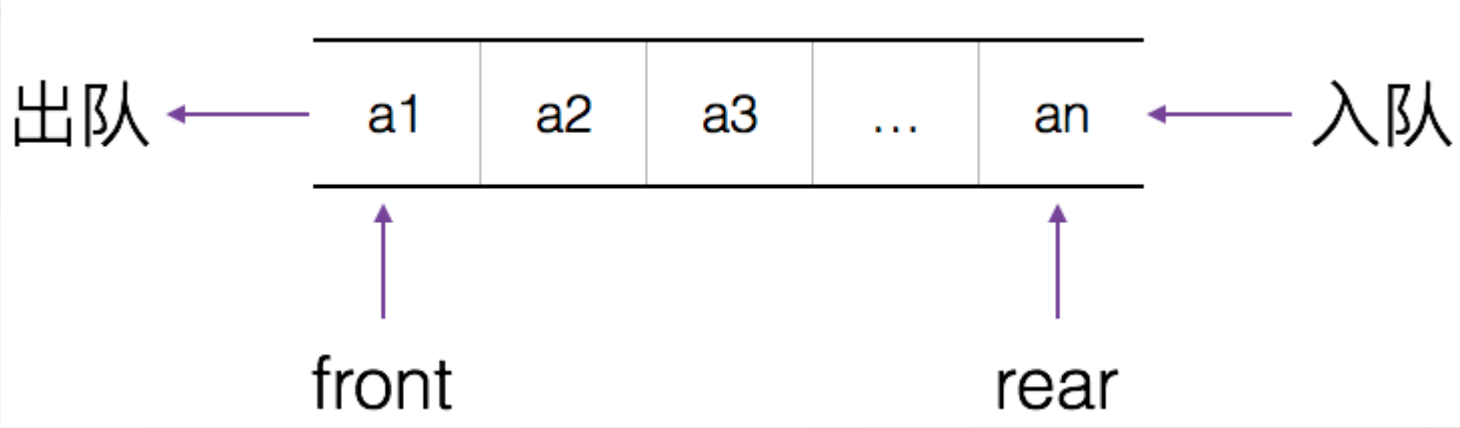
样例输出
https://www.jisuanke.com/course/476
https://www.taobao.com/
https://www.jisuanke.com/course/476
Ignore
https://www.taobao.com/
Ignore
https://www.jisuanke.com/course/476
https://www.jisuanke.com/course/429
Ignore
https://www.jisuanke.com/course/476

队列

- 队列（queue） 是一种线性的数据结构，和栈一样是一种运算受限制的线性表。其限制只允许从表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作。一般允许进行插入的一端我们称为队尾，允许删除的一端称为队首。队列的插入操作又叫入队，队列的删除操作又叫出队。 可以把队列想象成购物时排队结账的时候的队伍，先排队的人会先结账，后排队的人会后结账，并且不允许有插队的行为，只能在队伍的末尾进行排队。这就是队列的特点，具有先进先出（FIFO——First In First Out）的性质。

队列的结构与操作

- 队列的结构如下图所示：



- 队列的主要操作包括： 入队（push） 出队（pop） 判断队列是否为空（empty） 统计队列元素的个数（size） 访问队头元素（front） 访问队尾元素（back）

队列的实现与使用

- 在日常使用中，往往不需要手动实现队列，在 C++ 和 Java 中均有已经写好的队列，供我们使用。下面分别是使用 C++ 和 Java 队列的示例代码。

操作	C++	Java
入队	push	add
出队	pop	remove
访问队首元素	front	element
大小	size	size
是否为空	empty	isEmpty

```cpp
#include <queue>
#include <iostream>
using namespace std;
int main() {
    queue<int> q; // 声明一个装 int 类型数据的队列
    q.push(1); // 入队
    q.push(2);
    q.push(3);
    cout << q.size() << endl; // 输出队列元素个数
    while (!q.empty()) { // 判断队列是否为空
        cout << q.front() << endl; // 访问队首元素
        q.pop(); // 出队
    }
    return 0;
}
```

练习题：报数

- 有 n个小朋友做游戏，他们的编号分别是1,2,3...n。他们按照编号从小到大依次顺时针围成一个圆圈，从第一个小朋友开始从 1 报数，依次按照顺时针方向报数(加一)，报 m 的人会离开队伍，然后下一个小朋友会继续从 1开始报数，直到只剩一个小朋友为止。
- 输入格式：第一行输入俩个整数，n，m（1≤n,m≤1000）
- 输出格式：输出最后一个小朋友的编号，占一行。
- 样例输入
- 10 5
- 样例输出
- 3

练习题：敲7

- 有一种酒桌游戏叫做“敲7”，规则是从一个人开始，说出任意数字，其他人会顺序往后报，如果一个数字包含 7，或者是 7 的倍数，那么需要敲打杯子或盘子，不能说出。
- 现在 n 个人围坐在一个圆桌周围，他们编号从 1 到 n 顺时针排列。从某一人开始报出一个数字，其他人会按照顺时针方向顺序往后报(加一)，如果某个人的数字包含 7，或者是 7 的倍数，那么他将退出游戏，下一个人继续接着报，直到剩一个人为止。
- 输入格式：第一行输入三个整数，n，m，t。n 代表总人数，m 代表从第 m 个人开始报数，他报出的数字是 t。（1≤m≤n≤1000，1≤t≤100）接下来的 n 行，每一行输入一个字符串，代表这 n 个人的名字，字符串的长度不超过 20。
- 输出格式：输出剩下的那个人的名字，占一行。
- 样例输入
- 5 3 20
- donglali
- nanlali
- xilali
- beilali
- chuanpu
- 样例输出
- chuanpu

优先队列priorit_queue

- 前面已经接触了队列这个数据结构。利用队列先进先出的性质，可以解决很多实际问题，但对于一些特殊的情况，队列是无法解决的。 例如在医院里，重症急诊患者肯定不能像普通患者那样依次排队就诊。这时候，我们就需要一种新的数据结构——优先队列，先访问优先级高的元素（例如下图中的重症急诊患者）。

优先队列的操作

- 在队列中，元素从队尾进入，从队首删除。相比队列，优先队列里的元素增加了优先级的属性，优先级高的元素先被删除。
- STL 中的优先队列 - priorit_queue，包含在头文件”queue”中，可以使用具有默认优先级的已有数据结构；也可以再定义优先队列的时候传入自定义的优先级比较对象；或者使用自定义对象(数据结构)，但是必须重载好< 操作符。
- 优先队列的使用和队列基本上没有区别，右面给出 C++ 实例代码。

```cpp
#include <queue>
#include <iostream>
using namespace std;
int main() {
    priority_queue<int> q; // 声明一个装 int 类型数据的优先队列
    q.push(1); // 入队
    q.push(2);
    q.push(3);
    while (!q.empty()) { // 判断队列是否为空
        cout << q.top() << endl; // 访问队列首元素
        q.pop(); // 出队
    }
    return 0;
}
/*
输出为
3
2
1
*/
```

优先队列常用定义和重载运算符方法①默认优先级

- #include <queue>
- using namespace std;

priority_queue<int> **q;**

- 通过<操作符可知在整数中元素大的优先级高。

②传入一个比较函数，使用functional.h函数对象作为比较函数。

- #include <queue>
- #include <functional>
- using namespace std;

- **priority_queue**<int, vector<int>, **greater**<int> > **q;**
- 此时整数中元素小的优先级高。

③传入比较结构体，自定义优先级。

```
#include <queue>
using namespace std;

struct cmp1
    { bool operator ()(int a,int b)
        { //通过传入不同类型来定义不同类型优先级
        return a>b; //最小值优先
        }
    };
struct cmp2
    { bool operator ()(int a,int b)
        { return a<b; //最大值优先 }
    };
priority_queue<int, vector<int>, cmp1> q1;
priority_queue<int, vector<int>, cmp2> q2;
```

④自定义数据结构，自定义优先级。

```
#include <queue>
using namespace std;
struct node
{    int priority; int value;
     friend bool operator < (const node &a, const node &b)
          { return a.priority < b.priority; }
     /* 这样写也可以
     bool operator < (const node &a) const
          { return priority < a.priority; }
     */
};
priority_queue<node> q;
```

练习题：任务系统

- 小明设计了一个任务系统。这个系统是为了定时提醒小明去完成一些事情。系统大致如下，初始的时候，小明可能会注册很多任务，每一个任务的注册如下:
- Register Q_num Period
- 表示从系统启动开始，每过 Period 秒提醒小明完成编号为 Q_num 的任务。你能计算出小明最先被提醒的 k 个任务吗?
- 输入格式：第一行输入 n(0<n≤3000)，k(0<k≤10000)，其中 n 表示小明注册的任务数量。
- 接下来 n 行，每行输入一条注册命令，其中 0<q_num≤3000，0≤Period≤3000。
- 输出格式：顺序输出 k 行，表示依次提醒的任务的编号。如果同一时间有多个任务，最先提醒编号小的任务。
- 样例输入
- 2 5
- Register 2004 200
- Register 2005 300
- 样例输出
- 2004
- 2005
- 2004
- 2004
- 2005

练习题：n个最小和

- 给出两个包含n 个整数的数组 A，B。分别在 A, B 中任意出一个数并且相加，可以得到 n^2个和。求这些和中最小的 n 个。
- 输入格式:输入第一行一个整数 n(1≤n≤50000)
  接下来一行输入数组 A，用空格隔开。
  接下来一行输入数组 B，用空格隔开。
  1≤Ai,Bi≤10^9
- 输出格式:从小到大输出最小的 n 个和，用空格隔开。
- 样例输入
- 4
  1 3 5 7
  2 4 6 8
- 样例输出
- 3 5 5 7

并查集

- 并查集，顾名思义，合并 查找 集合
- 这是一种树型的数据结构，用于处理一些不相交集合（Disjoint Sets）的合并和查询问题。在使用中常常以森林来表示。
- 并查集也是用来维护集合的，和前面学习的set不同之处在于，并查集能很方便地同时维护很多集合。如果用set来维护会非常的麻烦。并查集的核心思想是记录每个结点的父亲结点是哪个结点。
- 1) 初始化：初始的时候每个结点各自为一个集合，father[i] 表示结点 i的父亲结点，如果 father[i]=i，我们认为这个结点是当前集合根结点。

```
void init() {
   for (int i = 1; i <= n; ++i) {
         father[i] = i;
     }
 }
```

2) 查找：查找结点所在集合的根结点，结点 x的根结点必然也是其父亲结点的根结点。

```
int get(int x) {
    if (father[x] == x) { // x 结点就是根结点
        return x;
    }
    return get(father[x]); // 返回父结点的根结点
}
```

3) 合并：将两个元素所在的集合合并在一起，通常来说，合并之前先判断两个元素是否属于同一集合。

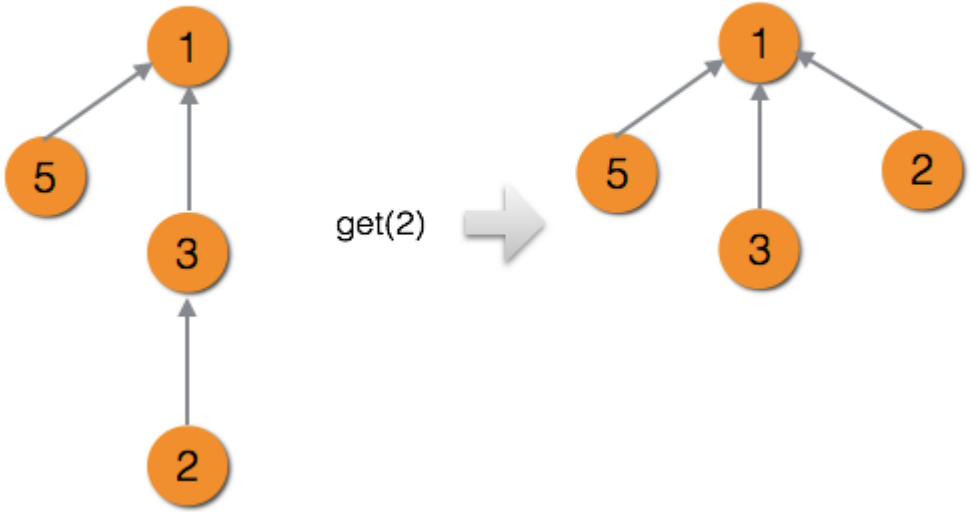
```
void merge(int x, int y) {
    x = get(x);
    y = get(y);
    if (x != y) { // 不在同一个集合
        father[y] = x;
    }
}
```

路径压缩

- 前面的并查集的复杂度实际上在有些极端情况会很慢。比如树的结构正好是一条链，那么最坏情况下，每次查询的复杂度达到了O(n)。这并不是我们期望的结果。
- 路径压缩 的思想是，我们只关心每个结点的父结点，而并不太关心树的真正的结构。这样我们在一次查询的时候，可以把查询路径上的所有结点的father[i] 都赋值成为根结点。只需要在我们之前的查询函数上面进行很小的改动。

```
int get(int x) {
    if (father[x] == x) { // x 结点就是根结点
        return x;
    }
    return father[x] = get(father[x]);
// 返回父结点的根结点，并令当前结点父结点
//直接为根结点
 }
```

下面图片是路径压缩前后的对比。



- 路径压缩在实际应用中效率很高，其一次查询复杂度平摊下来可以认为是一个常数。并且在实际应用中，我们基本都用带路径压缩的并查集。

带权并查集

- 所谓带权并查集，是指结点存有权值信息的并查集。并查集以森林的形式存在，而结点的权值，大多是记录该结点与祖先关系的信息。比如权值可以记录该结点到根结点的距离。 例题 在排队过程中，初始时，一人一列。一共有如下两种操作： 合并：令其中的两个队列 A,BA,B 合并，也就是将队列 AA 排在队列 BB 的后面 查询：询问某个人在其所在队列中排在第几位 例题解析 我们不妨设 size[]size[] 为集合中的元素个数，dist[]dist[] 为元素到队首的距离，合并时，dist[A.root]dist[A.root] 需要加上 size[B.root]]size[B.root]]（每个元素到队首的距离应该是到根路径上所有点的 dist[]dist[] 求和），size[B.root]size[B.root] 需要加上 size[A.root]size[A.root]（每个元素所在集合的元素个数只需查询该集合中根的 size[x.root]size[x.root]）。

1) 初始化：

```
void init() {
  for(int i = 1; i <= n; i++) {
      father[i] = i, dist[i] = 0, size[i] = 1;
    }
}
```

2) 查找

查找元素所在的集合，即根结点。

```
1 int get(int x) {
2     if(father[x] == x) {
3         return x;
4     }
5     int y = father[x];
6     father[x] = get(y);
7     dist[x] += dist[y];
// x 到根结点的距离等于 x 到之前父亲结点距离加上之前父亲结点到根结点的距离
8     return father[x];
9 }
```

路径压缩的时候，不需考虑 size[]，但 dist[] 需要更新成到整个集合根的距离。

3) 合并

将两个元素所在的集合合并为一个集合。 通常来说，合并之前，应先判断两个元素是否属于同一集合，这可用上面的“查找”操作实现。

```
1 void merge(int a, int b) {
2     a = get(a);
3     b = get(b);
4     if(a != b) {  // 判断两个元素是否属于同一集合
5         father[a] = b;
6         dist[a] = size[b];
7         size[b] += size[a];
8     }
9 }
```

通过小小的改动，我们就可以查询并查集这一森林中，每个元素到祖先的相关信息。

练习题：朋友

- 在社交的过程中，通过朋友，也能认识新的朋友。在某个朋友关系图中，假定 A 和 B 是朋友，B 和 C 是朋友，那么 A 和 C 也会成为朋友。即，我们规定朋友的朋友也是朋友。现在，已知若干对朋友关系，询问某两个人是不是朋友。请编写一个程序来解决这个问题吧。
- 输入格式：第一行：三个整数 n,m,p(n≤5000,m≤5000,p≤5000)分别表示有n 个人，m 个朋友关系，询问p 对朋友关系。接下来 m 行：每行两个数 Ai,Bi1≤Ai,Bi≤N，表示Ai 和 Bi具有朋友关系。接下来 p 行：每行两个数，询问两人是否为朋友。
- 输出格式：输出共 p 行，每行一个Yes或No。表示第 i个询问的答案为是否朋友。
- 样例输入　　样例输出
- 6 5 3　　Yes
- 1 2　　Yes
- 1 5　　No
- 3 4
- 5 2
- 1 3
- 1 4
- 2 3
- 5 6

练习题：网络交友

- 在网络社交的过程中，通过朋友，也能认识新的朋友。在某个朋友关系图中，假定 A 和 B 是朋友，B 和 C 是朋友，那么 A 和 C 也会成为朋友。即，我们规定朋友的朋友也是朋友。现在要求你每当有一对新的朋友认识的时候，你需要计算两人的朋友圈合并以后的大小。
- 输入格式：第一行：一个整数 n(n≤5000)，表示有 n 对朋友认识。
- 接下来 n 行：每行输入两个名字。表示新认识的两人的名字，用空格隔开。（名字是一个首字母大写后面全是小写字母且长度不超过 20 的串）。
- 输出格式：对于每一对新认识的朋友，输出合并以后的朋友圈的大小。
- 样例输入
- 3
- Fred Barney
- Barney Betty
- Betty Wilma
- 样例输出
- 2
- 3
- 4

练习题：找出所有谎言

小明有很多卡片，每张卡片正面上印着“剪刀”，“石头”或者“布”三种图案中的一种，反面则印着卡片的序号。“剪刀”，“石头”和“布”三种构成了一个有趣的环形，“剪刀”可以战胜“布”，“布”可以战胜“石头”，“石头”可以战胜“剪刀”。

现有 N 张卡片，以 1-N 编号。每张卡片印着“剪刀”，“石头”，“布”中的一种，但是我们并不知道它到底是哪一种。

有人用两种说法对这 N 张卡片所构成的关系进行描述： 第一种说法是“1 X Y”，表示 X 号卡片和 Y 号卡片是同一种卡片。第二种说法是“2 X Y”，表示 X 号卡片可以战胜 Y 号卡片。

小明对 N张卡片，用上述两种说法，一句接一句地说出 K 句话，这K 句话有的是真的，有的是假的。当一句话满足下列三条之一时，这句话就是假话，否则就是真话。 1) 当前的话与前面的某些真的话冲突，就是假话； 2) 当前的话中 X 或 Y 的值比 N 大，就是假话； 3) 当前的话表示 X 能战胜 X，就是假话。

你的任务是根据给定的 N 和 K 句话，计算假话的总数。

输入格式：第一行是两个整数N(1≤N≤50,000) 和K(0≤K≤100,000)，以一个空格分隔。以下 K 行每行是三个正整数 D，X，Y，两数之间用一个空格隔开，其中 D 表示说法的种类。 若 D=1，则表示 X 和 Y 是同一种卡片。 若 D=2，则表示 X 能战胜 Y。

输出格式： 只有一个整数，表示假话的数目。

样例输入

```
100 7
1 101 1
2 1 2
2 2 3
2 3 3
1 1 3
2 3 1
1 5 5
```

样例输出

```
2
3
4
```

练习题：接龙

- 小明在玩一种接龙的游戏，小明有30000张卡片分别放在30000列，每列依次编号为 1,2,...,30000。同时，小明也把每张卡片依次编号为1,2,...,30000。游戏开始，小明让让第 i 张卡片处于第 i(i=1,2,...,30000) 列。然后小明会发出多次指令，每次调动指令 Mij会将第 i张卡片所在的队列的所有卡片，作为一个整体（头在前尾在后）接至第 j张卡片所在的队列的尾部。小明还会查看当前的情况，发出 Cij 的指令，即询问电脑，第 i张卡片与第 j张卡片当前是否在同一个队列中，如果在同一列中，那么它们之间一共有多少张卡片。
- 聪明的你能不能编写程序处理小明的指令，以及回答小明的询问呢?
- 输入格式：第一行有一个整数 T（1≤T≤500000），表示总共有 T 条指令。以下有 T 行，每行有一条指令。指令有两种格式：Mij：i和 j是两个整数（1≤i,j≤30000），表示指令涉及的卡片编号。你需要让第 i张卡片所在的队列的所有卡片，作为一个整体（头在前尾在后）接至第 j张卡片所在的队列的尾部，输入保证第 i号卡片与第 j号卡片不在同一列。Cij：i 和j是两个整数（1≤i,j≤30000），表示指令涉及的卡片编号。该指令是小明的询问指令。
- 输出格式：如果是小明调动指令，则表示卡片排列发生了变化，你的程序要注意到这一点，但是不要输出任何信息；如果是小明的询问指令，你的程序要输出一行，仅包含一个整数，表示在同一列上，第 i号卡片与第 j号卡片之间的卡片数目（不包括第 i张卡片和第 j张卡片）。如果第 i号卡片与第 j号卡片当前不在同一个队列种中，则输出 -1。

样例输入	样例输出
4 M 2 3 C 1 2 M 2 4 C 4 2	-1 1