**一、find（）函数：查找一个字符串是否包含特定另一个字符串**

find（）函数带有一个参数，两个参数和三个参数的情况。

find函数带一个参数，就是直接查找；

find函数带有两个参数，就是在被查找的字符串的某个特定位置开始查找；

find函数带有三个参数，就是在被查找的字符串的某个特定位置开始查找：查找字符串的前n个字符组成的字符串！

#include <iostream>

#include <cstring>

using namespace std;

int main(){

string s1="The first step is as good as half over.";

string s2 = "step";

//返回查找到的字符（串）s2在s1中的首地址

int found = s1.find(s2);

cout<<found<<endl;

//---------------------------------------

//返回查找到的字符（串）s2在s1中的首地址,此时，是从s1的第 n个字符开始寻找

found = s1.find(s2,3);

cout<<found<<endl;

//如果不存在这个字符串，那么返回的是-1

found = s1.find(s2,14);

cout<<found<<endl;

//-----------------------------------------

found = s1.find("step");

cout<<found<<endl;

//-------------------------------------------

//表示从 s1 中第9个字符开始寻找 “ism”的前两个字符

found = s1.find("ism",9,2);

cout<<found<<endl;

//-------------------------------------------

return 0;

}

运行结果：

10

10

-1

10

15

**二、find\_first\_of（）函数：查找字符第一次出现在被查找字符串中的位置**

函数find\_first\_of() 查找在字符串中第1个出现的字符c，而函数find\_last\_of()查找最后一个出现的c。匹配的位置是返回值。如果没有匹配发生，则函数返回-1.

int find\_first\_of(char c, int start = 0):查找字符串中第1个出现的c,由位置start开始。

如果有匹配，则返回匹配位置；否则，返回-1。默认情况下，start为0，函数搜索整个字符串。

#include <iostream>

#include <cstring>

using namespace std;

int main(){

string s1="The first step is as good as half over.";

string s2 = "step";

//字符第一次出现在字符串中的位置

int found = s1.find\_first\_of("a");

cout<<found<<endl;

found = s1.find\_first\_of("as");

cout<<found<<endl;

cout<<"-----------------------"<<endl;

int index=0;

while((index=s1.find\_first\_of("s",index))!=-1) {

cout<<index<<endl;

index++;

}

}

运行结果：

18

7

--------------------------

7

10

16

19

27

**三、find\_last\_of（）函数**

int find\_last\_of(char c):查找字符串中最后一个出现的c。有匹配，则返回匹配位置；否则返回-1.

该搜索在字符末尾查找匹配，所以起始位置默认为字符串末尾，除非有参数指定起始位置。

#include <iostream>

#include <cstring>

using namespace std;

int main(){

string s1="The first step is as good as half over.";

string s2 = "s";

//从后往前找，与first相反而已

int index=s1.length();

while((index=s1.find\_last\_of("s",index))!=-1) {

cout<<index<<endl;

index--;

}

}

运行结婚：

27

19

16

10

7

**四、string substr(int start,int count)函数**

带一个参数和带两个参数

#include <iostream>

#include <cstring>

using namespace std;

int main(){

string s1="The first step is as good as half over.";

string s2 = s1.substr(2,3);

cout<<s2<<endl;

s2=s1.substr(0,3);

cout<<s2<<endl;

//-----------------------------------------

s2=s1.substr(4);

cout<<s2<<endl;

}

运行结果：

e f

The

first step is as good as half over.

五、insert（）函数

字符连接(+)是在字符串尾添加字符串。

insert()函数扩展了这个能力，允许在任意位置添加字符串。

void insert(int statr,const string& s): 将子串s放入字符串中，起始于位置start。插入操作增加了原始字符串的长度。

#include <iostream>

#include <cstring>

using namespace std;

int main(){

string s1="The first step ";

string s2 = "is as good as half over.";

string s= s1+s2;

cout<<s<<endl;

s1.insert(2,s2);

cout<<s1<<endl;

}

运行结果：

The first step is as good as half over.

This as good as half over.e first step

六、void erase(int start=0,int count=-1)函数:

从start开始，从字符串中删除count个字符。

如果现有的字符串少于count个字符，或者count为-1，则删除到字符串尾部的所有字符。默认情况下,start为0，函数从字符串是起始位置开始删除字符串。默认情况下，函数也删除到字符串尾。

**SPFA算法**

/\*

test input:

5 7

0 1 3

0 2 2

1 3 1

3 0 4

3 2 6

3 4 4

4 2 4

5 7

0 4 10

0 1 2

1 4 7

1 2 3

4 2 6

2 3 4

3 4 5

\*/

#include <iostream>

#include <string>

#include <cstring>

#include <vector>

#include <stdio.h>

#include <queue>

#define MAXM 100

#define INF (1000)

using namespace std;

struct EDGE{

int u, v, w, next;

EDGE() {}

EDGE(int u, int v, int w, int next):

u(u), v(v), w(w), next(next) {}

}edge[MAXM];

int head[MAXM];

int edgeCount = 0;

int d[MAXM];

void addEdge(int u, int v, int w)

{

edge[edgeCount] = EDGE(u, v, w, head[u]);

head[u] = edgeCount++;

}

bool spfa(int s, int n) //s为源节点号，n为点数

{

queue <int> que;

int visit[MAXM], inq[MAXM];

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

{

visit[i] = 0;

inq[i] = 0;

d[i] = (i==s)?0:INF;

}

que.push(s);

inq[s] = 1;

while (!que.empty())

{

int node = que.front();

que.pop();

EDGE edge1 = edge[head[node]]; //important head[node]

int u, v, w;

u = edge1.u;

v = edge1.v;

w = edge1.w;

inq[u] = 0;

if (visit[u]++ > n) //important

{

return true;

}

for (int e=head[u]; e!=-1; e=edge[e].next)

{

EDGE edge2 = edge[e];

u = edge2.u;

v = edge2.v;

w = edge2.w;

if (d[u]+w < d[v])

{

d[v] = d[u]+w;

if (!inq[v])

{

inq[v] = 1;

que.push(v);

}

}

}

}

return false;

}

int main()

{

memset((void \*)head, -1, sizeof(head));

int n, e, u, v, w;

cin >> n >> e;

while (e--)

{

cin >> u >> v >> w;

addEdge(u, v, w);

}

spfa(0, n);

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

{

cout << "0->" << i << ": " << d[i] << endl;

}

return 0;

}

**URL映射**

#include <iostream>

#include <string>

using namespace std;

int is\_int(string str)

{

int i;

for (i=0; i<str.length(); i++)

{

if (str[i] < '0' || str[i] > '9')

{

return 0;

}

}

return 1;

}

int match(string rule, string name, string url)

{

int flag = 1, final = 0;

int rpos1, rpos2, upos1, upos2;

string str1, str2;

string re;

rpos1 = rpos2 = upos1 = upos2 = 0;

//get the first str to match

if (str1[rpos1] == '/')

{

if (str2[upos1] != '/')

{

return 0;

}

rpos1 = upos1 = 1;

rpos2 = rule.find('/', rpos1+1);

upos2 = url.find('/', upos1+1);

str1 = rule.substr(rpos1+1, rpos2-rpos1-1);

str2 = url.substr(upos1+1, upos2-upos1-1);

}

else

{

if (str2[upos1] == '/')

{

return 0;

}

rpos2 = rule.find('/', rpos1+1);

upos2 = url.find('/', upos1+1);

if (rpos2 == -1)

{

str1 = rule.substr(rpos1+1, rule.length()-1);

}

else

{

str1 = rule.substr(rpos1+1, rpos2-rpos1-1);

}

if (upos2 == -1)

{

str2 = url.substr(upos1+1, url.length()-1);

}

else

{

str2 = url.substr(upos1+1, upos2-upos1-1);

}

}

while (flag)

{

if (str1 == "<str>")

{

if (str2 == "")

{

return 0;

}

re = re + " " + str2;

}

else if (str1 == "<int>")

{

if (!is\_int(str2))

{

return 0;

}

while (str2.find('0') == 0)

{

str2 = str2.substr(1);

}

re = re + " " + str2;

}

else if (str1 == "<path>")

{

if(str2 == "")

{

return 0;

}

//match

str2 = url.substr(upos1+1);

re = re + " " + str2;

re = name + re;

cout << re << endl;

return 1;

}

else if (str1 != str2)

{

return 0;

}

if (rpos2 == -1)

{

if (upos2 != -1)

{

return 0;

}

}

rpos1 = rpos2;

upos1 = upos2;

rpos2 = rule.find('/', rpos1+1);

upos2 = url.find('/', upos1+1);

if (rpos1 == -1)

{

if (upos1 != -1)

{

return 0;

}

else

{

//match

re = name + re;

cout << re << endl;

return 1;

}

}

if (rpos2 == -1)

{

if (rpos1 == rule.length()-1) //rule has ended

{

if (upos1 == url.length()-1)

{

//match

re = name + re;

cout << re << endl;

return 1;

}

else

{

return 0;

}

}

else

{

if (upos1 == url.length()-1) //url ended

{

return 0;

}

else

{

str1 = rule.substr(rpos1+1, rule.length()-1);

if (upos2 == -1)

{

str2 = url.substr(upos1+1, url.length()-1);

}

else

{

str2 = url.substr(upos1+1, upos2-upos1-1);

}

continue;

}

}

}

if (upos2 == -1)

{

return 0;

}

str1 = rule.substr(rpos1+1, rpos2-rpos1-1);

str2 = url.substr(upos1+1, upos2-upos1-1);

}

return 0;

}

int is\_legal(string url)

{

int i=0, j=url.length();

for (i=0; i<j; i++)

{

char c = url[i];

if ((c=='/') || (c>='a'&&c<='z') || (c>='A'&&c<='Z') || (c<='9'&&c>='0') || (c=='-') || (c=='\_') || (c=='.'))

{

continue;

}

return 0;

}

return 1;

}

int main() {

int n, m;

int find;

cin >> n >> m;

string rule[n], name[n];

string url[m];

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

{

cin >> rule[i] >> name[i];

}

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

{

cin >> url[i];

}

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

{

find = 0;

if (!is\_legal(url[i]))

{

cout << "404" << endl;

continue;

}

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

{

if (match(rule[j], name[j], url[i]))

{

find = 1;

break;

}

}

if (!find) cout << "404" << endl;

}

return 0;

}

**棋局评估**

#include <iostream>

#include <string>

using namespace std;

int mp[3][3];

int win()

{

int i, j;

for (i=0; i<3; i++)

{

//column

if (mp[i][0]==mp[i][1] && mp[i][1]==mp[i][2])

{

if (mp[i][0] != 0)

{

return mp[i][0];

}

}

//row

if (mp[0][i]==mp[1][i] && mp[1][i]==mp[2][i])

{

if (mp[0][i] != 0)

{

return mp[0][i];

}

}

}

//¶Ô½ÇÏß

if (mp[1][1] != 0)

{

if (mp[0][0]==mp[1][1] && mp[1][1]==mp[2][2])

{

return mp[1][1];

}

if (mp[0][2]==mp[1][1] && mp[1][1]==mp[2][0])

{

return mp[1][1];

}

}

for (i=0; i<3; i++)

{

for (j=0; j<3; j++)

{

if (mp[i][j] == 0)

{

return 0;

}

}

}

return -1;//Æ½¾Ö

}

int DFS(int turn)

{

int i, j;

int re = win();

int score = 0;

if (re != 0)

{

if (re == -1)

{

return 0;

}

else

{

for (i=0; i<3; i++)

{

for (j=0; j<3; j++)

{

if (mp[i][j] == 0)

{

score++;

}

}

}

if (re == 1) //Alice wins

{

score = score+1;

}

else

{

score = -(score+1);

}

return score;

}

}

if (turn == 1)

{

score = -100;

}

else

{

score = 100;

}

for (i=0; i<3; i++)

{

for (j=0; j<3; j++)

{

if (mp[i][j] != 0)

{

continue;

}

if (turn == 1)

{

mp[i][j] = 1;

score = max(score, DFS(0));

}

else

{

mp[i][j] = 2;

score = min(score, DFS(1));

}

mp[i][j] = 0;

}

}

return score;

}

int main() {

int T;

cin >> T;

while(T--)

{

for (int i=0; i<3; i++)

{

for (int j=0; j<3; j++)

{

cin >> mp[i][j];

}

}

cout << DFS(1) << endl;

}

return 0;

}

**元素选择器**

#include <iostream>

#include <string>

#include <cstring>

#include <vector>

#include <stdio.h>

using namespace std;

string getTag(string str)

{

while (str[0] == '.')

{

str = str.substr(1);

}

int pos = str.find(' ');

if (pos == -1)

{

return str.substr(0);

}

else

{

return str.substr(0, pos);

}

}

string getId(string str)

{

int pos1 = 0;

pos1 = str.find('#');

if (pos1 == -1)

return string("");

return str.substr(pos1);

}

int getLevel(string str)

{

int count = 0;

while (str[count] == '.')

{

count++;

}

return count / 2;

}

int myFind(vector <string> document, vector <string> str, int startLine, int goalNum)

{

if (goalNum >= str.size())

{

return startLine;

}

int i = 0;

string tag, id;

string startStr = document[startLine];

string curStr = "";

string goalStr = str[goalNum];

int startLevel = getLevel(startStr); //祖先等级

int curLevel = 0;

if (goalStr[0] == '#') //select id

{

if (str.size() == 1) //只查找一个时不能重复选中一个

{

i = startLine + 1;

}

else if (goalNum > 0)

{

if (str[goalNum - 1] == str[goalNum]) //防止div div p重复选中

{

i = startLine + 1;

}

else

{

i = startLine;

}

}

else

{

i = startLine;

}

for (; i < document.size(); i++)

{

curStr = document[i];

curLevel = getLevel(curStr);

if (curLevel < startLevel)

{

return 0;

}

if (goalStr == getId(curStr))

{

return myFind(document, str, i, goalNum + 1);

}

}

}

else //select tag

{

if (str.size() == 1) //只查找一个时不能重复选中一个

{

i = startLine + 1;

}

else if (goalNum > 0)

{

if (str[goalNum - 1] == str[goalNum]) //防止div div p重复选中

{

i = startLine + 1;

}

else

{

i = startLine;

}

}

else

{

i = startLine;

}

for (; i < document.size(); i++)

{

curStr = document[i];

curLevel = getLevel(curStr);

if (curLevel < startLevel)

{

return 0;

}

if (goalStr == getTag(curStr))

{

return myFind(document, str, i, goalNum + 1);

}

}

}

return 0;

}

void select(vector <string> document, string selector)

{

vector <string> str; //选择器数组

vector <int> number; //行号数组

int x = 0;

int pos1 = 0, pos2 = 0; //pos1:选择器开始下标 pos2:选择器后紧接着的空格下标

while (1)

{

pos2 = selector.find(' ', pos1 + 1);

if (pos2 == -1)

{

str.push\_back(selector.substr(pos1));

break;

}

str.push\_back(selector.substr(pos1, pos2 - pos1));

pos1 = pos2 + 1;

}

int line = 1;

number.clear();

if (str.size() == 1)

{

string key = str[0]; //最后一个选择器

string tag, id;

for (int i = 1; i < document.size(); i++)

{

if (key[0] == '#') //不存在同id

{

id = getId(document[i]);

if (id == key)

{

number.push\_back(i);

break;

}

}

else

{

tag = getTag(document[i]);

if (tag == key)

{

number.push\_back(i);

}

}

}

}

else

{

int end = 0;

while (line && !end)

{

line = myFind(document, str, line, 0);

if (line)

{

number.push\_back(line);

//贪心

int level1 = getLevel(document[line]);

if (line < document.size() - 1)

{

line++;

int level2 = getLevel(document[line]);

string key = str[str.size() - 1]; //最后一个选择器

string tag, id;

while (level2 >= level1)

{

if (key[0] == '#') //不存在同id

{

break;

}

else

{

tag = getTag(document[line]);

if (tag == key)

{

number.push\_back(line);

}

line++;

if (line == document.size())

{

end = 1;

break;

}

level2 = getLevel(document[line]);

}

}

}

}

}

}

cout << number.size();

for (int i = 0; i < number.size(); i++)

{

cout << " " << number[i];

}

cout << endl;

}

int main() {

int n, m;

int i;

std::string name;

vector <string> document;

vector <string> selector;

cin >> n >> m;

selector.resize(m);

document.push\_back("ignore");

document.resize(n + 1);

getchar();

for (i = 1; i < n + 1; i++)

{

getline(cin, document[i]);

name = document[i];

}

for (i = 0; i < m; i++)

{

getline(cin, selector[i]);

name = selector[i];

}

for (i = 0; i < m; i++)

{

select(document, selector[i]);

}

return 0;

}

**再卖菜**

//2.0

#include <iostream>

#include <string>

#include <cstring>

#include <vector>

#include <stdio.h>

#include <queue>

#define MAXM 2006

using namespace std;

struct EDGE{

int u, v, w, next;

EDGE() {}

EDGE(int u, int v, int w, int next):

u(u), v(v), w(w), next(next) {}

}edge[MAXM];

int head[MAXM];

int edgeCount = 0;

int d[MAXM];

void addEdge(int u, int v, int w)

{

edge[edgeCount] = EDGE(u, v, w, head[u]);

head[u] = edgeCount++;

}

int a[301], b[301];

bool spfa(int s, int n) //s为源节点号，n为点数

{

queue <int> que;

int visit[MAXM], inq[MAXM];

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

{

que.push(i);

visit[i] = 1;

inq[i] = 0;

d[i] = 0;

}

que.push(s);

inq[s] = 1;

while (!que.empty())

{

int node = que.front();

que.pop();

EDGE edge1 = edge[head[node]]; //important head[node]

int u, v, w;

u = edge1.u;

v = edge1.v;

w = edge1.w;

inq[u] = 0;

if (visit[u]++ > n)

{

return true;

}

for (int e=head[node]; e!=-1; e=edge[e].next)

{

EDGE edge2 = edge[e];

u = edge2.u;

v = edge2.v;

w = edge2.w;

if (d[u]+w > d[v])

{

d[v] = d[u]+w;

if (!inq[v])

{

inq[v] = 1;

que.push(v);

}

}

}

}

return false;

}

int main()

{

memset((void \*)head, -1, sizeof(head));

int n;

cin >> n;

for (int i=1; i<=n; i++)

{

cin >> a[i];

}

addEdge(2, 0, -(a[1]\*2+1));

addEdge(0, 2, a[1]\*2);

addEdge(n-2, n, 2\*a[n]);

addEdge(n, n-2, -(2\*a[n]+1));

for (int i=2; i<n; i++)

{

addEdge(i-2, i+1, 3\*a[i]);

addEdge(i+1, i-2, -(a[i]\*3+2));

}

for (int i=1; i<=n; i++)

{

addEdge(i-1, i, 1);

}

spfa(0, n);

cout << d[1];

for (int i=2; i<=n; i++)

{

cout << " " << d[i]-d[i-1];

}

cout << endl;

return 0;

}

**模板**

#include <iostream>

#include <string>

#include <cstring>

#include <vector>

#include <stdio.h>

#include <queue>

#define MAXM 1000

using namespace std;

int main()

{

int n;

cin >> n;

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

{

cin >> ;

}

for (int i=; i<; i++)

{

}

cout << endl;

return 0;

}