零.参考

一、不成章节的几点注意

二.头文件

1.#include<bits/stdc++.h>

2.#include<algorithm>

3.#include<limits>

三.STL之容器篇

1.vector —— 动态数组

2.string —— 字符串

3.list

4.set —— 集合

5.pair —— 对

6.map —— 哈希

7.priority_queue 优先队列

8.deque

四.STL之封装的方法

1.memset —— 数组初始化

2.sizeof —— 数组大小

3.sort —— 排序

4.reverse —— 倒序

5.to_string() —— int转string

6.stoi()和atoi() —— string转int

零.参考

1. 《STL教程：C++ STL快速入门（非常详细）》http://c.biancheng.net/stl/

2. 《C++中STL用法超详细总结》

https://blog.csdn.net/u010183728/article/details/81913729

一、不成章节的几点注意

1. 涉及求和的时候请注意是否需要使用 long long 型

2. 很长的一个计算式中可能会化为int型，所以要注意是否要使用强制转换

3. 在算法竞赛中，我们常常需要用到设置一个常量用来代表“无穷大”

   int型可以用---- 0x3f、0x3fffff等

long long型可以用 —— 4e18

二.头文件

1.#include<bits/stdc++.h>

<bits/stdc++.h>包含了全部的C++头文件。这样做题时直接敲上一句#include <bits/stdc++.h>而不是很多个#include。

但是 using namespace std; 这句还要自己敲

2.#include<algorithm>

3.#include<limits>

在算法竞赛中，我们常常需要用到设置一个常量用来代表“无穷大” ---- 0x3f3f3f3f

比如对于int类型的数，有的人会采用INT_MAX，即0x7fffffff作为无穷大。但是以INT_MAX为无穷大常常面临一个问题，即加一个其他的数会溢出。

所以在算法竞赛中，我们常采用0x3f3f3f3f来作为无穷大。0x3f3f3f3f的十进制为1061109567,相加依然不会溢出。

4.

三.STL之容器篇

1.vector —— 动态数组

（1）向量（Vector）是一个封装了动态大小数组的顺序容器。

（2）vector构造和基本的成员函数

```
// vector变量的定义和初始化
vector<int> v1(); //创建一个空vector
vector<int> v = {1,2,5}; // 初始化赋值
vector<int> v2(5); //创建一个大小为5的vector
vector<int> v3(5, 1); //创建一个大小为5初值为1
vector<int> v4(v3); //复制构造函数
vector<int> v5(v3.begin(), v3.end()); //复制[begin,end)区间

// 增加函数
 void push_back(const T& x):向量尾部增加一个元素X
 iterator insert(iterator it,const T& x):向量中迭代器指向元素前增加一个元素x
 iterator insert(iterator it,int n,const T& x):向量中迭代器指向元素前增加n个相同的元素x
 iterator insert(iterator it,const_iterator first,const_iterator last):向量中迭代器指向元素前插入另一个相同类型向量的[first,last)间的数据

// 删除函数
 iterator erase(iterator it):删除向量中迭代器指向元素
 iterator erase(iterator first,iterator last):删除向量中[first,last)中元素
 void pop_back():删除向量中最后一个元素
 void clear():清空向量中所有元素

// 遍历函数
 reference at(int pos):返回pos位置元素的引用
 reference front():返回首元素的引用
 reference back():返回尾元素的引用
 iterator begin():返回向量头指针，指向第一个元素
 iterator end():返回向量尾指针，指向向量最后一个元素的下一个位置
 reverse_iterator rbegin():反向迭代器，指向最后一个元素
 reverse_iterator rend():反向迭代器，指向第一个元素之前的位置

//查找函数
 vector<int>::iterator result = find(v.begin(), v.end(), a);
 if (result == v.end()) cout << "没有找到a";


// 其他
 bool empty() const:判断向量是否为空，若为空，则向量中无元素
 int size() const:返回向量中元素的个数
 int capacity() const:返回当前向量张红所能容纳的最大元素值
```

```
int max_size() const:返回最大可允许的vector元素数量值
void swap(vector&):交换两个同类型向量的数据
void assign(int n,const T& x):设置向量中第n个元素的值为x
void assign(const_iterator first,const_iterator last):向量中[first,last)中元素设置成当前向量元素
```

2.string —— 字符串

（1）经常使用string其实也是一个STL容器

（2）string和char*、char[]的转换

```
//char*转string：直接转换
char *p = "hello";//直接赋值
string s = p;

//string转char*
string s = "hello";
char *p = str.c_str(); //用c_str()方法

char p[50];//用单个char赋值法
 s.copy(p, 5, 0);
 *(p+5)='\0';//要手动加上终止符
```

（3）string和int互相转换

```
//string转int
string s="100";
int v=stoi(s);
```

（4）构造和基本成员函数

```
//string的构造
string s1 = "hello";
string* s2 = new string("hello");


//增加函数
s = "insertString:  " + s +" world"; //使用重构的+号，但是只能在开头和末尾
s.insert(s.length(), "!"); //使用insert函数，可以在任何位置
s.insert(pos, c); 在pos位置前插入字符c
 s.insert(pos, n , c); 在pos位置前插入n个字符c
```

```
s.insert(pos, str); 在pos位置前插入字符串str
s.push_back(c); 只能插入字符 等价与 s.insert(s.size(),c) 等价于 s += c
s.append(str); 等价与 s.insert(s.size(),str) 等价于 s += str


//删除函数
//（1）erase(pos,n); 删除从pos开始的n个字符，比如erase(0,1)就是删除第一个字符
//（2）erase(position);删除position处的一个字符(position是个string类型的迭代器)
//（3）erase(first,last);删除从first到last之间的字符（first和last都是迭代器）
s.erase(2,2); //删除从位置2开始的2个字符
s.clear(); 清除所有内容 等价于 s.erase(s.begin(), s.end());
s.pop_back(); 移除末尾字符 等价于 s.erase(s.end()-1);


//访问函数
string::iterator biter = s.begin();//等于s[0]
string::iterator eiter = s.end(); /指向最后一个字符后面的元素,不能输出哦,等于s[s.length()]
char c = s[5]; //按照位置访问
char c = s.at(5); //按照位置访问
char c = s.front();
char c = s.back();


//遍历
for(char c:s) cout<<c;


//查找函数
int index = s.find('a');
int index = s.find('a', int pos); //从指定位置pos起向后查找，直到串尾
int index = s.rfind('a', int pos); //从指定位置pos起向前查找，直到串首

//剪切函数
//（1）string ss = s.substr(pos, n); 返回从pos开始的n个字符
//（2）string ss = s.substr(pos); 返回从pos开始到结束的字符串，n的默认值是s.size() - pos
//（3）string ss = s.substr(); 返回整个字符串，pos的默认值是0
s.substr(pos, n);

```

```
//拷贝函数
//s.copy(p, n, pos=0):  从string类型对象中至多复制n个字符到字符指针p指向的空间中。
//默认从首字符开始，但是也可以指定，开始的位置（记住从0开始）。
//用户要确保p指向的空间足够保存n个字符。
char *p = new char[s.length()+1];
s.copy(p, s.length());
p[s.length()]='\0';//手动加上停止符

//其他
int n = s.length(); //获取字符串长度
int n = s.size(); //获取字符串长度
s.swap(s2); //字符串s和s2交换内容
s.empty(); 检查字符串是否为空
```

3.list

（1）List是stl实现的双向链表，与向量(vectors)相比，它允许快速的插入和删除，但是随机访问却比较慢

（2）构造和基本的成员函数

```
//list变量的定义和初始化
list<int>lst1;  //创建空list
list<int>lst2(5);   //创建含有5个元素的list
list<int>lst3(3,2);  //创建含有3个元素且初始化为2的list
list<int>lst4(lst2);    //使用lst2初始化lst4
list<int>lst5(lst2.begin(),lst2.end());  //同lst4

//遍历函数
list.front() 返回第一个元素
list.back() 返回最后一个元素
list.begin() 返回指向第一个元素的迭代器
list.end() 返回末尾的迭代器

//删除函数
list.remove() 从list删除元素
list.remove_if() 按指定条件删除元素
list.clear() 删除所有元素
list.erase() 删除一个元素
list.pop_front() 删除第一个元素
```

```
list.pop_back() 删除最后一个元素

//增加函数
list.push_front() 在list的头部添加一个元素
list.push_back() 在list的末尾添加一个元素
list.insert() 插入一个元素到list中

//其他
list.empty() 如果list是空的则返回true
list.reverse() 把list的元素倒转
list.size() 返回list中的元素个数
list.sort() 给list排序
list.swap() 交换两个list
```

4.set —— 集合

（1）set是一个关联容器，关联容器中的元素是按关键字来保存和访问的，set是一个只保存关键字的容器，set具备的两个特点：

- set中的元素都是排序好的
- set中的元素都是唯一的，没有重复的

（2）其他的set容器

- set(关键字即值，即只保存关键字的容器)；
- multiset(关键字可重复出现的set)；
- unordered_set(用哈希函数组织的set)；
- unordered_multiset(哈希组织的set，关键字可以重复出现)。

(3) 构造和基本的成员函数

```
// 初始化
set<int> s1;
set<string> s2 = {"one", "two", "three"};

// 插入函数
// set::insert() 返回一个二元组（Pair)：pair::first 为指向新插入元素或已经存在的元素的迭代器、pair::second 是一个 bool 值，新的元素被插入返回 true
s2.insert("four"); //插入单个元素
```

```
s2.insert({ "five","six"}); //数组形式插入多个元素
s2.insert("one").second ? printf("新元素成功插入! ") : printf("元素已经存在! ");



//删除函数
s2.erase("four"); //按键值删除

//访问和查找
if(s2.find("four") != s2.end()) cout<<"找到了";


//遍历
for(set<string>::iterator iter = s2.begin(); iter!=s2.end(); iter++) {
 cout<<*iter<<endl;
}
```

5.pair —— 对

（1）pair可以构造一个对（两个任意类型的值）

（2）pair的初始化和常用方法

```
#include<iostream> //pair包含在iostream中
using namespace std;

//初始化
pair<int, string> p;
pair<int, string> p(1, "one");

// 赋值
p = make_pair(1, "one");

// 取值
p.first
p.second
```

6.map —— 哈希

（1）自动建立Key－value的对应。key 和 value可以是任意你需要的类型，可以实现：

- 快速插入Key -Value 记录。
- 快速删除记录
- 根据Key 修改value记录。注意可以修改实值，而不能修改key。
- 遍历所有记录

（2）map内部数据是有序的。map内部数据的组织，map内部自建一颗红黑树（一种非严格意义上的平衡二叉树），这颗树具有对数据自动排序的功能，所以在map内部所有的数据都是有序的。

（3）map数据类型的初始化和常用成员函数

```
#include <map>

// map变量的定义和初始化
map<int,string> m; //定义一个空的map
map<int,string> m = {{1,"one"}, {2,"two"}}; //赋初值

// 增加函数
m[1] = "one"; //数组形式插入
m.insert(make_pair(1, "one")); //使用成员函数插入

// 访问函数
cout<<m[1]; //根据键访问值
map<int, string>::iterator iter = m.begin(); cout<<iter->first<<iter->second; //迭代器访问键和值

// 遍历函数
template <typename K, typename V>
void outputMap(map<K,V> m){
 for(auto iter = m.begin(); iter!=m.end(); iter++) {
 cout << iter->first << " : " << iter->second << endl;
 }
}

// 删除函数
int n = mapStudent.erase(1); //键值法删除，如果删除了会返回1，否则返回0
map<int, string>::iterator iter = mapStudent.find(1); m.erase(iter); //迭代器法删除
m.erase(m.begin(), m.end()); //范围删除
```

```
27
28 // 查找函数
29 map<int, string>::iterator iter = m.find(1); if(iter!= m.end()) cout<<"找
打了，其值："<<iter->second; //找到了
30
31
32  // 其他函数
33  int n = m.size();
```

7.priority_queue 优先队列

（1）优先队列和普通队列不同的地方是优先队列里面是由顺序的，按照元素优先级排序

在优先队列中，元素被赋予优先级。当访问元素时，具有最高优先级的元素最先删除。优先队列具有最高级先出 （first in, largest out）的行为特征。

优先级可以由我们自己定义，可以是数越大的优先级越高，也可以是数越小的优先级越高

（2）构造和基本的成员函数

```
1 #include<queue>
2
3 // 构造
4 priority_queue <int,vector<int>,greater<int> > q;//构造升序队列
5 priority_queue <int,vector<int>,less<int> >q;//降序构造队列
6
7 // 成员函数
8 top 访问队头元素
9 empty 队列是否为空
10  size 返回队列内元素个数
11  push 插入元素到队尾 (并排序)
12  emplace 原地构造一个元素并插入队列
13  pop 弹出队头元素
14  swap 交换内容
```

8.deque

（1）双端队列是一种随机访问的数据类型，提供了在序列两端快速插入和删除操作的功能

（2）构造和基本的成员函数

```
声明deque容器
#include<deque> // 头文件
deque<type> deq; // 声明一个元素类型为type的双端队列que
deque<type> deq(size); // 声明一个类型为type、含有size个默认值初始化元素的的双端队列que
deque<type> deq(size, value); // 声明一个元素类型为type、含有size个value元素的双端队列que
deque<type> deq(mydeque); // deq是mydeque的一个副本
deque<type> deq(first, last); // 使用迭代器first、last范围内的元素初始化deq

常用成员函数
 deque[i]：用来访问双向队列中单个的元素。
 deque.front()：返回第一个元素的引用。
 deque.back()：返回最后一个元素的引用。
 deque.push_front(x)：把元素x插入到双向队列的头部。
 deque.pop_front()：弹出双向队列的第一个元素。
 deque.push_back(x)：把元素x插入到双向队列的尾部。
 deque.pop_back()：弹出双向队列的最后一个元素。
```

四.STL之封装的方法

1.memset —— 数组初始化

memset是按字节赋值的，取变量a的后8位二进制进行赋值。

所以使用memset()初始化数组时候，只能将值初始化为-1,0，0x3f等是可以的

但是要吧数组初始化为一个指定的其他值memset无法做到，只能用for循环方法

```
int a[5];
memset(&a, 0, sizeof(a)); //第一种方法
memset(a, 0, sizeof(a)); //第二种方法

int a2[5][5];
memset(&a2, 0, sizeof(a2)); //第一种方法
memset(a2, 0, sizeof(a2)); //第二种方法
```

2.sizeof —— 数组大小

```
int a[4][3];
cout<<sizeof(a);
cout<<sizeof(a[0]);
```

3.sort —— 排序

```
#include<algorithm>
using namespace std;
sort(begin, end, less<type>()) //升序
sort(begin, end, greater<type>()) //降序
sort(begin, end, cmp) //自定义
```

4.reverse —— 倒序

```
#include<algorithm>
using namespace std;
reverse(beg,end)
```

5.to_string() —— int转string

使用 to_string() 函数，但是这个要求编辑器能使用c++11特性

```
int n = 12345;
string s = to_string(n);
```

使用流实现转换，这个不要求c++11特性，应用更广

```
int n = 1345;
stringstream ss;
ss << n;
string s;
ss >> s;
```

6.stoi()和atoi() —— string转int

使用 stoi() 函数，但是这个同样要求编辑器能使用c++11特性

```
string s="100";
int n = stoi(s);
```

使用 atoi() 函数，这个就不需要c++11特性，但是要用 c_str() 转为char型才能使用哦

```
string s="100";
int n = atoi(s.c_str());
```

使用流实现转换，这个不要求c++11特性，应用更广

```
1 istringstream is("12"); //构造输入字符串流，流的内容初始化为“12”的字符串
2 int i;
3 is >> i; //从is流中读入一个int整数存入i中
```

```
string s="100";
stringstream ss;
ss << s;
int n;
ss >> n;
```

7.min_element \ max_element 数组中的最值

```
bool cmp(int i, int j) { return i<j; }
int min_val = *min_element(a, a+n, cmp); //注意返回的是指针，要取值要加星号
int min_val = *max_element(a, a+n, cmp); //注意返回的是指针，要取值要加星号
```

8.nth_element 数组中第k小的数

nth_element() 函数无返回值，只是将第 k 大的元素排好了位置，若在其后的程序中想要使用第 k 大的元素，直接调用 a[k-1] 即可

```
bool cmp(int i, int j) { return i<j; }
nth_element(a, a+k, a+n, cmp);
cout<<a[k-1]<<endl;
```

9. lower_bound( ) \ upper_bound( ) 二分法在有序数组中找到恰当位置

```
在从小到大的排序数组中，
```

```
2
3 lower_bound( begin,end,num)：从数组的begin位置到end-1位置二分查找第一个大于或等于num的数字，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。
4
5 upper_bound( begin,end,num)：从数组的begin位置到end-1位置二分查找第一个大于num的数字，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。
6
7 在从大到小的排序数组中，重载lower_bound()和upper_bound()
8
9 lower_bound( begin,end,num,greater<type>() ):从数组的begin位置到end-1位置二分查找第一个小于或等于num的数字，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。
10
11 upper_bound( begin,end,num,greater<type>() ):从数组的begin位置到end-1位置二分查找第一个小于num的数字，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。
12
```

10.next_permutation 生成1-N的全排列，且按字典序输出

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

int N;
int a[100];

int main(){
 cin>>N;
 for(int i=1; i<=N; ++i) a[i]=i;
  do {
  for(int i=1; i<=N; ++i) cout<<a[i]<<" ";
  cout<<endl;
  } while(next_permutation(a+1,a+N+1));
  return 0;
}
```

五.指针篇

1.开辟空间

在C中使用malloc

```
struct TreeNode *newnode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
```

在C++中使用new

```
struct TreeNode *newnode = new struct TreeNode();
```

请遵守使用指针的准则：使用指针需要指明指针指向地址

```
int *p = 1; //写法错误，没有给指针开辟空间
int *p = new int(3); //给指针new个int空间，初始值为3
int *p = new int[3]; //给指针new出3个int空间

char *str="hello";//char指针可以直接赋值，但会出现警告
```

2.函数中的指针参数传递

```
void fun(int* a) { cout<<sizeof(a); }

int main(){
 int a[]={1,2,3};
 fun(a); //传过去的只是数组首地址
}
```

3.函数中使用&传递地址

```
void fun(string &str) { }

string str;
fun(str);
```

六、c++中的输入输出流

1.C++的输入输出分为三种：

(1)基于控制台的I/O

头文件	类型
iostream	istream从流中读取 ostream写到流中去 iostream对流进行读写，从istream和ostream派生

(2)基于文件的I/O

头文件	类型
fstream	ifstream从文件中读取，由istream派生 ofstream写到文件中去，由ostream派生 fstream 对文件进行读写，由iostream派生

(3)基于字符串的I/O

头文件	类型
sstream	istringstream从**string**对象中读取，由istream派生 ostringstream写到**string**对象中去，由ostream派生 stringstream对**string**对象进行读写，由iostream派生

2.是用 scanf/printf 还是用 cin/cout

scanf/printf 的速度是比 cin/cout 来的快，但是在任何情况下就一定要使用 scanf/printf 吗？这倒不是：

（1）当有大批量的输入输出的时候，请使用 scanf/printf

（2）当只有单个输入输出时候推荐使用 cin/cout ，因为这个没有严格的格式要求，可预防出现问题

（3）当不清楚输入输出个数的时候，可以使用 cin/cout 方法

3.加快cin的速度

这可以加快读取数据的速度，但是有一个非常不好的副作用就是不能与scanf这类的输入输出方法混用了

```
ios::sync_with_stdio(0); //关闭cin的同步，也就是解除与scanf的关联
cin.tie(0);
```

原来而cin，cout之所以效率低，是因为先把要输出的东西存入缓冲区，再输出，导致效率降低，而这段语句可以来打消iostream的输入 输出缓存，可以节省许多时间，使效率与scanf与printf相差无几

4.freopen从txt读取数据到输入流

```
freopen("in.txt", "r", stdin);
```