1. 一定记得先写

Int mian（）

{

Return 0；

}

因为你的编译器不会有问题，但是ccf测试结果成绩为0。

2.一定反复审题，审题后自己写出一到两个样例，而且还有注意Corner case。

3. cout << sizeof(long int);

cout << sizeof(long long);

Long int和int在32位时一样长的，只有4个字节，32位

Long long才是8个字节，64 位

4.遇到无输出的情况可能是：

声明数组超过上限2147483647个字节；

5.字符串的处理

strcat(s1, s2);  
连接字符串 s2 到字符串 s1 的末尾。 注意直接声明的数组a[100]，里面为，必须写入’\0’之后才能使用strcat；

调用函数对字符串进行处理后，若返回值为一个声明的局部数组的地址，那么返回的结果将会什么都没有，因为作用域的原因。

6.atoi 和itoa

头文件：Include<stdlib.h>

int atoi(const char \*nptr);

返回值为int型，可为正为负，若不能转化为数字，则返回0；

？？处不处理前导零啊（好像处理的）

头文件：<cstdlib>

string itoa(int x,char \*string,int jz)；

写入到数组中。

7.对于string类型，scanf和printf均不能使用：

全部使用scanf读入，但是如果读string，由于scanf不会改变size所以读不进去，采用如下：

char tmp[20];

scanf("%s",tmp);

**px.assign(tmp);**

对于输出应该使用：

printf("%s\n",**a.c\_str()**);

8.使用sscanf进行字符串的切割

int main()

{

char s[100];

int a;

sscanf("ab 123pc", "%s %dpc", s, &a);

cout << s << ' ' << a << endl; //ab 123

}

注意对于string px要使用

sscanf(px.c\_str(), "%d ", &ipn[i].num[0]);

9.结构体的声明：

struct ru{

string name;

vector <string> wos;

int flag;

};

10.传参和返回值为&的含义？

11.在判断条件千万不要写 “res=1” 和 “0<=res<=1”,

正确表示**res==1；和0<=res&&res<=1;**

12.字符串划分函数：

void split(string s, string c, vector<string> &res) {//注意此处的传参方式

int pos1, pos2;

pos1 = 0;

pos2 = s.find(c,0);

while (pos2 != string::npos) {

res.push\_back(s.substr(pos1, pos2 - pos1));

pos1 = pos2 + c.size();

pos2 = s.find(c, pos1);

}

if (pos1 != s.size())

{

res.push\_back(s.substr(pos1));

}

}

13.读取一行的输入以“\n” 作为分割，则使用getline；

需要注意getline前一个输入是否残留换行符“\r\n”或者“\n”;

char d[100];

cin.getline(d, 100);

14.不要仅使用vs进行编译，因为会自动添加头文件

15.对于**char \*型变量，不能使用“==”**，（但不会报错所以易错），要使用strcmp，需要加入string.h头文件。

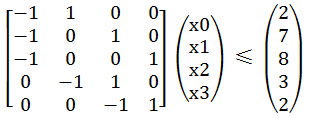
# 图算法

1. 最短生成树

181204，审题困难，最好假设它是，然后反推

2.查分约束

对于如若一个系统由n个变量和m个不等式组成，并且这m个不等式对应的系数矩阵中每一行有且仅有一个1和-1，其它的都为0，这样的系统称为**差分约束( difference constraints )**系统。引例中的不等式组可以表示成如图三-1-1的系数矩阵。



图三-1-1

特点：全是整数，可以将＜转化为≤

常用方法：

1. 对于想用查分约束的情况，若发现项数不满足上述情况，可以尝试前缀和的形式。

我们可以经b1+b2 b1+b2+b3 等转化为 s2-s0 s3-s0  这种前缀和的形式 然后再将答案转化为各个数字

2.注意不要漏掉特殊情况，比如某个变量等于确定的值，即

s[23] = T，它并不是一个不等式，我们需要将它也转化成不等式，由于设定s[-1] = 0，所以 s[23] - s[-1] = T，它可以转化成两个不等式：

      s[23] - s[-1] >= T

      s[-1] - s[23] >= -T

3.对于原本不是整数，后来变为整数的舍入等于，可以改写为≤

4. 如果给出的不等式有"<="也有">="，又该如何解决呢？很明显，首先需要关注最后的问题是什么，如果需要求的是两个变量差的最大值，那么需要将所有不等式转变成"<="的形式，建图后求最短路；相反，如果需要求的是两个变量差的最小值，那么需要将所有不等式转化成">="，建图后求最长路。怎么求最长路？

.记忆化搜索

3.. STL

1. 算法 头文件：<algorithm>，<numeric>和<functional>
2. 容器 头文件：<vector>,<list>,<deque>,<set>,<map>,<stack>和<queue>

下面详细介绍各种容器：

C++中有两种类型的容器：顺序容器和关联容器。顺序容器主要有vector、list、deque等。其中vector表示一段连续的内存，基于数组实现，list表示非连续的内存，基于链表实现，deque与vector类似，但是对首元素提供插入和删除的双向支持。关联容器主要有map和set。map是key-value形式，set是单值。map和set只能存放唯一的key，multimap和multiset可以存放多个相同的key。

容器类自动申请和释放内存，因此无需new和delete操作。

1. Vector

用到改知识的题目：（201709-4通信网络）

vector基于模板实现，需包含头文件vector。

|  |
| --- |
| //1.定义和初始化      vector<int> vec1;    //默认初始化，vec1为空      vector<int> vec2(vec1);  //使用vec1初始化vec2      vector<int> vec3(vec1.begin(),vec1.end());//使用vec1初始化vec2      vector<int> vec[100];    //每一个vec[i]是vector类型，vec[100]相当于一个二维数组。        //2.常用操作方法      vec1.push\_back(100);            //添加元素      int size = vec1.size();         //元素个数      bool isEmpty = vec1.empty();    //判断是否为空      cout<<vec1[0]<<endl;        //取得第一个元素      vec1.insert(vec1.end(),5,3);    //从vec1.back位置插入个值为的元素      //vec1.pop\_back();              //删除末尾元素      //vec1.erase(vec1.begin(),vec1.end());//删除之间的元素，其他元素前移      cout<<(vec1==vec2)?true:false;  //判断是否相等==、！=、>=、<=...  //vec1.clear();                 //清空元素        //3.遍历         int length = vec1.size();      for(int i=0;i<length;i++)      {         cout<<vec1[i];      }      cout<<endl<<endl; |

二、list

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。需要添加头文件list

|  |
| --- |
| //1.定义和初始化      list<int> lst1;          //创建空list      list<int> lst2(3);       //创建含有三个元素的list      list<int> lst3(3,2); //创建含有三个元素的list      list<int> lst4(lst2);    //使用lst2初始化lst4      list<int> lst5(lst2.begin(),lst2.end());  //同lst4        //2.常用操作方法      lst1.assign(lst2.begin(),lst2.end());  //分配值      lst1.push\_back(10);                    //添加值      lst1.pop\_back();                   //删除末尾值      lst1.begin();                      //返回首值的迭代器      lst1.end();                            //返回尾值的迭代器      lst1.clear();                      //清空值      bool isEmpty1 = lst1.empty();          //判断为空      lst1.erase(lst1.begin(),lst1.end());                        //删除元素      lst1.front();                      //返回第一个元素的引用      lst1.back();                       //返回最后一个元素的引用      lst1.insert(lst1.begin(),3,2);         //从指定位置插入个      lst1.rbegin();                         //返回第一个元素的前向指针      lst1.remove(2);                        //相同的元素全部删除      lst1.reverse();                        //反转      lst1.size();                       //含有元素个数      lst1.sort();                       //排序      lst1.unique();                         //删除相邻重复元素        //3.遍历      //迭代器法      for(list<int>::const\_iterator iter = lst1.begin();iter != lst1.end();iter++)      {         cout<<\*iter;      }      cout<<endl; |

三、deque

deque容器类与vector类似，支持随机访问和快速插入删除，它在容器中某一位置上的操作所花费的是线性时间。与vector不同的是，deque还支持从开始端插入数据：push\_front()。其余类似vector操作方法的使用。

1. Map

（201709-4通信网络）

C++中map容器提供一个键值对（key/value）容器，map与multimap差别仅仅在于multiple允许一个键对应多个值。需要包含头文件map。Map会根据key自动排序。

|  |
| --- |
| //1.定义和初始化      map<int,string> map1;                  //空map        //2.常用操作方法      map1[3] = "Saniya";                    //添加元素      map1.insert(map<int,string>::value\_type(2,"Diyabi"));//插入元素      //map1.insert(pair<int,string>(1,"Siqinsini"));      map1.insert(make\_pair<int,string>(4,"V5"));      string str = map1[3];                  //根据key取得value，key不能修改      map<int,string>::iterator iter\_map = map1.begin();//取得迭代器首地址      int key = iter\_map->first;             //取得eky      string value = iter\_map->second;       //取得value      map1.erase(iter\_map);                  //删除迭代器数据      map1.erase(3);                         //根据key删除value      map1.size();                       //元素个数      map1.empty();                       //判断空      map1.clear();                      //清空所有元素        //3.遍历      for(map<int,string>::iterator iter = map1.begin();iter!=map1.end();iter++)      {         int keyk = iter->first;         string valuev = iter->second;      } |

1. Stack

stack 模板类的定义在<stack>头文件中。  
stack 模板类需要两个模板参数，一个是元素类型，一个容器类型，但只有元素类型是必要的，在不指定容器类型时，默认的容器类型为deque。  
定义stack 对象的示例代码如下：  
stack<int> s1;  
stack<string> s2;  
stack 的基本操作有：  
入栈，如例：s.push(x);  
出栈，如例：s.pop();注意，出栈操作只是删除栈顶元素，并不返回该元素。  
访问栈顶，如例：s.top()  
判断栈空，如例：s.empty()，当栈空时，返回true。  
访问栈中的元素个数，如例：s.size()。

六、queue

queue 模板类的定义在<queue>头文件中。  
与stack 模板类很相似，queue 模板类也需要两个模板参数，一个是元素类型，一个容器类  
型，元素类型是必要的，容器类型是可选的，默认为deque 类型。  
定义queue 对象的示例代码如下：  
queue<int> q1;  
queue<double> q2;

queue 的基本操作有：  
入队，如例：q.push(x); 将x 接到队列的末端。  
出队，如例：q.pop(); 弹出队列的第一个元素，注意，并不会返回被弹出元素的值。  
访问队首元素，如例：q.front()，即最早被压入队列的元素。  
访问队尾元素，如例：q.back()，即最后被压入队列的元素。  
判断队列空，如例：q.empty()，当队列空时，返回true。  
访问队列中的元素个数，如例：q.size()

 Eg：

#include <cstdlib>  
#include <iostream>  
#include <queue>

using namespace std;

int main()  
{  
    int e,n,m;  
    queue<int> q1;  
    for(int i=0;i<10;i++)  
       q1.push(i);  
    if(!q1.empty())  
    cout<<"dui lie  bu kong\n";  
    n=q1.size();  
    cout<<n<<endl;  
    m=q1.back();  
    cout<<m<<endl;  
    for(int j=0;j<n;j++)  
    {  
       e=q1.front();  
       cout<<e<<" ";  
       q1.pop();  
    }  
    cout<<endl;  
    if(q1.empty())  
    cout<<"dui lie  bu kong\n";  
    system("PAUSE");  
    return 0;  
}

七、priority\_queue

在<queue>头文件中，还定义了另一个非常有用的模板类priority\_queue(优先队列）。优先队列与队列的差别在于优先队列不是按照入队的顺序出队，而是按照队列中元素的优先权顺序出队（默认为大者优先，也可以通过指定算子来指定自己的优先顺序）。  
priority\_queue 模板类有三个模板参数，第一个是元素类型，第二个容器类型，第三个是比较算子。其中后两个都可以省略，默认容器为vector，默认算子为less，即小的往前排，大的往后排（出队时序列尾的元素出队）。  
定义priority\_queue 对象的示例代码如下：  
priority\_queue<int> q1; priority\_queue< pair<int, int> > q2; // 注意在两个尖括号之间一定要留空格。  
priority\_queue<int, vector<int>, greater<int> > q3; // 定义小的先出队  
priority\_queue 的基本操作与queue 相同。初学者在使用priority\_queue 时，最困难的可能就是如何定义比较算子了。如果是基本数据类型，或已定义了比较运算符的类，可以直接用STL 的less 算子和greater算子——默认为使用less 算子，即小的往前排，大的先出队。如果要定义自己的比较算子，方法有多种，这里介绍其中的一种：重载比较运算符。优先队列试图将两个元素x 和y 代入比较运算符(对less 算子，调用x<y，对greater 算子，调用x>y)，若结果为真，则x 排在y 前面，y 将先于x 出队，反之，则将y 排在x 前面，x 将先出队。

看下面这个简单的示例：  
#include <iostream>

#include <queue>  
using namespace std;  
class T  
{  
　　public:  
　　int x, y, z;  
　　T(int a, int b, int c):x(a), y(b), z(c)  
　　{  
　　}  
};  
bool operator < (const T &t1, const T &t2)  
{  
　　return t1.z < t2.z; // 按照z 的顺序来决定t1 和t2 的顺序  
}  
main()  
{  
　　priority\_queue<T> q;  
　　q.push(T(4,4,3));  
　　q.push(T(2,2,5));  
　　q.push(T(1,5,4));  
　　q.push(T(3,3,6));  
　　while (!q.empty())  
　　{  
　　　　T t = q.top(); q.pop();  
　　　　cout << t.x << " " << t.y << " " << t.z << endl;  
　　}  
　　return 1;  
}  
输出结果为(注意是按照z 的顺序从大到小出队的)：  
3 3 6  
2 2 5  
1 5 4  
4 4 3  
再看一个按照z 的顺序从小到大出队的例子：  
#include <iostream>  
#include <queue>  
using namespace std;  
class T  
{  
public:  
　　int x, y, z;  
　　T(int a, int b, int c):x(a), y(b), z(c)  
　　{  
　　}  
};  
bool operator > (const T &t1, const T &t2)  
{  
　　return t1.z > t2.z;  
}  
main()  
{  
　　priority\_queue<T, vector<T>, greater<T> > q;  
　　q.push(T(4,4,3));  
　　q.push(T(2,2,5));  
　　q.push(T(1,5,4));  
　　q.push(T(3,3,6));  
　　while (!q.empty())  
　　{  
　　　　T t = q.top(); q.pop();  
　　　　cout << t.x << " " << t.y << " " << t.z << endl;  
　　}  
　　return 1;  
}  
输出结果为：  
4 4 3  
1 5 4  
2 2 5  
3 3 6  
如果我们把第一个例子中的比较运算符重载为：  
bool operator < (const T &t1, const T &t2)  
{  
　　return t1.z > t2.z; // 按照z 的顺序来决定t1 和t2 的顺序  
}  
则第一个例子的程序会得到和第二个例子的程序相同的输出结果。

一、string的初始化

首先，为了在程序中使用string类型，必须包含头文件 <string>。如下：

#include <string>

注意这里不是string.h，string.h是C字符串头文件。

string类是一个模板类，位于名字空间std中，通常为方便使用还需要增加：

using namespace std;

声明一个字符串变量很简单：

string str;

测试代码:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | #include <iostream>  #include <string>  using namespace std;  int main ( )  {      string str;  //定义了一个空字符串str      str = "Hello world";   // 给str赋值为"Hello world"      char cstr[] = "abcde";  //定义了一个C字符串      string s1(str);       //调用复制构造函数生成s1，s1为str的复制品      cout<<s1<<endl;      string s2(str,6);     //将str内，开始于位置6的部分当作s2的初值      cout<<s2<<endl;      string s3(str,6,3);  //将str内，开始于6且长度顶多为3的部分作为s3的初值          cout<<s3<<endl;      string s4(cstr);   //将C字符串作为s4的初值      cout<<s4<<endl;      string s5(cstr,3);  //将C字符串前3个字符作为字符串s5的初值。      cout<<s5<<endl;      string s6(5,'A');  //生成一个字符串，包含5个'A'字符      cout<<s6<<endl;      string s7(str.begin(),str.begin()+5); //区间str.begin()和str.begin()+5内的字符作为初值      cout<<s7<<endl;      return 0;  } |

程序执行结果为：

Hello world

world

wor

abcde

abc

AAAAA

Hello

二、string的比较等操作

你可以用 ==、>、<、>=、<=、和!=比较字符串，可以用+或者+=操作符连接两个字符串，并且可以用[]获取特定的字符。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | #include <iostream>  #include <string>  using namespace std;  int main()  {      string str;      cout << "Please input your name:"<<endl;      cin >> str;      if( str == "Li" )   // 字符串相等比较          cout << "you are Li!"<<endl;      else if( str != "Wang" )  // 字符串不等比较          cout << "you are not Wang!"<<endl;      else if( str < "Li")     // 字符串小于比较，>、>=、<=类似          cout << "your name should be ahead of Li"<<endl;      else          cout << "your name should be after of Li"<<endl;      str += ", Welcome!";  // 字符串+=      cout << str<<endl;      for(int i = 0 ; i < str.size(); i ++)          cout<<str[i];  // 类似数组，通过[]获取特定的字符      return 0;  } |

程序执行结果为：

Please input your name:

Zhang↙

you are not Wang!

Zhang, Welcome!

Zhang, Welcome!

上例中，“ cout<< str[i]; ”可换为： cout<< str.at(i);

三、string特性描述

可用下列函数来获得string的一些特性：

int capacity()const; //返回当前容量（即string中不必增加内存即可存放的元素个数）

int max\_size()const; //返回string对象中可存放的最大字符串的长度

int size()const; //返回当前字符串的大小

int length()const; //返回当前字符串的长度

bool empty()const; //当前字符串是否为空

void resize(int len,char c); //把字符串当前大小置为len，多去少补，多出的字符c填充不足的部分

测试代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | #include <iostream>  #include <string>  using namespace std;  int main()  {      string str;          if (str.empty())          cout<<"str is NULL."<<endl;          else          cout<<"str is not NULL."<<endl;      str = str + "abcdefg";      cout<<"str is "<<str<<endl;          cout<<"str's size is "<<str.size()<<endl;  　　     cout<<"str's capacity is "<<str.capacity()<<endl;      cout<<"str's max size is "<<str.max\_size()<<endl;      cout<<"str's length is "<<str.length()<<endl;      str.resize(20,'c');      cout<<"str is "<<str<<endl;      str.resize(5);      cout<<"str is "<<str<<endl;      return 0;  } |

程序执行结果为：

str is NULL.

str is abcdefg

str's size is 7

str's capacity is 15

str's max size is 4294967294

str's length is 7

str is abcdefgccc

str is abcde

四、string的查找

由于查找是使用最为频繁的功能之一，string提供了非常丰富的查找函数：（注：string::npos）

size\_type find( const basic\_string &str, size\_type index ); //返回str在字符串中第一次出现的位置（从index开始查找），如果没找到则返回string::npos

size\_type find( const char \*str, size\_type index ); // 同上

size\_type find( const char \*str, size\_type index, size\_type length ); //返回str在字符串中第一次出现的位置（从index开始查找，长度为length），如果没找到就返回string::npos

size\_type find( char ch, size\_type index ); // 返回字符ch在字符串中第一次出现的位置（从index开始查找），如果没找到就返回string::npos

注意：查找字符串a是否包含子串b,不是用 strA.find(strB) > 0 而是 strA.find(strB) != string:npos 这是为什么呢？（初学者比较容易犯的一个错误）本部分参考自[web100](https://www.cnblogs.com/web100/archive/2012/12/02/cpp-string-find-npos.html)与[luhao1993](https://blog.csdn.net/luhao1993/article/details/17690453)

　　先看下面的代码

int idx = str.find("abc");

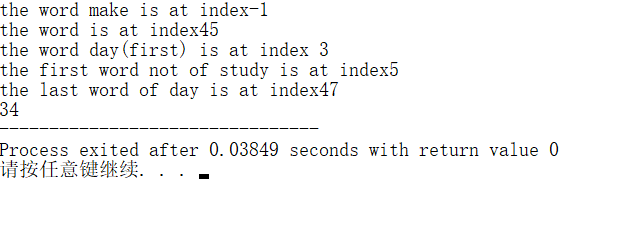
if (idx == string::npos);

　　上述代码中，idx的类型被定义为int，这是错误的，即使定义为 unsigned int 也是错的，它必须定义为 string::size\_type。npos 是这样定义的： static const size\_type npos = -1; 因为 string::size\_type (由字符串配置器 allocator 定义) 描述的是 size，故需为无符号整数型别。因为缺省配置器以型别 size\_t 作为 size\_type，于是 -1 被转换为无符号整数型别，npos 也就成了该型别的最大无符号值。不过实际数值还是取决于型别 size\_type 的实际定义。不幸的是这些最大值都不相同。事实上，(unsigned long)-1 和 (unsigned short)-1 不同(前提是两者型别大小不同)。因此，比较式 idx == string::npos 中，如果 idx 的值为-1，由于 idx 和字符串string::npos 型别不同，比较结果可能得到 false。因此要想判断 find()等查找函数的结果是否为npos，最好的办法是直接比较。

测试代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | #include<iostream>  #include<string>  using namespace std;  int main(){      int loc;      string s="study hard and make progress everyday! every day!!";      loc=s.rfind("make",10);      cout<<"the word make is at index"<<loc<<endl;//-1表示没找到      loc=s.rfind("make");//缺省状态下，从最后一个往前找      cout<<"the word make is at index"<<loc<<endl;      loc=s.find\_first\_of("day");      cout<<"the word day(first) is at index "<<loc<<endl;      loc=s.find\_first\_not\_of("study");      cout<<"the first word not of study is at index"<<loc<<endl;      loc=s.find\_last\_of("day");      cout<<"the last word of day is at index"<<loc<<endl;      loc=s.find("day");//缺陷状态下从第一个往后找      cout<<loc;      return 0;  } |

运行结果：



五、其他常用函数

string &insert(int p,const string &s); //在p位置插入字符串s

string &replace(int p, int n,const char \*s); //删除从p开始的n个字符，然后在p处插入串s

string &erase(int p, int n); //删除p开始的n个字符，返回修改后的字符串

string substr(int pos = 0,int n = npos) const; //返回pos开始的n个字符组成的字符串

void swap(string &s2); //交换当前字符串与s2的值

string &append(const char \*s); //把字符串s连接到当前字符串结尾

void push\_back(char c) //当前字符串尾部加一个字符c

const char \*data()const; //返回一个非null终止的c字符数组，data():与c\_str()类似，用于string转const char\*其中它返回的数组是不以空字符终止,

const char \*c\_str()const; //返回一个以null终止的c字符串，即c\_str()函数返回一个指向正规C字符串的指针, 内容与本string串相同,用于string转const char\*

测试代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | #include <iostream>  #include <string>  using namespace std;  int main()  {      string str1 = "abc123defg";      string str2 = "swap!";      cout<<str1<<endl;      cout<<str1.erase(3,3)<<endl;  //从索引3开始的3个字符,即删除掉了"123"      cout<<str1.insert(0,"123")<<endl; //在头部插入      cout<<str1.append("123")<<endl;   //append()方法可以添加字符串      str1.push\_back('A');  //push\_back()方法只能添加一个字符      cout<<str1<<endl;      cout<<str1.replace(0,3,"hello")<<endl; //即将索引0开始的3个字符替换成"hello"      cout<<str1.substr(5,7)<<endl; //从索引5开始7个字节      str1.swap(str2);      cout<<str1<<endl;      const char\* p = str.c\_str();      printf("%s\n",p);      return 0;  } |

程序执行结果为：

abc123defg

abcdefg

123abcdefg

123abcdefg123

123abcdefg123A

helloabcdefg123A

abcdefg

swap!

swap!

买菜代码

#include<stdio.h>

#include<algorithm>

#include <vector>

#define MAX\_N 2005

using namespace std;

struct times{

int st;

int ft;

times (int aa,int bb):st(aa),ft(bb){}

};

bool compare(times a, times b)

{

return a.st<b.st;

}

int main()

{

int n = 0;

int i;

int lasttime=0;

int talk =0;

int min;

int a1,b1;

vector <times> a;

scanf("%d",&n);

for (i=0;i<2\*n;i++)

{

scanf("%d %d",&a1,&b1);

a.push\_back(times(a1,b1));

}

sort(a.begin(),a.begin()+2\*n,compare);

for (i= 0;i<2\*n;i++)

{

if (a[i].st<lasttime)

{

min =a[i].ft<lasttime? a[i].ft:lasttime;

talk +=(min- a[i].st);

}

if (a[i].ft>lasttime)

lasttime = a[i].ft;

}

printf("%d ",talk);

return 0;

}

I）Sort函数包含在头文件为#include<algorithm>的c++标准库中，调用标准库里的排序方法可以不必知道其内部是如何实现的，只要出现我们想要的结果即可！

II）Sort函数有三个参数：

（1）第一个是要排序的数组的起始地址。

（2）第二个是结束的地址（最后一位要排序的地址）

（3）第三个参数是排序的方法，可以是从大到小也可是从小到大，还可以不写第三个参数，此时默认的排序方法是从小到大排序。

Sort函数使用模板:

Sort(start,end,排序方法)

下面就具体使用sort（）函数结合对数组里的十个数进行排序做一个说明！

例一：sort函数没有第三个参数，实现的是从小到大

[复制代码](javascript:void(0);)

1 #include<iostream>

2

3 #include<algorithm>

4 using namespace std;

5 int main()

6 {

7 int a[10]={9,6,3,8,5,2,7,4,1,0};

8 for(int i=0;i<10;i++)

9 cout<<a[i]<<endl;

10 sort(a,a+10);

11 for(int i=0;i<10;i++)

12 cout<<a[i]<<endl;

13 return 0;

14 }

[复制代码](javascript:void(0);)

例二

通过上面的例子，会产生疑问：要实现从大到小的排序肿么办？

 这就如前文所说需要在sort（）函数里的第三个参数里做文章了，告诉程序我要从大到小排序！

需要加入一个比较函数 complare(),此函数的实现过程是这样的

|  |  |
| --- | --- |
| 1  2  3  4 | bool complare(int a,int b)  {      return a>b;  } |

　　这就是告诉程序要实现从大到小的排序的方法！

[复制代码](javascript:void(0);)

1 #include<iostream>

2 #include<algorithm>

3 using namespace std;

4 bool complare(int a,int b)

5 {

6 return a>b;

7 }

8 int main()

9 {

10 int a[10]={9,6,3,8,5,2,7,4,1,0};

11 for(int i=0;i<10;i++)

12 cout<<a[i]<<endl;

13 sort(a,a+10,complare);//在这里就不需要对complare函数传入参数了，

14 //这是规则

15 for(int i=0;i<10;i++)

16 cout<<a[i]<<endl;

17 return 0;

18 }

[复制代码](javascript:void(0);)

 假设自己定义了一个结构体node

[复制代码](javascript:void(0);)

1 struct node

2 {

3 int a;

4 int b;

5 double c;

6 }

[复制代码](javascript:void(0);)

有一个node类型的数组node arr[100]，想对它进行排序：先按a值升序排列，如果a值相同，再按b值降序排列，如果b还相同，就按c降序排列。就可以写这样一个比较函数：

以下是代码片段：

[复制代码](javascript:void(0);)

1 bool cmp(node x,node y)

2 {

3 if(x.a!=y.a) return x.a<y.a;

4 if(x.b!=y.b) return x.b>y.b;

5 return x.c>y.c;

6 }

[复制代码](javascript:void(0);)

URL

#include<iostream>

2 #include<sstream>

3 #include<cstring>

4 #include<string>

5 #include<cstdio>

6 #include<vector>

7 #include<algorithm>

8 #define LL unsigned long long

9 using namespace std;

10 const int maxn=101;

11 int cnt1[maxn],isp,sp[maxn]; //cnt1[i]--第i个字符串的项数，isp--待查询的字符串末尾是否为‘/’，sp[i]--第i个字符串末尾是否为‘/’

12 string str[maxn],str1[maxn],str2; //str[i]--第i个字符串的配置信息，str1[i]--第i个映射规则，str2--当前需要被查询的字符串

13 string sp1[maxn][maxn],sp2[maxn]; //sp1[i]--保存第i个字符串的所有项，sp2--保存当前被查询字符串的所有项。

14 string is\_num(string s){ //判断某一项是否为整数：是--去掉前导0并返回整数；不是--返回“-”

15 int ok=0;

16 string ss="";

17 int len=s.length();

18 for(int i=0;i<len;i++){

19 if(s[i]<'0'||s[i]>'9')return "-";

20 if(ok||s[i]!='0')ss+=s[i],ok=1;

21 }

22 if(ss=="")ss="0";

23 return ss;

24 }

25 void getinfo(string s,string s1[],int &f,int &t){ //分离并保存一个字符串的所有项，标记末尾是否为‘/’

26 f=t=0;

27 int len=s.length();

28 if(s[len-1]=='/')f=1;

29 for(int p=0;p<len;p++){

30 if(s[p]=='/')s[p]=' ';

31 }

32 string ss;

33 stringstream in(s);

34 while(in>>ss)s1[t++]=ss;

35 }

36 bool match(int t,int j,string &s){ //判断被查询字符串与第j个规则是否能匹配

37 s="";

38 int p1=0,p2=0;

39 while(p1<t&&p2<cnt1[j]){

40 if(sp2[p1]==sp1[j][p2]);

41 else if(sp1[j][p2]=="<int>"){

42 string f=is\_num(sp2[p1]);

43 if(f=="-"){return 0;}

44 s+=" "+f;

45 }

46 else if(sp1[j][p2]=="<str>"){s+=" "+sp2[p1];}

47 else if(sp1[j][p2]=="<path>"){ //<path>直接全部加上

48 s+=" "+sp2[p1++];

49 while(p1<t)s+="/"+sp2[p1++];

50 if(isp)s+='/';

51 return 1;

52 }

53 else return 0;

54 p1++;p2++;

55 }

56 if(isp^sp[j])return 0; //末尾判断--同时有‘/’或同时无‘/’才能匹配

57 if(p1!=t||p2!=cnt1[j])return 0; //完全匹配

58 return 1;

59 }

60 int main(){

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

62 int n,m;

63 cin>>n>>m;

64 for(int i=0;i<n;i++){

65 cin>>str1[i]>>str[i];

66 getinfo(str1[i],sp1[i],sp[i],cnt1[i]); //string, split, 末尾是否'/', 字符串数量

67 }

68 for(int i=0;i<m;i++){

69 string ans;

70 int cnt=0;isp=0;

71 cin>>str2;

72 getinfo(str2,sp2,isp,cnt);

73 bool ok=0;

74 for(int j=0;j<n;j++){

75 if(match(cnt,j,ans)){

76 cout<<str[j]<<ans<<endl;;

77 ok=1;break;

78 }

79 }

80 if(!ok)cout<<404<<endl;

81 }

82 return 0;