1. 万能头文件

#include<bits/stdc++.h>

2.保留小数点后3位

#include <iomanip>

cout << setiosflags(ios::fixed) << setprecision(4) << x << endl;

3.在vector中删除某个元素

q.erase(q.begin() + t);

for (vector<int>::iterator it = q.begin(); it != q.end(); )

{

if (\*it == i)

it = q.erase(it); //不能写成arr.erase(it);

else

++it;

}

4.在vector中查找是否存在某个元素

vector<int>::iterator result = find(q.begin(), q.end(), i);

if (result != q.end())

{

\*\*\*\*\*\*

}

5.memset设置为-1

memset(vis, 0xff, sizeof(vis));

6.输出数组中最大的元素

#include <algorithm>

cout << \* max\_element(maxn, maxn + n) << endl;

7.字符串string的查询

8.结构体的定义与赋值

struct ways

{

int sou, des, day;

ways(int s, int d, int dd) : sou(s), des(d), day(dd) {};

};

9.数字int status转化为字符串char temp[10]

sprintf(temp, "%09d", status);

10.结构体内运算符重载

struct node

{

int des, depth;

friend bool operator<(const node &n, const node &n1)

{

return n.depth > n1.depth;

}

node(){}

node(int d, int de) :des(d), depth(de) {}

};

11.string的常用操作

string str;

str.find("/../") != -1;

str.rfind('/', pos);

str.erase(start, length);

str[i]=tolower(str[i]);

12.vector排序

sort(vec.begin(), vec.end(), cmp);

13.结构体比较大小

bool operator<(const trading& t1) const

{

if (t == 's')

return price > t1.price;

else

return price < t1.price;

}

14.大小写转换

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

s1 += tolower(s[i]);

15.map的遍历输出

map<char, int>::iterator it;

for (it = ori.begin(); it != ori.end(); it++)

{

cout << " -" << it->first;

if (it->second == 2)

cout << " " << st[it->first];

}

// 140303Command Line Options.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//请你写一个命令行分析程序, 用以分析给定的命令行里包含哪些选项。

//每个命令行由若干个字符串组成, 它们之间恰好由一个空格分隔。

//这些字符串中的第一个为该命令行工具的名字, 由小写字母组成, 你的程序不用对它进行处理。

//在工具名字之后可能会包含若干选项, 然后可能会包含一 些不是选项的参数。

//选项有两类 : 带参数的选项和不带参数的选项。一个合法的无参数选项的形式是一个减号后面跟单个小写字母, 如"-a" 或"-b"。

//而带参数选项则由两个由空格分隔的字符串构成, 前者的格式要求与无参数选项相同,

//后者则是该选项的参数, 是由小写字母, 数字和减号组成的非空字符串。

// 该命令行工具的作者提供给你一个格式字符串以指定他的命令行工具需要接受哪些选项。

//这个字符串由若干小写字母和冒号组成, 其中的每个小写字母表示一个该程序接受的选项。

//如果该小写字母后面紧跟了一个冒号, 它就表示一个带参数的选项, 否则则为不带参数的选项。

//例如, "ab:m:" 表示该程序接受三种选项, 即"-a"(不带参数), "-b"(带参数), 以及"-m"(带参数)。

// 命令行工具的作者准备了若干条命令行用以测试你的程序。对于每个命令行, 你的工具应当一直向后分析。

//当你的工具遇到某个字符串既不是合法的选项, 又不是某个合法选项的参数时, 分析就停止。

//命令行剩余的未分析部分不构成该命令的选项, 因此你的程序应当忽略它们。

// 输入格式

// 输入的第一行是一个格式字符串, 它至少包含一个字符, 且长度不超过 52。

//格式字符串只包含小写字母和冒号, 保证每个小写字母至多出现一次, 不会有两个相邻的冒号, 也不会以冒号开头。

// 输入的第二行是一个正整数 N(1 ≤ N ≤ 20), 表示你需要处理的命令行的个数。

// 接下来有 N 行, 每行是一个待处理的命令行, 它包括不超过 256 个字符。

//该命令行一定是若干个由单个空格分隔的字符串构成, 每个字符串里只包含小写字母, 数字和减号。

// 输出格式

// 输出有 N 行。其中第 i 行以"Case i:" 开始, 然后应当有恰好一个空格, 然后应当按照字母升序输出该命令行中用到的所有选项的名称,

//对于带参数的选项, 在输出它的名称之后还要输出它的参数。如果一个选项在命令行中出现了多次, 只输出一次。

//如果一个带参数的选项在命令行中出 现了多次, 只输出最后一次出现时所带的参数。

// 样例输入

// albw : x

// 4

// ls - a - l - a documents - b

// ls

// ls - w 10 - x - w 15

// ls - a - b - c - d - e - l

// 样例输出

// Case 1: -a - l

// Case 2 :

// Case 3 : -w 15 - x

// Case 4 : -a - b

#include "stdafx.h"

#include <iostream>

#include <cstring>

#include <algorithm>

#include <queue>

#include <string>

#include <map>

using namespace std;

string s;

int n;

map<char, int> ma, ori;

map<char, string> st;

int main()

{

cin >> s;

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

{

if (s[i] != ':')

ma[s[i]] = 1;

else

ma[s[i - 1]] = 2;

}

cin >> n;

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

{

st.clear();

ori.clear();

getchar();

string str;

getline(cin, str);

int t;

cout << "Case " << l << ":";

int j;

for (j = 0; j < str.size(); j++)

if (str[j] == ' ')

break;

if (str[j] == 0)

{

cout << endl;

continue;

}

int i = j;

while(str[i] != 0)

{

if (str[++i] != '-')

break;

if (ma[str[++i]] == 0)

break;

if (ma[str[i]] == 1)

{

if (str[i + 1] == ' ' || str[i + 1] == 0)

ori[str[i++]] = 1;

else

break;

}

else if (str[i + 1] == ' ' && str[i + 2] != 0)

{

int temp = i;

string tmp = "";

for (i = i + 2; str[i] != ' ' && str[i] != 0; i++)

tmp += str[i];

ori[str[temp]] = 2;

st[str[temp]] = tmp;

}

else

break;

}

map<char, int>::iterator it;

for (it = ori.begin(); it != ori.end(); it++)

{

cout << " -" << it->first;

if (it->second == 2)

cout << " " << st[it->first];

}

cout << endl;

}

return 0;

}

// 140903String Matching.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//给出一个字符串和多行文字，在这些文字中找到字符串出现的那些行。

//你的程序还需支持大小写敏感选项：当选项打开时，表示同一个字母的大写和小写看作不同的字符；

//当选项关闭时，表示同一个字母的大写和小写看作相同的字符。

//输入格式

//输入的第一行包含一个字符串S，由大小写英文字母组成。

//第二行包含一个数字，表示大小写敏感的选项，当数字为0时表示大小写不敏感，当数字为1时表示大小写敏感。

//第三行包含一个整数n，表示给出的文字的行数。

//接下来n行，每行包含一个字符串，字符串由大小写英文字母组成，不含空格和其他字符。

//输出格式

//输出多行，每行包含一个字符串，按出现的顺序依次给出那些包含了字符串S的行。

//样例输入

//Hello

//1

//5

//HelloWorld

//HiHiHelloHiHi

//GrepIsAGreatTool

//HELLO

//HELLOisNOTHello

//样例输出

//HelloWorld

//HiHiHelloHiHi

//HELLOisNOTHello

//样例说明

//在上面的样例中，第四个字符串虽然也是Hello，但是大小写不正确。如果将输入的第二行改为0，则第四个字符串应该输出。

//评测用例规模与约定

//1 <= n <= 100，每个字符串的长度不超过100。

#include "stdafx.h"

#include <iostream>

#include <string>

using namespace std;

string s;

string s1;

int is;

int n;

string mat[105];

string mat1[105];

int main()

{

cin >> s;

cin >> is;

if (!is)

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

s1 += tolower(s[i]);

cin >> n;

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

{

cin >> mat[i];

if (!is)

for (int j = 0; j < mat[i].size(); j++)

mat1[i] += tolower(mat[i][j]);

}

if (is)

{

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

if (mat[i].find(s) != string::npos)

cout << mat[i] << endl;

}

else

{

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

if (mat1[i].find(s1) != string::npos)

cout << mat[i] << endl;

}

return 0;

}

// 141203Call Auction.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//某股票交易所请你编写一个程序，根据开盘前客户提交的订单来确定某特定股票的开盘价和开盘成交量。

//该程序的输入由很多行构成，每一行为一条记录，记录可能有以下几种：

//1. buy p s 表示一个购买股票的买单，每手出价为p，购买股数为s。

//2. sell p s 表示一个出售股票的卖单，每手出价为p，出售股数为s。

//3. cancel i表示撤销第i行的记录。

//如果开盘价为p0，则系统可以将所有出价至少为p0的买单和所有出价至多为p0的卖单进行匹配。

//因此，此时的开盘成交量为出价至少为p0的买单的总股数和所有出价至多为p0的卖单的总股数之间的较小值。

//你的程序需要确定一个开盘价，使得开盘成交量尽可能地大。如果有多个符合条件的开盘价，你的程序应当输出最高的那一个。

//输入格式

//输入数据有任意多行，每一行是一条记录。保证输入合法。股数为不超过108的正整数，出价为精确到恰好小数点后两位的正实数，且不超过10000.00。

//输出格式

//你需要输出一行，包含两个数，以一个空格分隔。第一个数是开盘价，第二个是此开盘价下的成交量。开盘价需要精确到小数点后恰好两位。

//样例输入

//buy 9.25 100

//buy 8.88 175

//sell 9.00 1000

//buy 9.00 400

//sell 8.92 400

//cancel 1

//buy 100.00 50

//样例输出

//9.00 450

//评测用例规模与约定

//对于100%的数据，输入的行数不超过5000。

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

using namespace std;

struct trading

{

char t;

int no;

double price;

long long num;

trading(){}

trading(char tt, int n, double p, int nn) :t(tt), no(n), price(p), num(nn) {};

bool operator<(const trading& t1) const

{

if (t == 's')

return price > t1.price;

else

return price < t1.price;

}

};

priority\_queue<trading> buy;

priority\_queue<trading> sell;

bool iscancel[5005];

int main()

{

char c[10];

double p;

long long n;

memset(iscancel, false, sizeof(iscancel));

int h = 0;

while (cin >> c)

{

h++;

if (c[0] == 'c')

{

cin >> n;

iscancel[n] = true;

}

else if (c[0] == 'b')

{

cin >> p >> n;

buy.push(trading('b', h, p, n));

}

else if (c[0] == 's')

{

cin >> p >> n;

sell.push(trading('s', h, p, n));

}

else

break;

}

trading b, s, t;

t.num = 0;

while (1)

{

while (!buy.empty())

{

b = buy.top();

if (iscancel[b.no])

buy.pop();

else

break;

}

while (!sell.empty())

{

s = sell.top();

if (iscancel[s.no])

sell.pop();

else

break;

}

if (buy.empty() || sell.empty())

break;

if (b.price >= s.price)

{

t.num += min(b.num, s.num);

t.price = b.price;

if (b.num == s.num)

{

buy.pop();

sell.pop();

}

else if (b.num < s.num)

{

s.num -= b.num;

sell.pop();

buy.pop();

sell.push(s);

}

else

{

b.num -= s.num;

sell.pop();

buy.pop();

buy.push(b);

}

}

else

break;

}

cout << setiosflags(ios::fixed) << setprecision(2) << t.price << ' ';

cout << t.num << endl;

return 0;

}

// 150304Network Delay.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//给定一个公司的网络，由n台交换机和m台终端电脑组成，交换机与交换机、交换机与电脑之间使用网络连接。

//交换机按层级设置，编号为1的交换机为根交换机，层级为1。

//其他的交换机都连接到一台比自己上一层的交换机上，其层级为对应交换机的层级加1。所有的终端电脑都直接连接到交换机上。

//当信息在电脑、交换机之间传递时，每一步只能通过自己传递到自己所连接的另一台电脑或交换机。

//请问，电脑与电脑之间传递消息、或者电脑与交换机之间传递消息、或者交换机与交换机之间传递消息最多需要多少步。

//输入格式

//输入的第一行包含两个整数n, m，分别表示交换机的台数和终端电脑的台数。

//第二行包含n - 1个整数，分别表示第2、3、……、n台交换机所连接的比自己上一层的交换机的编号。

//第i台交换机所连接的上一层的交换机编号一定比自己的编号小。

//第三行包含m个整数，分别表示第1、2、……、m台终端电脑所连接的交换机的编号。

//输出格式

//输出一个整数，表示消息传递最多需要的步数。

//样例输入

//4 2

//1 1 3

//2 1

//样例输出

//4

//样例说明

//样例的网络连接模式如下，其中圆圈表示交换机，方框表示电脑：

//

//其中电脑1与交换机4之间的消息传递花费的时间最长，为4个单位时间。

//样例输入

//4 4

//1 2 2

//3 4 4 4

//样例输出

//4

//样例说明

//样例的网络连接模式如下：

//

//其中电脑1与电脑4之间的消息传递花费的时间最长，为4个单位时间。

//评测用例规模与约定

//前30%的评测用例满足：n ≤ 5, m ≤ 5。

//前50%的评测用例满足：n ≤ 20, m ≤ 20。

//前70%的评测用例满足：n ≤ 100, m ≤ 100。

//所有评测用例都满足：1 ≤ n ≤ 10000，1 ≤ m ≤ 10000。

#include "stdafx.h"

#include <iostream>

#include <algorithm>

#include <vector>

#include <queue>

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

using namespace std;

int ans = 0;

int n, m;

vector<int> vec[10001 \* 2];

int vis[10001 \* 2];

struct node

{

int des, depth;

friend bool operator<(const node &n, const node &n1)

{

return n.depth > n1.depth;

}

node(){}

node(int d, int de) :des(d), depth(de) {}

};

queue<node> q[10001 \* 2];

int d[10001 \* 2];

int bfs(int k, int &p)

{

int ans = 0;

memset(vis, 0, sizeof(vis));

memset(d, 0, sizeof(d));

q[k].push(node(k, 0));

vis[k] = 1;

d[k] = 0;

p = k;

int depth = 0;

while (!q[k].empty())

{

node t = q[k].front();

q[k].pop();

int de = t.des;

for (int i = 0; i < vec[de].size(); i++)

{

int tt = vec[de][i];

if (!vis[tt])

{

d[tt] = d[de] + 1;

if (d[tt] > ans)

{

p = tt;

ans = d[tt];

}

vis[tt] = 1;

q[k].push(node(tt, t.depth + 1));

}

}

}

return ans;

}

int main()

{

cin >> n >> m;

int i;

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

{

int a;

cin >> a;

vec[i].push\_back(a);

vec[a].push\_back(i);

}

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

{

int b;

cin >> b;

vec[i].push\_back(b);

vec[b].push\_back(i);

}

memset(vis, 0, sizeof(vis));

int p;

ans = bfs(1, p);

ans = max(ans, bfs(p, p));

cout << ans << endl;

return 0;

}

// 150403Festival.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//有一类节日的日期并不是固定的，而是以“a月的第b个星期c”的形式定下来的，比如说母亲节就定为每年的五月的第二个星期日。

//现在，给你a，b，c和y1, y2(1850 ≤ y1, y2 ≤ 2050)，希望你输出从公元y1年到公元y2年间的每年的a月的第b个星期c的日期。

//提示：关于闰年的规则：年份是400的整数倍时是闰年，否则年份是4的倍数并且不是100的倍数时是闰年，其他年份都不是闰年。

//例如1900年就不是闰年，而2000年是闰年。

//为了方便你推算，已知1850年1月1日是星期二。

//输入格式

//输入包含恰好一行，有五个整数a, b, c, y1, y2。其中c = 1, 2, ……, 6, 7分别表示星期一、二、……、六、日。

//输出格式

//对于y1和y2之间的每一个年份，包括y1和y2，按照年份从小到大的顺序输出一行。

//如果该年的a月第b个星期c确实存在，则以"yyyy/mm/dd"的格式输出，即输出四位数的年份，两位数的月份，两位数的日期，

//中间用斜杠“ / ”分隔，位数不足时前补零。

//如果该年的a月第b个星期c并不存在，则输出"none"（不包含双引号)。

//样例输入

//5 2 7 2014 2015

//样例输出

//2014 / 05 / 11

//2015 / 05 / 10

//评测用例规模与约定

//所有评测用例都满足：1 ≤ a ≤ 12，1 ≤ b ≤ 5，1 ≤ c ≤ 7，1850 ≤ y1, y2 ≤ 2050。

#include "stdafx.h"

#include <iostream>

using namespace std;

int a, b, c, yy, y2;

int month[] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

bool isleapyear(int y)

{

if (y % 400 == 0 || (y % 4 == 0 && y % 100 != 0))

return true;

return false;

}

int weekday(int y, int m, int d)

{

int days = 0;

for (int i = 1850; i < y; i++)

if (isleapyear(i))

days += 366;

else

days += 365;

if (isleapyear(y))

month[2] = 29;

else

month[2] = 28;

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

days += month[i];

days += d;

int t = (days % 7) + 1;

return t;

}

int main()

{

cin >> a >> b >> c >> yy >> y2;

weekday(1851, 1, 1);

for (int i = yy; i <= y2; i++)

{

if (isleapyear(i))

month[2] = 29;

else

month[2] = 28;

int count = 0;

int j;

for (j = 1; j <= month[a]; j++)

{

if (weekday(i, a, j) == c)

count++;

if (count == b)

break;

}

if (j == month[a] + 1)

cout << "none" << endl;

else

{

cout << i << '/';

if (a < 10)

cout << '0' << a << '/';

else

cout << a << '/';

if (j < 10)

cout << '0' << j << endl;

else

cout << j << endl;

}

}

return 0;

}

// 150903Template Generation System.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//成成最近在搭建一个网站，其中一些页面的部分内容来自数据库中不同的数据记录，但是页面的基本结构是相同的。

//例如，对于展示用户信息的页面，当用户为 Tom 时，网页的源代码是

//

//

//而当用户为 Jerry 时，网页的源代码是

//

//

//这样的例子在包含动态内容的网站中还有很多。为了简化生成网页的工作，成成觉得他需要引入一套模板生成系统。

//模板是包含特殊标记的文本。成成用到的模板只包含一种特殊标记，格式为{ { VAR } }，其中 VAR 是一个变量。

//该标记在模板生成时会被变量 VAR 的值所替代。例如，如果变量 name = "Tom"，则{ { name } } 会生成 Tom。具体的规则如下：

//·变量名由大小写字母、数字和下划线(\_) 构成，且第一个字符不是数字，长度不超过 16 个字符。

//·变量名是大小写敏感的，Name 和 name 是两个不同的变量。

//·变量的值是字符串。

//·如果标记中的变量没有定义，则生成空串，相当于把标记从模板中删除。

//·模板不递归生成。也就是说，如果变量的值中包含形如{ { VAR } } 的内容，不再做进一步的替换。

//输入格式

//输入的第一行包含两个整数 m, n，分别表示模板的行数和模板生成时给出的变量个数。

//接下来 m 行，每行是一个字符串，表示模板。

//接下来 n 行，每行表示一个变量和它的值，中间用一个空格分隔。

//值是字符串，用双引号(") 括起来，内容可包含除双引号以外的任意可打印 ASCII 字符（ASCII 码范围 32, 33, 35-126）。

// 输出格式

// 输出包含若干行，表示模板生成的结果。

// 样例输入

// 11 2

// <!DOCTYPE html>

// <html>

// <head>

// <title>User{ { name } }< / title>

// < / head>

// <body>

// <h1>{ { name }}< / h1>

// <p>Email: <a href = "mailto:{{ email }}">{ { email } }< / a>< / p>

// <p>Address : { { address }}< / p>

// < / body>

// < / html>

// name "David Beckham"

// email "david@beckham.com"

// 样例输出

// <!DOCTYPE html>

// <html>

// <head>

// <title>User David Beckham< / title>

// < / head>

// <body>

// <h1>David Beckham< / h1>

// <p>Email: <a href = "mailto:david@beckham.com">david@beckham.com< / a>< / p>

// <p>Address: < / p>

// < / body>

// < / html>

// 评测用例规模与约定

// 0 ≤ m ≤ 100

// 0 ≤ n ≤ 100

// 输入的模板每行长度不超过 80 个字符（不包含换行符）。

// 输入保证模板中所有以{ { 开始的子串都是合法的标记，开始是两个左大括号和一个空格，然后是变量名，结尾是一个空格和两个右大括号。

// 输入中所有变量的值字符串长度不超过 100 个字符（不包括双引号）。

// 保证输入的所有变量的名字各不相同。

#include "stdafx.h"

#include <iostream>

#include <string>

#include <vector>

#include <map>

using namespace std;

int m, n;

string str;

map<string, string> ma;

int main()

{

cin >> m >> n;

getchar();

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

{

string s;

getline(cin, s);

str += s + "\n";

}

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

{

string s;

getline(cin, s);

int t = s.find('\"');

string a = s.substr(0, t - 1);

string b = s.substr(t + 1, s.size() - t - 2);

ma[a] = b;

}

string tmp = str;

int k, l;

int len = str.size();

for (int j = 0; j < len; j = l + 2)

{

string t;

k = str.find("{{");

l = str.find("}}");

if (k == string::npos || l == string::npos)

break;

t = str.substr(k + 3, l - k - 4);

//str = str.substr(0, k) + ma[t] + str.substr(l + 2, str.size() - l - 2);

str.replace(k, l - k + 2, ma[t]);

len = str.size();

}

cout << str;

return 0;

}

// 150904Expressway.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//某国有n个城市，为了使得城市间的交通更便利，该国国王打算在城市之间修一些高速公路，

//由于经费限制，国王打算第一阶段先在部分城市之间修一些单向的高速公路。

//现在，大臣们帮国王拟了一个修高速公路的计划。

//看了计划后，国王发现，有些城市之间可以通过高速公路直接（不经过其他城市）或间接（经过一个或多个其他城市）到达，而有的却不能。

//如果城市A可以通过高速公路到达城市B，而且城市B也可以通过高速公路到达城市A，则这两个城市被称为便利城市对。

//国王想知道，在大臣们给他的计划中，有多少个便利城市对。

//输入格式

//输入的第一行包含两个整数n, m，分别表示城市和单向高速公路的数量。

//接下来m行，每行两个整数a, b，表示城市a有一条单向的高速公路连向城市b。

//输出格式

//输出一行，包含一个整数，表示便利城市对的数量。

//样例输入

//5 5

//1 2

//2 3

//3 4

//4 2

//3 5

//样例输出

//3

//样例说明

//

//城市间的连接如图所示。有3个便利城市对，它们分别是(2, 3), (2, 4), (3, 4)，请注意(2, 3)和(3, 2)看成同一个便利城市对。

//评测用例规模与约定

//前30%的评测用例满足1 ≤ n ≤ 100, 1 ≤ m ≤ 1000；

//前60%的评测用例满足1 ≤ n ≤ 1000, 1 ≤ m ≤ 10000；

//所有评测用例满足1 ≤ n ≤ 10000, 1 ≤ m ≤ 100000。

#include "stdafx.h"

#include <iostream>

#include <vector>

#include <queue>

#include <stack>

#include <cstring>

#include <algorithm>

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

using namespace std;

const int N = 10005;

vector<int> G[N];

int vis[N];

int time;

int LOW[N], DFN[N];

stack<int> s;

int istack[N];

int result = 0;

void dfs(int k)

{

LOW[k] = DFN[k] = time++;

vis[k] = 1;

s.push(k);

istack[k] = 1;

for (int i = 0; i < G[k].size(); i++)

{

int t = G[k][i];

if (!vis[t])

{

dfs(t);

LOW[k] = min(LOW[t], LOW[k]);

}

else

LOW[k] = min(LOW[t], LOW[k]);

}

int cnt = 0;

int m;

if (LOW[k] == DFN[k])

{

do

{

m = s.top();

s.pop();

istack[m] = 0;

cnt++;

} while (m != k);

if (cnt > 1)

result += (cnt - 1) \* cnt / 2;

}

}

int main()

{

int n, m, a, b;

cin >> n >> m;

while (m--)

{

cin >> a >> b;

G[a].push\_back(b);

}

memset(vis, 0, sizeof(vis));

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

{

time = 1;

if (!vis[i])

{

dfs(i);

}

}

cout << result << endl;

return 0;

}

// 151203Paint.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//用 ASCII 字符来画图是一件有趣的事情，并形成了一门被称为 ASCII Art 的艺术。例如，下图是用 ASCII 字符画出来的 CSPRO 字样。

//..\_\_\_\_.\_\_\_\_..\_\_\_\_..\_\_\_\_...\_\_\_..

//. / .\_\_\_ / .\_\_\_ || ..\_.\ | ..\_.\. / .\_.\.

//| . | ...\\_\_\_.\ | . | \_). | . | \_). | . | . | .|

//|. | \_\_\_.\_\_\_). | ..\_\_/|..\_.< | . | \_ | . |

// .\\_\_\_\_ | \_\_\_\_/|\_ | ... | \_ | .\\_\\\_\_\_ / .

// 本题要求编程实现一个用 ASCII 字符来画图的程序，支持以下两种操作：

//  画线：给出两个端点的坐标，画一条连接这两个端点的线段。简便起见题目保证要画的每条线段都是水平或者竖直的。

//水平线段用字符 - 来画，竖直线段用字符 | 来画。如果一条水平线段和一条竖直线段在某个位置相交，则相交位置用字符 + 代替。

//  填充：给出填充的起始位置坐标和需要填充的字符，从起始位置开始，用该字符填充相邻位置，直到遇到画布边缘或已经画好的线段。

//注意这里的相邻位置只需要考虑上下左右 4 个方向，如下图所示，字符 @ 只和 4 个字符 \* 相邻。

// .\*.

// \*@\*

// .\*.

// 输入格式

// 第1行有三个整数m, n和q。m和n分别表示画布的宽度和高度，以字符为单位。q表示画图操作的个数。

// 第2行至第q + 1行，每行是以下两种形式之一：

//  0 x1 y1 x2 y2：表示画线段的操作，(x1, y1)和(x2, y2)分别是线段的两端，满足要么x1 = x2 且y1 ≠ y2，要么 y1 = y2 且 x1 ≠ x2。

//  1 x y c：表示填充操作，(x, y)是起始位置，保证不会落在任何已有的线段上；c 为填充字符，是大小写字母。

// 画布的左下角是坐标为(0, 0) 的位置，向右为x坐标增大的方向，向上为y坐标增大的方向。这q个操作按照数据给出的顺序依次执行。

//画布最初时所有位置都是字符.（小数点）。

// 输出格式

// 输出有n行，每行m个字符，表示依次执行这q个操作后得到的画图结果。

// 样例输入

// 4 2 3

// 1 0 0 B

// 0 1 0 2 0

// 1 0 0 A

// 样例输出

// AAAA

// A--A

// 样例输入

// 16 13 9

// 0 3 1 12 1

// 0 12 1 12 3

// 0 12 3 6 3

// 0 6 3 6 9

// 0 6 9 12 9

// 0 12 9 12 11

// 0 12 11 3 11

// 0 3 11 3 1

// 1 4 2 C

// 样例输出

// ................

// ... + -------- + ...

// ... | CCCCCCCC | ...

// ... | CC + ---- - +...

// ... | CC | .........

// ... | CC | .........

// ... | CC | .........

// ... | CC | .........

// ... | CC | .........

// ... | CC + ---- - +...

// ... | CCCCCCCC | ...

// ... + -------- + ...

// ................

// 评测用例规模与约定

// 所有的评测用例满足：2 ≤ m, n ≤ 100，0 ≤ q ≤ 100，0 ≤ x < m（x表示输入数据中所有位置的x坐标），

//0 ≤ y < n（y表示输入数据中所有位置的y坐标）。

//

#include "stdafx.h"

#include <iostream>

#include <algorithm>

#include <cstring>

using namespace std;

int n, m, t;

char p[101][101];

int ff, x1, x2, yy, y2;

char f;

int maxx = 0, maxy = 0;

int vis[101][101];

int posx[4] = { -1, 0, 0, 1 };

int posy[4] = { 0, -1, 1, 0 };

int ispaint[101][101];

void dfs(int x, int y)

{

ispaint[y][x] = 1;

vis[y][x] = 1;

for (int i = 0; i < 4; i++)

{

int px = x + posx[i];

int py = y + posy[i];

if (px < 0 || py < 0 || px >= m || py >= n)

continue;

if (vis[py][px])

continue;

if (p[py][px] == '|' || p[py][px] == '+' || p[py][px] == '-')

continue;

dfs(px, py);

}

}

int main()

{

cin >> m >> n >> t;

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

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

p[i][j] = '.';

while (t--)

{

cin >> ff;

if (ff == 0)

{

cin >> x1 >> yy >> x2 >> y2;

if (x1 == x2)

{

for (int i = min(yy, y2); i <= max(yy, y2); i++)

if (p[i][x1] == '-' || p[i][x1] == '+')

p[i][x1] = '+';

else

p[i][x1] = '|';

}

else if (yy == y2)

{

for (int i = min(x1, x2); i <= max(x1, x2); i++)

if (p[yy][i] == '|' || p[yy][i] == '+')

p[yy][i] = '+';

else

p[yy][i] = '-';

}

}

else if (ff == 1)

{

cin >> x1 >> yy;

cin >> f;

memset(vis, 0, sizeof(vis));

memset(ispaint, 0, sizeof(ispaint));

dfs(x1, yy);

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

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

{

if (ispaint[i][j])

p[i][j] = f;

}

}

}

for (int i = n - 1; i >= 0; i--)

{

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

cout << p[i][j];

cout << endl;

}

return 0;

}

// 151204Deliver Goods.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//为了增加公司收入，F公司新开设了物流业务。

//由于F公司在业界的良好口碑，物流业务一开通即受到了消费者的欢迎，物流业务马上遍及了城市的每条街道。

//然而，F公司现在只安排了小明一个人负责所有街道的服务。

//任务虽然繁重，但是小明有足够的信心，他拿到了城市的地图，准备研究最好的方案。

//城市中有n个交叉路口，m条街道连接在这些交叉路口之间，每条街道的首尾都正好连接着一个交叉路口。

//除开街道的首尾端点，街道不会在其他位置与其他街道相交。每个交叉路口都至少连接着一条街道，有的交叉路口可能只连接着一条或两条街道。

//小明希望设计一个方案，

//从编号为1的交叉路口出发，每次必须沿街道去往街道另一端的路口，再从新的路口出发去往下一个路口，直到所有的街道都经过了正好一次。

//输入格式

//输入的第一行包含两个整数n, m，表示交叉路口的数量和街道的数量，交叉路口从1到n标号。

//接下来m行，每行两个整数a, b，表示和标号为a的交叉路口和标号为b的交叉路口之间有一条街道，街道是双向的，小明可以从任意一端走向另一端。

//两个路口之间最多有一条街道。

//输出格式

//如果小明可以经过每条街道正好一次，则输出一行包含m + 1个整数p1, p2, p3, ..., pm + 1，表示小明经过的路口的顺序，

//相邻两个整数之间用一个空格分隔。

//如果有多种方案满足条件，则输出字典序最小的一种方案，即首先保证p1最小，p1最小的前提下再保证p2最小，依此类推。

//如果不存在方案使得小明经过每条街道正好一次，则输出一个整数 - 1。

//样例输入

//4 5

//1 2

//1 3

//1 4

//2 4

//3 4

//样例输出

//1 2 4 1 3 4

//样例说明

//城市的地图和小明的路径如下图所示。

//

//样例输入

//4 6

//1 2

//1 3

//1 4

//2 4

//3 4

//2 3

//样例输出

//- 1

//样例说明

//城市的地图如下图所示，不存在满足条件的路径。

//

//评测用例规模与约定

//前30%的评测用例满足：1 ≤ n ≤ 10, n - 1 ≤ m ≤ 20。

//前50%的评测用例满足：1 ≤ n ≤ 100, n - 1 ≤ m ≤ 10000。

//所有评测用例满足：1 ≤ n ≤ 10000，n - 1 ≤ m ≤ 100000。

#include "stdafx.h"

#include <iostream>

#include <cstring>

#include <queue>

#include <set>

#include <algorithm>

#include <stack>

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

using namespace std;

int n, m;

int a, b;

int isflag;

set<int> se[10001];

bool vis[10001][10001];

stack<int> st;

int v[10001];

class UF

{

private:

int length;

public:

UF(int n)

{

length = n;

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

v[i] = i;

}

int Find(int x)

{

if (x == v[x])

return x;

else

return v[x] = Find(v[x]);

}

bool Union(int x, int y)

{

int f1 = Find(x);

int f2 = Find(y);

if (f1 != f2)

{

v[f1] = f2;

return true;

}

else

return false;

}

};

void dfs(int k)

{

for (set<int>::iterator it = se[k].begin(); it != se[k].end(); it++)

{

if (!vis[k][\*it])

{

vis[k][\*it] = true;

vis[\*it][k] = true;

dfs(\*it);

}

}

st.push(k);

}

int main()

{

cin >> n >> m;

UF uf(n);

while (m--)

{

cin >> a >> b;

se[a].insert(b);

se[b].insert(a);

uf.Union(a, b);

}

isflag = false;

int root = uf.Find(1);

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

if (uf.Find(i) != root)

{

isflag = true;

break;

}

if (!isflag)

{

int count = 0;

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

if (se[i].size() % 2 == 1)

count++;

if (count == 0 || (count == 2 && se[1].size() == 1))

isflag = false;

else

isflag = true;

}

if (!isflag)

{

memset(vis, false, sizeof(vis));

dfs(1);

int t = st.top();

while (!st.empty())

{

st.pop();

cout << t << ' ';

}

cout << endl;

}

else

cout << -1 << endl;

return 0;

}

// 160404Game.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//小明在玩一个电脑游戏，游戏在一个n×m的方格图上进行，小明控制的角色开始的时候站在第一行第一列，目标是前往第n行第m列。

//方格图上有一些方格是始终安全的，有一些在一段时间是危险的，

//如果小明控制的角色到达一个方格的时候方格是危险的，则小明输掉了游戏，如果小明的角色到达了第n行第m列，则小明过关。

//第一行第一列和第n行第m列永远都是安全的。

//每个单位时间，小明的角色必须向上下左右四个方向相邻的方格中的一个移动一格。

//经过很多次尝试，小明掌握了方格图的安全和危险的规律：每一个方格出现危险的时间一定是连续的。

//并且，小明还掌握了每个方格在哪段时间是危险的。

//现在，小明想知道，自己最快经过几个时间单位可以达到第n行第m列过关。

//输入格式

//输入的第一行包含三个整数n, m, t，用一个空格分隔，表示方格图的行数n、列数m，以及方格图中有危险的方格数量。

//接下来t行，每行4个整数r, c, a, b，表示第r行第c列的方格在第a个时刻到第b个时刻之间是危险的，包括a和b。

//游戏开始时的时刻为0。输入数据保证r和c不同时为1，而且当r为n时c不为m。一个方格只有一段时间是危险的（或者说不会出现两行拥有相同的r和c）。

//输出格式

//输出一个整数，表示小明最快经过几个时间单位可以过关。输入数据保证小明一定可以过关。

//样例输入

//3 3 3

//2 1 1 1

//1 3 2 10

//2 2 2 10

//样例输出

//6

//样例说明

//第2行第1列时刻1是危险的，因此第一步必须走到第1行第2列。

//第二步可以走到第1行第1列，第三步走到第2行第1列，后面经过第3行第1列、第3行第2列到达第3行第3列。

//评测用例规模与约定

//前30%的评测用例满足：0 < n, m ≤ 10，0 ≤ t < 99。

// 所有评测用例满足：0 < n, m ≤ 100，0 ≤ t < 9999，1 ≤ r ≤ n，1 ≤ c ≤ m，0 ≤ a ≤ b ≤ 100。

#include "stdafx.h"

#include <iostream>

#include <algorithm>

#include <queue>

#include <cstring>

#include <set>

#include <vector>

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

using namespace std;

struct vex

{

int a, b, t;

vex(){}

vex(int aa, int bb, int tt) :a(aa), b(bb), t(tt) {};

};

struct node

{

int a, b;

node() {}

node(int aa, int bb) :a(aa), b(bb) {};

};

int n, m, t;

int r, c, a, b;

node danger[110][110];

int posx[4] = { -1, 0, 0, 1 };

int posy[4] = { 0, -1, 1, 0 };

bool flag[110][110][310];

void bfs()

{

queue<vex> q;

q.push(vex(1, 1, 0));

flag[1][1][0] = 1;

while (!q.empty())

{

vex t = q.front();

q.pop();

for (int i = 0; i < 4; i++)

{

int hang = t.a + posx[i];

int lie = t.b + posy[i];

if (hang == n && lie == m)

{

cout << t.t + 1<< endl;

return;

}

if (hang <= 0 || hang > n || lie <= 0 || lie > m)

continue;

if (t.t <= 100 && (t.t + 1 >= danger[hang][lie].a && t.t + 1 <= danger[hang][lie].b))

continue;

if (flag[hang][lie][t.t + 1])

continue;

flag[hang][lie][t.t + 1] = 1;

q.push(vex(hang, lie, t.t + 1));

}

}

}

int main()

{

cin >> n >> m >> t;

memset(flag, 0, sizeof(flag));

while (t--)

{

cin >> r >> c >> a >> b;

danger[r][c].a = a;

danger[r][c].b = b;

}

bfs();

return 0;

}

// 160403Path Analysis.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//在操作系统中，数据通常以文件的形式存储在文件系统中。

//文件系统一般采用层次化的组织形式，由目录（或者文件夹）和文件构成，形成一棵树的形状。

//文件有内容，用于存储数据。目录是容器，可包含文件或其他目录。

//同一个目录下的所有文件和目录的名字各不相同，不同目录下可以有名字相同的文件或目录。

//为了指定文件系统中的某个文件，需要用路径来定位。

//在类 Unix 系统（Linux、Max OS X、FreeBSD等）中，路径由若干部分构成，每个部分是一个目录或者文件的名字，相邻两个部分之间用 / 符号分隔。

//有一个特殊的目录被称为根目录，是整个文件系统形成的这棵树的根节点，用一个单独的 / 符号表示。

//在操作系统中，有当前目录的概念，表示用户目前正在工作的目录。根据出发点可以把路径分为两类：

// 绝对路径：以 / 符号开头，表示从根目录开始构建的路径。

// 相对路径：不以 / 符号开头，表示从当前目录开始构建的路径。

//

//例如，有一个文件系统的结构如下图所示。在这个文件系统中，有根目录 / 和其他普通目录 d1、d2、d3、d4，以及文件 f1、f2、f3、f1、f4。

//其中，两个 f1 是同名文件，但在不同的目录下。

//

//对于 d4 目录下的 f1 文件，可以用绝对路径 / d2 / d4 / f1 来指定。

//如果当前目录是 / d2 / d3，这个文件也可以用相对路径 .. / d4 / f1 来指定，这里 ..表示上一级目录（注意，根目录的上一级目录是它本身）。

//还有.表示本目录，例如 / d1 / . / f1 指定的就是 / d1 / f1。

//注意，如果有多个连续的 / 出现，其效果等同于一个 / ，例如 / d1///f1 指定的也是 /d1/f1。

//本题会给出一些路径，要求对于每个路径，给出正规化以后的形式。

//一个路径经过正规化操作后，其指定的文件不变，但是会变成一个不包含.和 ..的绝对路径，且不包含连续多个 / 符号。

//如果一个路径以 / 结尾，那么它代表的一定是一个目录，正规化操作要去掉结尾的 / 。若这个路径代表根目录，则正规化操作的结果是 / 。

//若路径为空字符串，则正规化操作的结果是当前目录。

//输入格式

//第一行包含一个整数 P，表示需要进行正规化操作的路径个数。

//第二行包含一个字符串，表示当前目录。

//以下 P 行，每行包含一个字符串，表示需要进行正规化操作的路径。

//输出格式

//共 P 行，每行一个字符串，表示经过正规化操作后的路径，顺序与输入对应。

//样例输入

//7

/// d2 / d3

/// d2 / d4 / f1

//.. / d4 / f1

/// d1 / . / f1

/// d1///f1

/// d1/

/////

///d1 / .. / .. / d2

//样例输出

/// d2 / d4 / f1

/// d2 / d4 / f1

/// d1 / f1

/// d1 / f1

/// d1

///

///d2

//评测用例规模与约定

//1 ≤ P ≤ 10。

//文件和目录的名字只包含大小写字母、数字和小数点.、减号 - 以及下划线 \_。

//不会有文件或目录的名字是.或 ..，它们具有题目描述中给出的特殊含义。

//输入的所有路径每个长度不超过 1000 个字符。

//输入的当前目录保证是一个经过正规化操作后的路径。

//对于前 30 % 的测试用例，需要正规化的路径的组成部分不包含.和 ..。

//对于前 60 % 的测试用例，需要正规化的路径都是绝对路径。

#include "stdafx.h"

#include <iostream>

#include <string>

using namespace std;

int main()

{

int P;

cin >> P;

string cur;

cin >> cur;

getchar();

int pos;

int p;

while (P--)

{

string str;

getline(cin, str);

if (str.size() == 0)

str = cur;

else if (str[0] != '/')

str = cur + "/" + str;

while ((pos = str.find("/../")) != -1)

{

if (pos == 0)

str.erase(0, 3);

else

{

p = str.rfind('/', pos - 1);

str.erase(p, pos - p + 3);

}

}

while ((pos = str.find("/./")) != -1)

str.erase(pos, 2);

while ((pos = str.find("//")) != -1)

str.erase(pos, 1);

if (str.size() > 1 && str[str.size() - 1] == '/')

str.erase(str.size() - 1, 1);

cout << str << endl;

}

return 0;

}

// 160901Maximum Fluctuation.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//小明正在利用股票的波动程度来研究股票。

//小明拿到了一只股票每天收盘时的价格，他想知道，这只股票连续几天的最大波动值是多少，

//即在这几天中某天收盘价格与前一天收盘价格之差的绝对值最大是多少。

//输入格式

//输入的第一行包含了一个整数n，表示小明拿到的收盘价格的连续天数。

//第二行包含n个正整数，依次表示每天的收盘价格。

//输出格式

//输出一个整数，表示这只股票这n天中的最大波动值。

//样例输入

//6

//2 5 5 7 3 5

//样例输出

//4

//样例说明

//第四天和第五天之间的波动最大，波动值为 | 3 - 7 |= 4。

//评测用例规模与约定

//对于所有评测用例，2 ≤ n ≤ 1000。股票每一天的价格为1到10000之间的整数。

#include "stdafx.h"

#include <iostream>

#include <algorithm>

#include <cmath>

using namespace std;

int n;

int a[1010];

int ans = 0;

int main()

{

cin >> n;

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

cin >> a[i];

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

ans = max(ans, abs(a[i] - a[i - 1]));

cout << ans << endl;

return 0;

}

// 160902Train Ticket Purchase.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//请实现一个铁路购票系统的简单座位分配算法，来处理一节车厢的座位分配。

//假设一节车厢有20排、每一排5个座位。为方便起见，我们用1到100来给所有的座位编号，

//第一排是1到5号，第二排是6到10号，依次类推，第20排是96到100号。

//购票时，一个人可能购一张或多张票，最多不超过5张。

//如果这几张票可以安排在同一排编号相邻的座位，则应该安排在编号最小的相邻座位。否则应该安排在编号最小的几个空座位中（不考虑是否相邻）。

//假设初始时车票全部未被购买，现在给了一些购票指令，请你处理这些指令。

//输入格式

//输入的第一行包含一个整数n，表示购票指令的数量。

//第二行包含n个整数，每个整数p在1到5之间，表示要购入的票数，相邻的两个数之间使用一个空格分隔。

//输出格式

//输出n行，每行对应一条指令的处理结果。

//对于购票指令p，输出p张车票的编号，按从小到大排序。

//样例输入

//4

//2 5 4 2

//样例输出

//1 2

//6 7 8 9 10

//11 12 13 14

//3 4

//样例说明

//1) 购2张票，得到座位1、2。

//2) 购5张票，得到座位6至10。

//3) 购4张票，得到座位11至14。

//4) 购2张票，得到座位3、4。

//评测用例规模与约定

//对于所有评测用例，1 ≤ n ≤ 100，所有购票数量之和不超过100。

#include "stdafx.h"

#include <iostream>

#include <cstring>

using namespace std;

int n;

int vis[20][5];

int blank[20];

int p;

int main()

{

cin >> n;

for (int i = 0; i < 20; i++)

blank[i] = 5;

memset(vis, 0, sizeof(vis));

bool flag = false;

while (n--)

{

cin >> p;

int i;

for (i = 0; i < 20; i++)

if (blank[i] >= p)

{

int t = 0;

for (int j = 0; j < 5; j++)

{

if (!vis[i][j] && t < p)

{

t++;

vis[i][j] = 1;

cout << i \* 5 + j + 1 << " ";

}

}

blank[i] -= p;

break;

}

if (i == 20)

{

for (int i = 0; i < 20; i++)

{

int t = 0;

int per;

for (int j = 0; j < 5; j++)

{

per = 0;

if (!vis[i][j] && t < p)

{

t++;

per++;

vis[i][j] = 1;

cout << i \* 5 + j + 1 << " ";

}

}

blank[i] -= per;

}

}

cout << endl;

}

return 0;

}

// 160903Heart Stone.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//《炉石传说：魔兽英雄传》（Hearthstone : Heroes of Warcraft，简称炉石传说）是暴雪娱乐开发的一款集换式卡牌游戏（如下图所示）。

//游戏在一个战斗棋盘上进行，由两名玩家轮流进行操作，本题所使用的炉石传说游戏的简化规则如下：

//

// \* 玩家会控制一些角色，每个角色有自己的生命值和攻击力。当生命值小于等于 0 时，该角色死亡。角色分为英雄和随从。

// \* 玩家各控制一个英雄，游戏开始时，英雄的生命值为 30，攻击力为 0。当英雄死亡时，游戏结束，英雄未死亡的一方获胜。

// \* 玩家可在游戏过程中召唤随从。

//棋盘上每方都有 7 个可用于放置随从的空位，从左到右一字排开，被称为战场。当随从死亡时，它将被从战场上移除。

// \* 游戏开始后，两位玩家轮流进行操作，每个玩家的连续一组操作称为一个回合。

// \* 每个回合中，当前玩家可进行零个或者多个以下操作：

// 1) 召唤随从：玩家召唤一个随从进入战场，随从具有指定的生命值和攻击力。

// 2) 随从攻击：玩家控制自己的某个随从攻击对手的英雄或者某个随从。

// 3) 结束回合：玩家声明自己的当前回合结束，游戏将进入对手的回合。该操作一定是一个回合的最后一个操作。

// \* 当随从攻击时，攻击方和被攻击方会同时对彼此造成等同于自己攻击力的伤害。

//受到伤害的角色的生命值将会减少，数值等同于受到的伤害。

//例如，随从 X 的生命值为 HX、攻击力为 AX，随从 Y 的生命值为 HY、攻击力为 AY，

//如果随从 X 攻击随从 Y，则攻击发生后随从 X 的生命值变为 HX - AY，随从 Y 的生命值变为 HY - AX。攻击发生后，角色的生命值可以为负数。

// 本题将给出一个游戏的过程，要求编写程序模拟该游戏过程并输出最后的局面。

// 输入格式

// 输入第一行是一个整数 n，表示操作的个数。接下来 n 行，每行描述一个操作，格式如下：

// <action> <arg1> <arg2> ...

// 其中<action>表示操作类型，是一个字符串，共有 3 种：summon表示召唤随从，attack表示随从攻击，end表示结束回合。

//这 3 种操作的具体格式如下：

// \* summon <position> <attack> <health>：当前玩家在位置<position>召唤一个生命值为<health>、攻击力为<attack>的随从。

//其中<position>是一个 1 到 7 的整数，表示召唤的随从出现在战场上的位置，原来该位置及右边的随从都将顺次向右移动一位。

// \* attack <attacker> <defender>：当前玩家的角色<attacker>攻击对方的角色 <defender>。

//<attacker>是 1 到 7 的整数，表示发起攻击的本方随从编号，<defender>是 0 到 7 的整数，表示被攻击的对方角色，

//0 表示攻击对方英雄，1 到 7 表示攻击对方随从的编号。

// \* end：当前玩家结束本回合。

// 注意：随从的编号会随着游戏的进程发生变化，当召唤一个随从时，玩家指定召唤该随从放入战场的位置，

//此时，原来该位置及右边的所有随从编号都会增加 1。而当一个随从死亡时，它右边的所有随从编号都会减少 1。

//任意时刻，战场上的随从总是从1开始连续编号。

// 输出格式

// 输出共 5 行。

// 第 1 行包含一个整数，表示这 n 次操作后（以下称为 T 时刻）游戏的胜负结果，

//1 表示先手玩家获胜， - 1 表示后手玩家获胜，0 表示游戏尚未结束，还没有人获胜。

// 第 2 行包含一个整数，表示 T 时刻先手玩家的英雄的生命值。

// 第 3 行包含若干个整数，第一个整数 p 表示 T 时刻先手玩家在战场上存活的随从个数，

//之后 p 个整数，分别表示这些随从在 T 时刻的生命值（按照从左往右的顺序）。

// 第 4 行和第 5 行与第 2 行和第 3 行类似，只是将玩家从先手玩家换为后手玩家。

// 样例输入

// 8

// summon 1 3 6

// summon 2 4 2

// end

// summon 1 4 5

// summon 1 2 1

// attack 1 2

// end

// attack 1 1

// 样例输出

// 0

// 30

// 1 2

// 30

// 1 2

// 样例说明

// 按照样例输入从第 2 行开始逐行的解释如下：

// 1. 先手玩家在位置 1 召唤一个生命值为 6、攻击力为 3 的随从 A，是本方战场上唯一的随从。

// 2. 先手玩家在位置 2 召唤一个生命值为 2、攻击力为 4 的随从 B，出现在随从 A 的右边。

// 3. 先手玩家回合结束。

// 4. 后手玩家在位置 1 召唤一个生命值为 5、攻击力为 4 的随从 C，是本方战场上唯一的随从。

// 5. 后手玩家在位置 1 召唤一个生命值为 1、攻击力为 2 的随从 D，出现在随从 C 的左边。

// 6. 随从 D 攻击随从 B，双方均死亡。

// 7. 后手玩家回合结束。

// 8. 随从 A 攻击随从 C，双方的生命值都降低至 2。

// 评测用例规模与约定

// \* 操作的个数0 ≤ n ≤ 1000。

// \* 随从的初始生命值为 1 到 100 的整数，攻击力为 0 到 100 的整数。

// \* 保证所有操作均合法，包括但不限于：

// 1) 召唤随从的位置一定是合法的，即如果当前本方战场上有 m 个随从，则召唤随从的位置一定在 1 到 m + 1 之间，

//其中 1 表示战场最左边的位置，m + 1 表示战场最右边的位置。

// 2) 当本方战场有 7 个随从时，不会再召唤新的随从。

// 3) 发起攻击和被攻击的角色一定存在，发起攻击的角色攻击力大于 0。

// 4) 一方英雄如果死亡，就不再会有后续操作。

// \* 数据约定：

// 前 20 % 的评测用例召唤随从的位置都是战场的最右边。

// 前 40 % 的评测用例没有 attack 操作。

// 前 60 % 的评测用例不会出现随从死亡的情况。

#include "stdafx.h"

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct fellow

{

int health;

int attack;

fellow(){}

fellow(int h, int a) :health(h), attack(a) {};

};

int m;

vector<fellow> p[2];

int main()

{

cin >> m;

int i = 0;

p[0].push\_back(fellow(30, 0));

p[1].push\_back(fellow(30, 0));

while (m--)

{

char s[10];

cin >> s;

int t = i % 2;

if (s[0] == 's')

{

int a, b, c;

cin >> a >> b >> c;

p[t].insert(p[t].begin() + a, fellow(c, b));

}

else if (s[0] == 'e')

{

i++;

continue;

}

else if (s[0] == 'a')

{

int a, b;

cin >> a >> b;

if (t == 0)

{

p[0][a].health -= p[1][b].attack;

p[1][b].health -= p[0][a].attack;

if (p[0][a].health <= 0 && a != 0)

p[0].erase(p[0].begin() + a);

if (p[1][b].health <= 0 && b != 0)

p[1].erase(p[1].begin() + b);

}

else if (t == 1)

{

p[1][a].health -= p[0][b].attack;

p[0][b].health -= p[1][a].attack;

if (p[1][a].health <= 0 && a != 0)

p[1].erase(p[1].begin() + a);

if (p[0][b].health <= 0 && b != 0)

p[0].erase(p[0].begin() + b);

}

if (p[0][0].health <= 0 || p[1][0].health <= 0)

break;

}

}

if (p[0][0].health > 0 && p[1][0].health > 0)

cout << 0 << endl;

else if (p[0][0].health > 0 && p[1][0].health <= 0)

cout << 1 << endl;

else if (p[0][0].health <= 0 && p[1][0].health > 0)

cout << -1 << endl;

for (int j = 0; j < 2; j++)

{

cout << p[j][0].health << endl;

cout << p[j].size() - 1 << " ";

for (int i = 1; i < p[j].size(); i++)

cout << p[j][i].health << " ";

cout << endl;

}

return 0;

}

// 160904Transport Planning.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//G国国王来中国参观后，被中国的高速铁路深深的震撼，决定为自己的国家也建设一个高速铁路系统。

//建设高速铁路投入非常大，为了节约建设成本，G国国王决定不新建铁路，而是将已有的铁路改造成高速铁路。

//现在，请你为G国国王提供一个方案，将现有的一部分铁路改造成高速铁路，使得任何两个城市间都可以通过高速铁路到达，

//而且从所有城市乘坐高速铁路到首都的最短路程和原来一样长。请你告诉G国国王在这些条件下最少要改造多长的铁路。

//输入格式

//输入的第一行包含两个整数n, m，分别表示G国城市的数量和城市间铁路的数量。所有的城市由1到n编号，首都为1号。

//接下来m行，每行三个整数a, b, c，表示城市a和城市b之间有一条长度为c的双向铁路。这条铁路不会经过a和b以外的城市。

//输出格式

//输出一行，表示在满足条件的情况下最少要改造的铁路长度。

//样例输入

//4 5

//1 2 4

//1 3 5

//2 3 2

//2 4 3

//3 4 2

//样例输出

//11

//评测用例规模与约定

//对于20%的评测用例，1 ≤ n ≤ 10，1 ≤ m ≤ 50；

//对于50%的评测用例，1 ≤ n ≤ 100，1 ≤ m ≤ 5000；

//对于80%的评测用例，1 ≤ n ≤ 1000，1 ≤ m ≤ 50000；

//对于100%的评测用例，1 ≤ n ≤ 10000，1 ≤ m ≤ 100000，1 ≤ a, b ≤ n，1 ≤ c ≤ 1000。输入保证每个城市都可以通过铁路达到首都。

#include "stdafx.h"

#include <iostream>

#include <queue>

#include <vector>

#include <algorithm>

#include <set>

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

using namespace std;

struct node

{

int des, len;

node(int d, int l) :des(d), len(l) {};

friend bool operator<(node n1, node n2)

{

return n1.len > n2.len;

}

};

vector<node> v[100010];

int n, m;

int a, b, c;

int len[10010];

int cost[10010];

void bfs()

{

set<node> se;

priority\_queue<node> q;

q.push(node(1, 0));

len[1] = 0;

cost[1] = 0;

while (!q.empty())

{

node t = q.top();

q.pop();

for (int i = 0; i < v[t.des].size(); i++)

{

node tt = v[t.des][i];

if (t.len + tt.len < len[tt.des])

{

len[tt.des] = t.len + tt.len;

cost[tt.des] = tt.len;

q.push(node(tt.des, len[tt.des]));

}

if (t.len + tt.len == len[tt.des])

cost[tt.des] = min(cost[tt.des], tt.len);

}

}

}

int main()

{

cin >> n >> m;

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

{

cin >> a >> b >> c;

v[a].push\_back(node(b, c));

v[b].push\_back(node(a, c));

}

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

{

len[i] = 1 << 30;

cost[i] = 1 << 30;

}

bfs();

long long ans = 0;

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

ans += cost[i];

cout << ans << endl;

return 0;

}

// 1612Mediant.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//在一个整数序列a1, a2, …, an中，如果存在某个数，大于它的整数数量等于小于它的整数数量，则称其为中间数。

//在一个序列中，可能存在多个下标不相同的中间数，这些中间数的值是相同的。

//给定一个整数序列，请找出这个整数序列的中间数的值。

//输入格式

//输入的第一行包含了一个整数n，表示整数序列中数的个数。

//第二行包含n个正整数，依次表示a1, a2, …, an。

//输出格式

//如果约定序列的中间数存在，则输出中间数的值，否则输出 - 1表示不存在中间数。

//样例输入

//6

//2 6 5 6 3 5

//样例输出

//5

//样例说明

//比5小的数有2个，比5大的数也有2个。

//样例输入

//4

//3 4 6 7

//样例输出

//- 1

//样例说明

//在序列中的4个数都不满足中间数的定义。

//样例输入

//5

//3 4 6 6 7

//样例输出

//- 1

//样例说明

//在序列中的5个数都不满足中间数的定义。

//评测用例规模与约定

//对于所有评测用例，1 ≤ n ≤ 1000，1 ≤ ai ≤ 1000。

#include "stdafx.h"

#include <iostream>

#include <algorithm>

using namespace std;

int n;

int a[1005];

int main()

{

cin >> n;

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

cin >> a[i];

sort(a, a + n);

int sm = 0;

int bi = 0;

int i;

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

{

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

{

if (a[i] <= a[j])

{

sm = j;

break;

}

}

for (int j = n - 1; j >= 0; j--)

{

if (a[i] >= a[j])

{

bi = n - 1 - j;

break;

}

}

if (sm == bi)

{

cout << a[i] << endl;

break;

}

}

if (i == n)

cout << -1 << endl;

return 0;

}

// 161202Salary Accounting.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//小明的公司每个月给小明发工资，而小明拿到的工资为交完个人所得税之后的工资。

//假设他一个月的税前工资（扣除五险一金后、未扣税前的工资）为S元，则他应交的个人所得税按如下公式计算：

//1） 个人所得税起征点为3500元，若S不超过3500，则不交税，3500元以上的部分才计算个人所得税，令A = S - 3500元；

//2） A中不超过1500元的部分，税率3%；

//3） A中超过1500元未超过4500元的部分，税率10%；

//4） A中超过4500元未超过9000元的部分，税率20%；

//5） A中超过9000元未超过35000元的部分，税率25%；

//6） A中超过35000元未超过55000元的部分，税率30%；

//7） A中超过55000元未超过80000元的部分，税率35%；

//8） A中超过80000元的部分，税率45%；

//例如，如果小明的税前工资为10000元，则A = 10000 - 3500 = 6500元，其中不超过1500元部分应缴税1500×3 %= 45元，

//超过1500元不超过4500元部分应缴税(4500 - 1500)×10 %= 300元，超过4500元部分应缴税(6500 - 4500)×20 %= 400元。

//总共缴税745元，税后所得为9255元。

//已知小明这个月税后所得为T元，请问他的税前工资S是多少元。

//输入格式

//输入的第一行包含一个整数T，表示小明的税后所得。所有评测数据保证小明的税前工资为一个整百的数。

//输出格式

//输出一个整数S，表示小明的税前工资。

//样例输入

//9255

//样例输出

//10000

//评测用例规模与约定

//对于所有评测用例，1 ≤ T ≤ 100000。

#include "stdafx.h"

#include <iostream>

using namespace std;

long long T;

long long S;

int main()

{

cin >> T;

if (T <= 3500 && T >=0)

S = T;

else if (T <= 4955)

S = (T - 105) / 0.97;

else if (T <= 7655)

S = (T - 455) / 0.9;

else if (T <= 11255)

S = (T - 1255) / 0.8;

else if (T <= 30755)

S = (T - 1880) / 0.75;

else if (T <= 44755)

S = (T - 3805) / 0.7;

else if (T <= 61005)

S = (T - 6730) / 0.65;

else if (T <= 999999)

S = (T - 15080) / 0.55;

cout << S << endl;

return 0;

}

// 161203Permission Query.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//授权(authorization) 是各类业务系统不可缺少的组成部分，系统用户通过授权机制获得系统中各个模块的操作权限。

//本题中的授权机制是这样设计的：每位用户具有若干角色，每种角色具有若干权限。

//例如，用户 david 具有 manager 角色，manager 角色有 crm : 2 权限，则用户 david 具有 crm : 2 权限，也就是 crm 类权限的第 2 等级的权限。

//具体地，用户名和角色名称都是由小写字母组成的字符串，长度不超过 32。

//权限分为分等级权限和不分等级权限两大类。

//分等级权限由权限类名和权限等级构成，中间用冒号“ : ”分隔。

//其中权限类名也是由小写字母组成的字符串，长度不超过 32。权限等级是一位数字，从 0 到 9，数字越大表示权限等级越高。

//系统规定如果用户具有某类某一等级的权限，那么他也将自动具有该类更低等级的权限。

//例如在上面的例子中，除 crm : 2 外，用户 david 也具有 crm : 1 和 crm : 0 权限。

//不分等级权限在描述权限时只有权限类名，没有权限等级（也没有用于分隔的冒号）。

//给出系统中用户、角色和权限的描述信息，你的程序需要回答多个关于用户和权限的查询。查询可分为以下几类：

//\* 不分等级权限的查询：如果权限本身是不分等级的，则查询时不指定等级，返回是否具有该权限；

//\* 分等级权限的带等级查询：如果权限本身分等级，查询也带等级，则返回是否具有该类的该等级权限；

//\* 分等级权限的不带等级查询：如果权限本身分等级，查询不带等级，则返回具有该类权限的等级；如果不具有该类的任何等级权限，则返回“否”。

//输入格式

//输入第一行是一个正整数 p，表示不同的权限类别的数量。紧接着的 p 行被称为 P 段，每行一个字符串，描述各个权限。

//对于分等级权限，格式为 <category> : <level>，其中 <category> 是权限类名，<level> 是该类权限的最高等级。

//对于不分等级权限，字符串只包含权限类名。

//接下来一行是一个正整数 r，表示不同的角色数量。紧接着的 r 行被称为 R 段，每行描述一种角色，格式为

//<role> <s> <privilege 1> <privilege 2> ... <privilege s>

//其中 <role> 是角色名称，<s> 表示该角色具有多少种权限。后面 <s> 个字符串描述该角色具有的权限，格式同 P 段。

//接下来一行是一个正整数 u，表示用户数量。紧接着的 u 行被称为 U 段，每行描述一个用户，格式为

//<user> <t> <role 1> <role 2> ... <role t>

//其中 <user> 是用户名，<t> 表示该用户具有多少种角色。后面 <t> 个字符串描述该用户具有的角色。

//接下来一行是一个正整数 q，表示权限查询的数量。

//紧接着的 q 行被称为 Q 段，每行描述一个授权查询，格式为 <user> <privilege>，表示查询用户 <user> 是否具有 <privilege> 权限。

//如果查询的权限是分等级权限，则查询中的 <privilege> 可指定等级，表示查询该用户是否具有该等级的权限；

//也可以不指定等级，表示查询该用户具有该权限的等级。对于不分等级权限，只能查询该用户是否具有该权限，查询中不能指定等级。

//输出格式

//输出共 q 行，每行为 false、true，或者一个数字。false 表示相应的用户不具有相应的权限，true 表示相应的用户具有相应的权限。

//对于分等级权限的不带等级查询，如果具有权限，则结果是一个数字，表示该用户具有该权限的（最高）等级。

//如果用户不存在，或者查询的权限没有定义，则应该返回 false。

//样例输入

//3

//crm:2

// git : 3

// game

// 4

// hr 1 crm : 2

// it 3 crm : 1 git : 1 game

// dev 2 git : 3 game

// qa 1 git : 2

// 3

// alice 1 hr

// bob 2 it qa

// charlie 1 dev

// 9

// alice game

// alice crm : 2

// alice git : 0

// bob git

// bob poweroff

// charlie game

// charlie crm

// charlie git : 3

// malice game

// 样例输出

// false

// true

// false

// 2

// false

// true

// false

// true

// false

// 样例说明

// 样例输入描述的场景中，各个用户实际的权限如下：

// \* 用户 alice 具有 crm : 2 权限

// \* 用户 bob 具有 crm : 1、git : 2 和 game 权限

// \* 用户 charlie 具有 git : 3 和 game 权限

// \* 用户 malice 未描述，因此不具有任何权限

// 评测用例规模与约定

// 评测用例规模：

// \* 1 ≤ p, r, u ≤ 100

// \* 1 ≤ q ≤ 10 000

// \* 每个用户具有的角色数不超过 10，每种角色具有的权限种类不超过 10

// 约定：

// \* 输入保证合法性，包括：

// 1) 角色对应的权限列表（R 段）中的权限都是之前（P 段）出现过的，权限可以重复出现，如果带等级的权限重复出现，以等级最高的为准

// 2) 用户对应的角色列表（U 段）中的角色都是之前（R 段）出现过的，如果多个角色都具有某一分等级权限，以等级最高的为准

// 3) 查询（Q 段）中的用户名和权限类名不保证在之前（U 段和 P 段）出现过

// \* 前 20 % 的评测用例只有一种角色

// \* 前 50 % 的评测用例权限都是不分等级的，查询也都不带等级

#include "stdafx.h"

#include <iostream>

#include <map>

#include <string>

#include <vector>

#include <algorithm>

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

using namespace std;

struct autho

{

string au\_str;

int au\_int;

};

struct role

{

string ro\_str;

struct ro\_au

{

string ro\_au\_str;

int ro\_au\_int;

};

vector<ro\_au> ro\_vec;

};

struct user

{

string us\_str;

vector<string> us\_vec;

};

int p, r, u, q;

role Role[110];

autho Autho[110];

user User[110];

string::size\_type idx;

int ii, jj, kk;

int main()

{

cin >> p;

ii = 0;

while (p--)

{

char s[40];

string str;

cin >> s;

str = s;

string t;

int a = 0;

idx = str.find(':');

if (idx != string::npos)

{

int i;

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

{

if (str[i] != ':')

t += str[i];

else

break;

}

i++;

while (i < str.size())

{

a = a \* 10 + str[i] - '0';

i++;

}

}

else

{

int i;

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

{

t += str[i];

}

a = -1;

}

Autho[ii].au\_str = t;

Autho[ii].au\_int = a;

ii++;

}

cin >> r;

jj = 0;

int i;

while (r--)

{

char st[40];

cin >> st;

int j;

cin >> j;

while (j--)

{

char res[40];

cin >> res;

string t;

int a = 0;

string str = res;

bool flag = false;

idx = str.find(':');

if (idx != string::npos)

{

int i;

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

{

if (str[i] != ':')

t += str[i];

else

break;

}

i++;

while (i < str.size())

{

a = a \* 10 + str[i] - '0';

i++;

}

}

else

{

int i;

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

{

t += str[i];

}

a = -1;

}

role::ro\_au Ro\_Au;

Ro\_Au.ro\_au\_int = a;

Ro\_Au.ro\_au\_str = t;

Role[jj].ro\_vec.push\_back(Ro\_Au);

}

string stst = st;

Role[jj].ro\_str = stst;

jj++;

}

cin >> u;

kk = 0;

while (u--)

{

char use[40];

cin >> use;

int k;

cin >> k;

while (k--)

{

char cls[40];

cin >> cls;

string t;

t = cls;

User[kk].us\_vec.push\_back(t);

}

string t;

t = use;

User[kk].us\_str = t;

kk++;

}

//int iii = ii - 1;

//int jjj = jj - 1;

//int kkk = kk - 1;

cin >> q;

while (q--)

{

bool result = false;

int results = -1;

bool flag1 = false;//输入是否有数字

bool flag2 = false;//输出是否用数字

char t[40];

char tt[40];

cin >> t >> tt;

string t1, tt1;

t1 = t;

tt1 = tt;

string qtt;

int a = 0;

idx = tt1.find(':');

if (idx != string::npos)

{

flag1 = true;

int i;

for (i = 0; i < tt1.size(); i++)

{

if (tt1[i] != ':')

qtt += tt1[i];

else

break;

}

i++;

while (i < tt1.size())

{

a = a \* 10 + tt1[i] - '0';

i++;

}

}

else

{

int i;

for (i = 0; i < tt1.size(); i++)

{

qtt += tt1[i];

}

a = -1;

}

int it;

vector<string> r;

int i;

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

if (User[i].us\_str == t1)

{

r = User[i].us\_vec;

break;

}

int check = -1;

int itit;

string rr;

for (it = 0; it < r.size(); it++)

{

rr = r[it];

for (itit = 0; itit < jj; itit++)

{

if (Role[itit].ro\_str == rr)

{

int iit;

for (iit = 0; iit < Role[itit].ro\_vec.size(); iit++)

{

string t1t1 = Role[itit].ro\_vec[iit].ro\_au\_str;

int check = Role[itit].ro\_vec[iit].ro\_au\_int;

if (qtt == t1t1)

{

if (flag1)

{

if (check >= a)

result = true;

}

else

{

int iitt;

for (iitt = 0; iitt < ii; iitt++)

{

if (Autho[iitt].au\_str == t1t1)

{

int y = Autho[iitt].au\_int;

if (y == -1)

result = true;

else

{

results = max(results, check);

flag2 = true;

}

break;

}

}

}

}

}

}

}

}

if (flag2)

if (results == -1)

cout << "true" << endl;

else

cout << results << endl;

else

if (result)

cout << "true" << endl;

else

cout << "false" << endl;

}

return 0;

}

// 161204Compressed Coding.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//给定一段文字，已知单词a1, a2, …, an出现的频率分别t1, t2, …, tn。

//可以用01串给这些单词编码，即将每个单词与一个01串对应，使得任何一个单词的编码（对应的01串）不是另一个单词编码的前缀，这种编码称为前缀码。

//使用前缀码编码一段文字是指将这段文字中的每个单词依次对应到其编码。一段文字经过前缀编码后的长度为：

//L = a1的编码长度×t1 + a2的编码长度×t2 + … + an的编码长度×tn。

//定义一个前缀编码为字典序编码，指对于1 ≤ i < n，ai的编码（对应的01串）的字典序在ai + 1编码之前，

//即a1, a2, …, an的编码是按字典序升序排列的。

// 例如，文字E A E C D E B C C E C B D B E中， 5个单词A、B、C、D、E出现的频率分别为1, 3, 4, 2, 5，

//则一种可行的编码方案是A:000, B : 001, C : 01, D : 10, E : 11，

//对应的编码后的01串为1100011011011001010111010011000111，对应的长度L为3×1 + 3×3 + 2×4 + 2×2 + 2×5 = 34。

// 在这个例子中，如果使用哈夫曼(Huffman)编码，对应的编码方案是A : 000, B : 01, C : 10, D : 001, E : 11，

//虽然最终文字编码后的总长度只有33，但是这个编码不满足字典序编码的性质，比如C的编码的字典序不在D的编码之前。

// 在这个例子中，有些人可能会想的另一个字典序编码是A : 000, B : 001, C : 010, D : 011, E : 1，编码后的文字长度为35。

// 请找出一个字典序编码，使得文字经过编码后的长度L最小。在输出时，你只需要输出最小的长度L，而不需要输出具体的方案。

//在上面的例子中，最小的长度L为34。

// 输入格式

// 输入的第一行包含一个整数n，表示单词的数量。

// 第二行包含n个整数，用空格分隔，分别表示a1, a2, …, an出现的频率，即t1, t2, …, tn。

//请注意a1, a2, …, an具体是什么单词并不影响本题的解，所以没有输入a1, a2, …, an。

// 输出格式

// 输出一个整数，表示文字经过编码后的长度L的最小值。

// 样例输入

// 5

// 1 3 4 2 5

// 样例输出

// 34

// 样例说明

// 这个样例就是问题描述中的例子。如果你得到了35，说明你算得有问题，请自行检查自己的算法而不要怀疑是样例输出写错了。

// 评测用例规模与约定

// 对于30%的评测用例，1 ≤ n ≤ 10，1 ≤ ti ≤ 20；

// 对于60%的评测用例，1 ≤ n ≤ 100，1 ≤ ti ≤ 100；

// 对于100%的评测用例，1 ≤ n ≤ 1000，1 ≤ ti ≤ 10000。

#include "stdafx.h"

#include<iostream>

using namespace std;

const int INF = 1 << 30;

int dp[1010][1010], a[1010], sum[1010];

int n;

int main()

{

int i, j, k, t;

cin >> n;

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

{

cin >> a[i];

sum[i] = sum[i - 1] + a[i];//用于指定区间内的计算总数

}

//沿斜线扫描

for (j = 2; j <= n; j++)

{

for (i = 1, k = j; i <= n - j + 1; i++, k++)

{

dp[i][k] = INF;

for (t = i; t<k; t++)

{

if (dp[i][k] > dp[i][t] + dp[t + 1][k] + sum[k] - sum[i - 1])

{

dp[i][k] = dp[i][t] + dp[t + 1][k] + sum[k] - sum[i - 1];

}

}

}

}

cout << dp[1][n];

return 0;

}

// Dividing Cakes.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//小明今天生日，他有n块蛋糕要分给朋友们吃，这n块蛋糕（编号为1到n）的重量分别为a1, a2, …, an。

//小明想分给每个朋友至少重量为k的蛋糕。

//小明的朋友们已经排好队准备领蛋糕，

//对于每个朋友，小明总是先将自己手中编号最小的蛋糕分给他，

//当这个朋友所分得蛋糕的重量不到k时，再继续将剩下的蛋糕中编号最小的给他，

//直到小明的蛋糕分完或者这个朋友分到的蛋糕的总重量大于等于k。

//请问当小明的蛋糕分完时，总共有多少个朋友分到了蛋糕。

//输入格式

//输入的第一行包含了两个整数n, k，意义如上所述。

//第二行包含n个正整数，依次表示a1, a2, …, an。

//输出格式

//输出一个整数，表示有多少个朋友分到了蛋糕。

//样例输入

//6 9

//2 6 5 6 3 5

//样例输出

//3

//样例说明

//第一个朋友分到了前3块蛋糕，第二个朋友分到了第4、5块蛋糕，第三个朋友分到了最后一块蛋糕。

//评测用例规模与约定

//对于所有评测用例，1 ≤ n ≤ 1000，1 ≤ k ≤ 10000，1 ≤ ai ≤ 1000。

#include "stdafx.h"

#include <iostream>

using namespace std;

int n, k;

int a[1010];

int sum = 0;

int temp = 0;

int main()

{

cin >> n >> k;

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

cin >> a[i];

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

{

if (temp >= k)

{

temp = a[i];

sum++;

}

else

{

temp += a[i];

}

}

if (temp != 0)

sum++;

cout << sum << endl;

return 0;

}

// Students Queuing.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//体育老师小明要将自己班上的学生按顺序排队。

//他首先让学生按学号从小到大的顺序排成一排，学号小的排在前面，然后进行多次调整。

//一次调整小明可能让一位同学出队，向前或者向后移动一段距离后再插入队列。

//例如，下面给出了一组移动的例子，例子中学生的人数为8人。

//0）初始队列中学生的学号依次为1, 2, 3, 4, 5, 6, 7, 8；

//1）第一次调整，命令为“3号同学向后移动2”，表示3号同学出队，向后移动2名同学的距离，

//再插入到队列中，新队列中学生的学号依次为1, 2, 4, 5, 3, 6, 7, 8；

//2）第二次调整，命令为“8号同学向前移动3”，表示8号同学出队，向前移动3名同学的距离，

//再插入到队列中，新队列中学生的学号依次为1, 2, 4, 5, 8, 3, 6, 7；

//3）第三次调整，命令为“3号同学向前移动2”，表示3号同学出队，向前移动2名同学的距离，

//再插入到队列中，新队列中学生的学号依次为1, 2, 4, 3, 5, 8, 6, 7。

//小明记录了所有调整的过程，请问，最终从前向后所有学生的学号依次是多少？

//请特别注意，上述移动过程中所涉及的号码指的是学号，而不是在队伍中的位置。

//在向后移动时，移动的距离不超过对应同学后面的人数，如果向后移动的距离正好等于对应同学后面的人数则该同学会移动到队列的最后面。

//在向前移动时，移动的距离不超过对应同学前面的人数，如果向前移动的距离正好等于对应同学前面的人数则该同学会移动到队列的最前面。

//输入格式

//输入的第一行包含一个整数n，表示学生的数量，学生的学号由1到n编号。

//第二行包含一个整数m，表示调整的次数。

//接下来m行，每行两个整数p, q，如果q为正，表示学号为p的同学向后移动q，如果q为负，表示学号为p的同学向前移动 - q。

//输出格式

//输出一行，包含n个整数，相邻两个整数之间由一个空格分隔，表示最终从前向后所有学生的学号。

//样例输入

//8

//3

//3 2

//8 - 3

//3 - 2

//样例输出

//1 2 4 3 5 8 6 7

//评测用例规模与约定

//对于所有评测用例，1 ≤ n ≤ 1000，1 ≤ m ≤ 1000，所有移动均合法。

#include "stdafx.h"

//用vector模拟再这里运行不了，但是在平台上可以通过

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main()

{

int n, m;

vector<int> vec;

cin >> n;

cin >> m;

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

vec.push\_back(i);

while (m--)

{

int x, y;

cin >> x >> y;

vector<int> ::iterator it;

for (it = vec.begin(); it != vec.end(); it++)

if (\*it == x)

break;

vec.erase(it);

vec.insert(it + y, x);

}

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

cout << vec[i] << " ";

cout << endl;

return 0;

}

// Markdown.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//Markdown 是一种很流行的轻量级标记语言（lightweight markup language），广泛用于撰写带格式的文档。

//例如以下这段文本就是用 Markdown 的语法写成的：

//这些用 Markdown 写成的文本，尽管本身是纯文本格式，然而读者可以很容易地看出它的文档结构。

//同时，还有很多工具可以自动把 Markdown 文本转换成 HTML 甚至 Word、PDF 等格式，取得更好的排版效果。

//例如上面这段文本通过转化得到的 HTML 代码如下所示：

//本题要求由你来编写一个 Markdown 的转换工具，完成 Markdown 文本到 HTML 代码的转换工作。

//简化起见，本题定义的 Markdown 语法规则和转换规则描述如下：

//●区块：区块是文档的顶级结构。本题的 Markdown 语法有 3 种区块格式。

//在输入中，相邻两个区块之间用一个或多个空行分隔。输出时删除所有分隔区块的空行。

//○段落：一般情况下，连续多行输入构成一个段落。段落的转换规则是在段落的第一行行首插入 `<p>`，在最后一行行末插入 `< / p>`。

//○标题：每个标题区块只有一行，由若干个 `#` 开头，接着一个或多个空格，然后是标题内容，直到行末。`#` 的个数决定了标题的等级。

//转换时，`# Heading` 转换为 `<h1>Heading< / h1>`，`## Heading` 转换为 `<h2>Heading< / h2>`，以此类推。标题等级最深为 6。

//○无序列表：无序列表由若干行组成，每行由 `\*` 开头，接着一个或多个空格，然后是列表项目的文字，直到行末。

//转换时，在最开始插入一行 `<ul>`，最后插入一行 `< / ul>`；对于每行，`\* Item` 转换为 `<li>Item< / li>`。本题中的无序列表只有一层，不会出现缩进的情况。

//●行内：对于区块中的内容，有以下两种行内结构。

//○强调：`\_Text\_` 转换为 `<em>Text< / em>`。强调不会出现嵌套，每行中 `\_` 的个数一定是偶数，且不会连续相邻。

//注意 `\_Text\_` 的前后不一定是空格字符。

//○超级链接：`[Text](Link)` 转换为 `<a href = "Link">Text< / a>`。超级链接和强调可以相互嵌套，但每种格式不会超过一层。

//输入格式

//输入由若干行组成，表示一个用本题规定的 Markdown 语法撰写的文档。

//输出格式

//输出由若干行组成，表示输入的 Markdown 文档转换成产生的 HTML 代码。

//样例输入

//# Hello

//

//Hello, world!

//样例输出

//<h1>Hello< / h1>

//<p>Hello, world!< / p>

#include "stdafx.h"

#include <iostream>

#include <string>

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

using namespace std;

int work1(string &res, string &str, int i);

int work2(string &res, string &str, int i);

int work1(string &res, string &str, int i)

{

res += "<a href=\"";

int j = i;

while (str[i] != '(')

i++;

i++;

for (; str[i] != ')'; i++)

{

if (str[i] == '\_')

i = work2(res, str, i);

else

res += str[i];

}

res += "\">";

int k = i;

i = j;

i++;

for (; str[i] != ']'; i++)

{

if (str[i] == '\_')

i = work2(res, str, i);

else

res += str[i];

}

res += "</a>";

return k;

}

int work2(string &res, string &str, int i)

{

res += "<em>";

i++;

for (; str[i] != '\_'; i++)

{

if (str[i] == '[')

i = work1(res, str, i);

else

res += str[i];

}

res += "</em>";

return i;

}

int main()

{

string str, res;

bool flag1 = false;

bool flag2 = false;

while (getline(cin, str))

{

if (str.size() == 0)

{

if (flag1)

{

flag1 = false;

res += "</ul>";

res += "\n";

}

if (flag2)

{

flag2 = false;

res += "</p>";

res += "\n";

}

continue;

}

if (str[0] == '#')

{

int num = 0, i;

for (i = 0; str[i] == '#'; i++)

num++;

res += "<h";

res += (num + '0');

res += ">";

while (str[i] == ' ')

i++;

for (; str[i]; i++)

{

if (str[i] == '[')

i = work1(res, str, i);

else if (str[i] == '\_')

i = work2(res, str, i);

else

res += str[i];

}

res += "</h";

res += (num + '0');

res += ">\n";

}

else if (str[0] == '\*')

{

if (!flag1)

{

res += "<ul>\n";

}

flag1 = true;

res += "<li>";

int i = 1;

while (str[i] == ' ')

i++;

for (; str[i]; i++)

{

if (str[i] == '[')

i = work1(res, str, i);

else if (str[i] == '\_')

i = work2(res, str, i);

else

res += str[i];

}

res += "</li>\n";

}

else

{

if (!flag2)

{

res += "<p>";

}

else

{

res += "\n";

}

flag2 = true;

int i = 0;

for (; str[i]; i++)

{

if (str[i] == '[')

i = work1(res, str, i);

else if (str[i] == '\_')

i = work2(res, str, i);

else

res += str[i];

}

}

}

if (flag1)

{

flag1 = false;

res += "</ul>";

res += "\n";

}

if (flag2)

{

flag2 = false;

res += "</p>";

res += "\n";

}

cout << res;

return 0;

}

// Subway Construction.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//A市有n个交通枢纽，其中1号和n号非常重要，为了加强运输能力，A市决定在1号到n号枢纽间修建一条地铁。

//地铁由很多段隧道组成，每段隧道连接两个交通枢纽。经过勘探，有m段隧道作为候选，

//两个交通枢纽之间最多只有一条候选的隧道，没有隧道两端连接着同一个交通枢纽。

//现在有n家隧道施工的公司，每段候选的隧道只能由一个公司施工，每家公司施工需要的天数一致。

//而每家公司最多只能修建一条候选隧道。所有公司同时开始施工。

//作为项目负责人，你获得了候选隧道的信息，现在你可以按自己的想法选择一部分隧道进行施工，请问修建整条地铁最少需要多少天。

//输入格式

//输入的第一行包含两个整数n, m，用一个空格分隔，分别表示交通枢纽的数量和候选隧道的数量。

//第2行到第m + 1行，每行包含三个整数a, b, c，表示枢纽a和枢纽b之间可以修建一条隧道，需要的时间为c天。

//输出格式

//输出一个整数，修建整条地铁线路最少需要的天数。

//样例输入

//6 6

//1 2 4

//2 3 4

//3 6 7

//1 4 2

//4 5 5

//5 6 6

//样例输出

//6

//样例说明

//可以修建的线路有两种。

//第一种经过的枢纽依次为1, 2, 3, 6，所需要的时间分别是4, 4, 7，则整条地铁线需要7天修完；

//第二种经过的枢纽依次为1, 4, 5, 6，所需要的时间分别是2, 5, 6，则整条地铁线需要6天修完。

//第二种方案所用的天数更少。

//评测用例规模与约定

//对于20%的评测用例，1 ≤ n ≤ 10，1 ≤ m ≤ 20；

//对于40%的评测用例，1 ≤ n ≤ 100，1 ≤ m ≤ 1000；

//对于60%的评测用例，1 ≤ n ≤ 1000，1 ≤ m ≤ 10000，1 ≤ c ≤ 1000；

//对于80%的评测用例，1 ≤ n ≤ 10000，1 ≤ m ≤ 100000；

//对于100%的评测用例，1 ≤ n ≤ 100000，1 ≤ m ≤ 200000，1 ≤ a, b ≤ n，1 ≤ c ≤ 1000000。

//所有评测用例保证在所有候选隧道都修通时1号枢纽可以通过隧道到达其他所有枢纽。

#include "stdafx.h"

#include <iostream>

#include <vector>

#include <algorithm>

#include <cstring>

using namespace std;

struct way

{

int sou, des, day;

way(int s, int d, int dd) :sou(s), des(d), day(dd) {};

bool operator< (way w)

{

return w.day > day;

}

};

int n, m;

vector<way> w;

int pre[100010];

int f(int x)

{

int f1 = x;

int f0 = x;

while (pre[f0] > 0)

f0 = pre[f0];

while (pre[f1] > 0)

{

int t = f1;

f1 = pre[f1];

pre[t] = f0;

}

return f0;

}

void u(int s, int e)

{

int f1 = f(s);

int f2 = f(e);

if (f1 != f2)

{

if (f1 <= f2)

{

pre[f1] += pre[f2];

pre[f2] = f1;

}

else

{

pre[f2] += pre[f1];

pre[f1] = f2;

}

}

}

int main()

{

cin >> n >> m;

memset(pre, 0xff, sizeof(pre));

while (m--)

{

int a, b, c;

cin >> a >> b >> c;

w.push\_back(way(a, b, c));

}

sort(w.begin(), w.end());

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

{

u(w[i].sou, w[i].des);

if (f(1) == f(n))

{

cout << w[i].day << endl;

break;

}

}

return 0;

}

// Buy some Sance.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//小明带着N元钱去买酱油。酱油10块钱一瓶，商家进行促销，每买3瓶送1瓶，或者每买5瓶送2瓶。请问小明最多可以得到多少瓶酱油。

//输入格式

//输入的第一行包含一个整数N，表示小明可用于买酱油的钱数。N是10的整数倍，N不超过300。

//输出格式

//输出一个整数，表示小明最多可以得到多少瓶酱油。

//样例输入

//40

//样例输出

//5

//样例说明

//把40元分成30元和10元，分别买3瓶和1瓶，其中3瓶送1瓶，共得到5瓶。

//样例输入

//80

//样例输出

//11

//样例说明

//把80元分成30元和50元，分别买3瓶和5瓶，其中3瓶送1瓶，5瓶送2瓶，共得到11瓶。

#include "stdafx.h"

#include <iostream>

#include <algorithm>

using namespace std;

int f(int n)

{

if (n <= 0)

return 0;

if (n <= 2)

return n;

if (n <= 4)

return n + 1;

if (n == 5)

return n + 2;

return max(f(n - 3) + 4, f(n - 5) + 7);

}

int main()

{

int sum = 0;

int mon;

cin >> mon;

mon = mon / 10;

cout << f(mon) << endl;

return 0;

}

// Public Key Box.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//有一个学校的老师共用N个教室，按照规定，所有的钥匙都必须放在公共钥匙盒里，老师不能带钥匙回家。

//每次老师上课前，都从公共钥匙盒里找到自己上课的教室的钥匙去开门，上完课后，再将钥匙放回到钥匙盒中。

//钥匙盒一共有N个挂钩，从左到右排成一排，用来挂N个教室的钥匙。

//一串钥匙没有固定的悬挂位置，但钥匙上有标识，所以老师们不会弄混钥匙。

//每次取钥匙的时候，老师们都会找到自己所需要的钥匙将其取走，而不会移动其他钥匙。

//每次还钥匙的时候，还钥匙的老师会找到最左边的空的挂钩，将钥匙挂在这个挂钩上。

//如果有多位老师还钥匙，则他们按钥匙编号从小到大的顺序还。

//如果同一时刻既有老师还钥匙又有老师取钥匙，则老师们会先将钥匙全还回去再取出。

//今天开始的时候钥匙是按编号从小到大的顺序放在钥匙盒里的。

//有K位老师要上课，给出每位老师所需要的钥匙、开始上课的时间和上课的时长，

//假设下课时间就是还钥匙时间，请问最终钥匙盒里面钥匙的顺序是怎样的？

//输入格式

//输入的第一行包含两个整数N, K。

//接下来K行，每行三个整数w, s, c，分别表示一位老师要使用的钥匙编号、开始上课的时间和上课的时长。

//可能有多位老师使用同一把钥匙，但是老师使用钥匙的时间不会重叠。

//保证输入数据满足输入格式，你不用检查数据合法性。

//输出格式

//输出一行，包含N个整数，相邻整数间用一个空格分隔，依次表示每个挂钩上挂的钥匙编号。

//样例输入

//5 2

//4 3 3

//2 2 7

//样例输出

//1 4 3 2 5

//样例说明

//第一位老师从时刻3开始使用4号教室的钥匙，使用3单位时间，所以在时刻6还钥匙。

//第二位老师从时刻2开始使用钥匙，使用7单位时间，所以在时刻9还钥匙。

//每个关键时刻后的钥匙状态如下（X表示空）：

//时刻2后为1X345；

//时刻3后为1X3X5；

//时刻6后为143X5；

//时刻9后为14325。

//样例输入

//5 7

//1 1 14

//3 3 12

//1 15 12

//2 7 20

//3 18 12

//4 21 19

//5 30 9

//样例输出

//1 2 3 5 4

//评测用例规模与约定

//对于30%的评测用例，1 ≤ N, K ≤ 10, 1 ≤ w ≤ N, 1 ≤ s, c ≤ 30；

//对于60%的评测用例，1 ≤ N, K ≤ 50，1 ≤ w ≤ N，1 ≤ s ≤ 300，1 ≤ c ≤ 50；

//对于所有评测用例，1 ≤ N, K ≤ 1000，1 ≤ w ≤ N，1 ≤ s ≤ 10000，1 ≤ c ≤ 100。

#include "stdafx.h"

#include <iostream>

#include <algorithm>

#include <cstring>

using namespace std;

struct teacher

{

int w, s, c;

friend bool operator< (teacher t, teacher tt)

{

if (t.c == tt.c)

return t.w < tt.w;

return t.c < tt.c;

}

};

struct teacher1

{

int w, s, c;

friend bool operator< (teacher1 t, teacher1 tt)

{

return t.s < tt.s;

}

};

teacher tea[1010];

teacher1 tea1[1010];

int box[1010];

int n, k;

int w, s, c;

int mini = 10010, maxi = 0;

int main()

{

cin >> n >> k;

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

box[i] = i + 1;

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

{

cin >> w >> s >> c;

c = s + c;

tea[i].w = w;

tea[i].s = s;

tea[i].c = c;

tea1[i].w = w;

tea1[i].s = s;

tea1[i].c = c;

mini = min(mini, s);

maxi = max(maxi, c);

}

sort(tea, tea + k);

sort(tea1, tea1 + k);

for (int i = mini, k = 0, k1 = 0; i <= maxi; i++)

{

while (i == tea[k].c)

{

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

if (box[j] == 0)

{

box[j] = tea[k].w;

k++;

break;

}

}

while (i == tea1[k1].s)

{

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

if (box[j] == tea1[k1].w)

{

box[j] = 0;

k1++;

break;

}

}

}

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

cout << box[i] << " ";

cout << endl;

return 0;

}

// JSON Query.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式，可以用来描述半结构化的数据。

//JSON 格式中的基本单元是值(value)，出于简化的目的本题只涉及 2 种类型的值：

//\* 字符串(string)：字符串是由双引号 " 括起来的一组字符（可以为空）。

//如果字符串的内容中出现双引号 "，在双引号前面加反斜杠，也就是用 \" 表示；如果出现反斜杠 \，则用两个反斜杠 \\ 表示。

//反斜杠后面不能出现 " 和 \ 以外的字符。例如：""、"hello"、"\"\\"。

//\* 对象(object)：对象是一组键值对的无序集合（可以为空）。键值对表示对象的属性，键是属性名，值是属性的内容。

//对象以左花括号 { 开始，右花括号 } 结束，键值对之间以逗号, 分隔。一个键值对的键和值之间以冒号 : 分隔。

//

//键必须是字符串，同一个对象所有键值对的键必须两两都不相同；值可以是字符串，也可以是另一个对象。

//例如：{}、{ "foo": "bar" }、{ "Mon": "weekday", "Tue" : "weekday", "Sun" : "weekend" }。

//除了字符串内部的位置，其他位置都可以插入一个或多个空格使得 JSON 的呈现更加美观，也可以在一些地方换行，

//不会影响所表示的数据内容。例如，上面举例的最后一个 JSON 数据也可以写成如下形式。

//{

// "Mon": "weekday",

// "Tue" : "weekday",

// "Sun" : "weekend"

//}

//给出一个 JSON 格式描述的数据，以及若干查询，编程返回这些查询的结果。

//输入格式

//第一行是两个正整数 n 和 m，分别表示 JSON 数据的行数和查询的个数。

//接下来 n 行，描述一个 JSON 数据，保证输入是一个合法的 JSON 对象。

//接下来 m 行，每行描述一个查询。给出要查询的属性名，要求返回对应属性的内容。

//需要支持多层查询，各层的属性名之间用小数点.连接。保证查询的格式都是合法的。

//输出格式

//对于输入的每一个查询，按顺序输出查询结果，每个结果占一行。

//如果查询结果是一个字符串，则输出 STRING <string>，其中 <string> 是字符串的值，中间用一个空格分隔。

//如果查询结果是一个对象，则输出 OBJECT，不需要输出对象的内容。

//如果查询结果不存在，则输出 NOTEXIST。

//样例输入

//10 5

//{

// "firstName": "John",

// "lastName" : "Smith",

// "address" : {

// "streetAddress": "2ndStreet",

// "city" : "NewYork",

// "state" : "NY"

// },

// "esc\\aped" : "\"hello\""

//}

//firstName

//address

//address.city

//address.postal

//esc\aped

//样例输出

//STRING John

//OBJECT

//STRING NewYork

//NOTEXIST

//STRING "hello"

//评测用例规模与约定

//n ≤ 100，每行不超过 80 个字符。

//m ≤ 100，每个查询的长度不超过 80 个字符。

//字符串中的字符均为 ASCII 码 33 - 126 的可打印字符，不会出现空格。所有字符串都不是空串。

//所有作为键的字符串不会包含小数点.。查询时键的大小写敏感。

//50 % 的评测用例输入的对象只有 1 层结构，80%的评测用例输入的对象结构层数不超过 2 层。

//举例来说，{ "a": "b" } 是一层结构的对象，{ "a": {"b": "c"} } 是二层结构的对象，以此类推。

#include "stdafx.h"

#include <iostream>

#include <map>

#include <cstring>

#include <string>

using namespace std;

string s, str;

map<string, string> json;

bool key = true;

string handle(string str, string s)

{

if (s.empty())

return str;

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

{

if (s[i] == ' ')

continue;

else if (s[i] == '{')

{

json[str] = "OBJECT";

key = true;

}

else if (s[i] == '}')

{

int j;

for (j = str.size() - 1; j >= 0; j--)

if (str[j] == '.')

break;

if (j < 0)

str = "";

else

str = str.substr(0, j);

}

else if (s[i] == ':')

key = false;

else if (s[i] == ',')

key = true;

else if (s[i] == '"')

{

string temp;

for (i = i + 1; i < s.size(); i++)

{

if (s[i] == '"')

break;

if (s[i] == '\\')

temp += s[++i];

else

temp += s[i];

}

if (key)

{

if (str == "")

str = temp;

else

str += '.' + temp;

}

else

{

json[str] = "STRING " + temp;

int j;

for (j = str.size() - 1; j >= 0; j--)

if (str[j] == '.')

break;

if (j < 0)

str = "";

else

str = str.substr(0, j);

}

}

}

return str;

}

int main()

{

int n, m;

cin >> n >> m;

cin.get();

getchar();

while (n--)

{

getline(cin, s);

str = handle(str, s);

}

while (m--)

{

cin >> s;

cout << (json[s] == "" ? "NOTEXIST" : json[s]) << endl;

}

return 0;

}

// Communication Network.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//某国的军队由N个部门组成，为了提高安全性，部门之间建立了M条通路，每条通路只能单向传递信息，

//即一条从部门a到部门b的通路只能由a向b传递信息。

//信息可以通过中转的方式进行传递，即如果a能将信息传递到b，b又能将信息传递到c，则a能将信息传递到c。

//一条信息可能通过多次中转最终到达目的地。

//由于保密工作做得很好，并不是所有部门之间都互相知道彼此的存在。

//只有当两个部门之间可以直接或间接传递信息时，他们才彼此知道对方的存在。部门之间不会把自己知道哪些部门告诉其他部门。

//

//上图中给了一个4个部门的例子，图中的单向边表示通路。

//部门1可以将消息发送给所有部门，部门4可以接收所有部门的消息，所以部门1和部门4知道所有其他部门的存在。

//部门2和部门3之间没有任何方式可以发送消息，所以部门2和部门3互相不知道彼此的存在。

//现在请问，有多少个部门知道所有N个部门的存在。或者说，有多少个部门所知道的部门数量（包括自己）正好是N。

//输入格式

//输入的第一行包含两个整数N, M，分别表示部门的数量和单向通路的数量。所有部门从1到N标号。

//接下来M行，每行两个整数a, b，表示部门a到部门b有一条单向通路。

//输出格式

//输出一行，包含一个整数，表示答案。

//样例输入

//4 4

//1 2

//1 3

//2 4

//3 4

//样例输出

//2

//样例说明

//部门1和部门4知道所有其他部门的存在。

//评测用例规模与约定

//对于30%的评测用例，1 ≤ N ≤ 10，1 ≤ M ≤ 20；

//对于60%的评测用例，1 ≤ N ≤ 100，1 ≤ M ≤ 1000；

//对于100%的评测用例，1 ≤ N ≤ 1000，1 ≤ M ≤ 10000。

#include "stdafx.h"

#include <iostream>

#include <vector>

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

using namespace std;

vector<int> v[1024];

int way[1024][1024];

int vis[1024];

int n, m;

int total = 0;

void dfs(int u, int vis[], int cur)

{

vis[u] = 1;

way[u][cur] = way[cur][u] = 1;

for (size\_t i = 0; i < v[u].size(); i++)

{

if (vis[v[u][i]] == 0)

{

dfs(v[u][i], vis, cur);

}

}

}

int main()

{

cin >> n >> m;

memset(way, 0, sizeof(way));

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

{

int a, b;

cin >> a >> b;

v[a].push\_back(b);

}

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

{

memset(vis, 0, sizeof(vis));

dfs(i, vis, i);

}

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

{

int j;

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

{

if (!way[i][j])

break;

}

if (j == n + 1)

total++;

}

cout << total << endl;

return 0;

}

// Minimum Difference.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//给定n个数，请找出其中相差（差的绝对值）最小的两个数，输出它们的差值的绝对值。

//输入格式

//输入第一行包含一个整数n。

//第二行包含n个正整数，相邻整数之间使用一个空格分隔。

//输出格式

//输出一个整数，表示答案。

//样例输入

//5

//1 5 4 8 20

//样例输出

//1

//样例说明

//相差最小的两个数是5和4，它们之间的差值是1。

//样例输入

//5

//9 3 6 1 3

//样例输出

//0

//样例说明

//有两个相同的数3，它们之间的差值是0.

//数据规模和约定

//对于所有评测用例，2 ≤ n ≤ 1000，每个给定的整数都是不超过10000的正整数。

#include "stdafx.h"

#include <iostream>

#include <algorithm>

#include <cmath>

using namespace std;

int n;

int mini = 10000;

int a[1010];

int Count(int a[], int b, int e, int m)

{

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

for (int j = m + 1; j <= e; j++)

mini = min(abs(a[i] - a[j]), mini);

return mini;

}

int mergesort(int a[], int b, int e)

{

int m = b + (e - b) / 2;

int t;

if (b < e)

{

mini = min(mini, mergesort(a, b, m));

mini = min(mergesort(a, m + 1, e), mini);

mini = min(Count(a, b, e, m), mini);

}

return mini;

}

int main()

{

cin >> n;

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

cin >> a[i];

mini = min(mini, mergesort(a, 0, n - 1));

cout << mini << endl;

return 0;

}

// Game.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//有n个小朋友围成一圈玩游戏，小朋友从1至n编号，2号小朋友坐在1号小朋友的顺时针方向，3号小朋友坐在2号小朋友的顺时针方向，

//……，1号小朋友坐在n号小朋友的顺时针方向。

//游戏开始，从1号小朋友开始顺时针报数，接下来每个小朋友的报数是上一个小朋友报的数加1。

//若一个小朋友报的数为k的倍数或其末位数（即数的个位）为k，则该小朋友被淘汰出局，不再参加以后的报数。

//当游戏中只剩下一个小朋友时，该小朋友获胜。

//例如，当n = 5, k = 2时：

//1号小朋友报数1；

//2号小朋友报数2淘汰；

//3号小朋友报数3；

//4号小朋友报数4淘汰；

//5号小朋友报数5；

//1号小朋友报数6淘汰；

//3号小朋友报数7；

//5号小朋友报数8淘汰；

//3号小朋友获胜。

//

//给定n和k，请问最后获胜的小朋友编号为多少？

//输入格式

//输入一行，包括两个整数n和k，意义如题目所述。

//输出格式

//输出一行，包含一个整数，表示获胜的小朋友编号。

//样例输入

//5 2

//样例输出

//3

//样例输入

//7 3

//样例输出

//4

//数据规模和约定

//对于所有评测用例，1 ≤ n ≤ 1000，1 ≤ k ≤ 9。

#include "stdafx.h"

#include <iostream>

#include <queue>

using namespace std;

int n, k;

int timer;

int main()

{

cin >> n >> k;

queue<int> game;

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

{

game.push(i);

}

timer = 0;

while (game.size() != 1)

{

timer++;

int top = game.front();

game.pop();

if (timer % k != 0 && timer % 10 != k)

game.push(top);

}

cout << game.front() << endl;

return 0;

}

// Division.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//小葱喜欢除法，所以他给了你N个数a1, a2, ⋯, aN，并且希望你执行M次操作，每次操作可能有以下两种：

//给你三个数l, r, v，你需要将al, al + 1, ⋯, ar之间所有v的倍数除以v。

//给你两个数l, r，你需要回答al + al + 1 + ⋯ + ar的值是多少。

//输入格式

//第一行两个整数N, M，代表数的个数和操作的次数。

//接下来一行N个整数，代表N个数一开始的值。

//接下来M行，每行代表依次操作。每行开始有一个整数opt。

//如果opt = 1，那么接下来有三个数l, r, v，代表这次操作需要将第l个数到第r个数中v的倍数除以v；

//如果opt = 2，那么接下来有两个数l, r，代表你需要回答第l个数到第r个数的和。

//输出格式

//对于每一次的第二种操作，输出一行代表这次操作所询问的值。

//样例输入

//5 3

//1 2 3 4 5

//2 1 5

//1 1 3 2

//2 1 5

//样例输出

//15

//14

//评测用例规模与约定

//对于30%的评测用例，1 ≤ N, M ≤ 1000；

//对于另外20%的评测用例，第一种操作中一定有l = r；

//对于另外20%的评测用例，第一种操作中一定有l = 1, r = N；

//对于100%的评测用例，1 ≤ N, M ≤ 105，0 ≤ a1, a2, ⋯, aN ≤ 106, 1 ≤ v ≤ 106, 1 ≤ l ≤ r ≤ N。

#include "stdafx.h"

#include <iostream>

using namespace std;

int n, m;

int a[101024];

long long tree[101024];

int opt, l, r, v;

void update(int i, int x)

{

while (i <= n)

{

tree[i] += x;

i += -i&i;

}

}

long long getsum(int i)

{

long long sum = 0;

while (i > 0)

{

sum += tree[i];

i -= -i&i;

}

return sum;

}

int main()

{

cin >> n >> m;

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

{

cin >> a[i];

update(i, a[i]);

}

while (m--)

{

cin >> opt >> l >> r;

if (opt == 1)

{

cin >> v;

if (v == 1)

continue;

while (l <= r)

{

if (a[l] >= v && a[l] % v == 0)

{

update(l, -(a[l] - a[l] / v));

a[l] /= v;

}

l++;

}

}

else

cout << getsum(r) - getsum(l - 1) << endl;

}

return 0;

}

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

#include <string>

#include <iostream>

#include <map>

#include <cstring>

#include <vector>

#include <set>

#include <ctype.h>

using namespace std;

int n;

string s, t;

char vMon[][4] = { "", "jan", "feb", "mar", "apr", "may", "jun", "jul", "aug", "sep", "nov", "dec" };

char vWek[][4] = { "sun", "mon", "tue", "wed", "thu", "fri", "sat" };

int mtharray[] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

map<string, int> mMon, mWek;

map<string, vector<string>> mrt;

void buildmonandwekmap()

{

for (int i = 1; i <= 12; i++)

mMon[vMon[i]] = i;

for (int i = 0; i <= 6; i++)

mWek[vWek[i]] = i;

}

int stoi(string &s)

{

int len = s.size();

int t = 0;

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

{

t = t \* 10 + s[i] - '0';

}

return t;

}

string itos(int a)

{

string s = "";

while (a)

{

int t = a % 10;

s = (char)('0' + t) + s;

a = a / 10;

}

return s;

}

void standard(string &s)

{

int len = s.size();

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

s[i] = tolower(s[i]);

}

vector<string> splitsring(string &str, int TAG)

{

str += ",";

vector<string> vret;

size\_t found = str.find(',');

while (found != string::npos)

{

string x = str.substr(0, found);

str = str.substr(found + 1, str.size() - found - 1);

size\_t fdx = x.find('-');

if (fdx == string::npos)

{

if (TAG == 1 && isalpha(x[0]))

x = itos(mMon[x]);

if (TAG == 2 && isalpha(x[0]))

x = itos(mWek[x]);

if (x.size() == 1)

x = "0" + x;

vret.push\_back(x);

}

else

{

string L = x.substr(0, fdx), R = x.substr(fdx + 1, x.size() - fdx - 1);

int left, right;

if (TAG == 0)

{

left = stoi(L);

right = stoi(R);

}

else if (TAG == 1)

{

left = (isalpha(L[0])) ? mMon[L] : stoi(L);

right = (isalpha(R[0])) ? mMon[R] : stoi(R);

}

else if (TAG == 2)

{

left = (isalpha(L[0])) ? mWek[L] : stoi(L);

right = (isalpha(R[0])) ? mWek[R] : stoi(R);

}

while (left <= right)

{

string num = itos(left);

if (num.size() == 1)

num = "0" + num;

vret.push\_back(num);

left++;

}

}

found = str.find(',');

}

return vret;

}

bool isleapyear(int y)

{

if ((y % 4 == 0 && y % 100 != 0) || y % 400 == 0)

return true;

else

return false;

}

string getweekday(string yyear, string mmonth, string dday)

{

int year = stoi(yyear);

int month = stoi(mmonth);

int day = stoi(dday);

int y = 1970;

int sum = 0;

while (y < year)

{

if (isleapyear(y))

sum += 366;

else

sum += 365;

y++;

}

int m = 1;

while (m < month)

{

if (m == 2)

{

if (isleapyear(year))

sum += 29;

else

sum += 28;

}

else

sum += mtharray[m];

m++;

}

sum += day;

return "0" + itos((3 + sum % 7) % 7);

}

int main()

{

cin >> n;

cin >> s;

cin >> t;

buildmonandwekmap();

string sy = s.substr(0, 4), ty = t.substr(0, 4), smon = s.substr(4, 6), tmon = t.substr(4, 6),

sd = s.substr(6, 8), td = t.substr(6, 8), sh = s.substr(8, 10), th = t.substr(8, 10),

smin = s.substr(10, 12), tmin = t.substr(10, 12);

int syy, tyy, smmon, tmmon, sdd, tdd, shh, thh, smmin, tmmin;

syy = stoi(sy);

tyy = stoi(ty);

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

{

vector<string> vminutes, vhours, vday\_of\_month, vmonth, vday\_of\_week;

string minutes, hours, day\_of\_month, month, day\_of\_week, command;

cin >> minutes >> hours >> day\_of\_month >> month >> day\_of\_week >> command;

standard(month);

standard(day\_of\_week);

if (minutes == "\*")

minutes = "0-59";

vminutes = splitsring(minutes, 0);

if (hours == "\*")

hours = "0-23";

vhours = splitsring(hours, 0);

if (day\_of\_month == "\*")

day\_of\_month = "1-31";

vday\_of\_month = splitsring(day\_of\_month, 0);

if (month == "\*")

month = "1-12";

vmonth = splitsring(month, 1);

if (day\_of\_week == "\*")

day\_of\_week = "0-6";

vday\_of\_week = splitsring(day\_of\_week, 2);

set<string> wekexist;

for (size\_t i = 0; i < vday\_of\_week.size(); i++)

wekexist.insert(vday\_of\_week[i]);

int curyear = syy;

while (curyear <= tyy)

{

if (isleapyear(curyear))

mtharray[2] = 29;

else

mtharray[2] = 28;

string year = itos(curyear);

for (size\_t mi = 0; mi < vmonth.size(); mi++)

{

string curmonth = vmonth[mi];

for (size\_t di = 0; di < vday\_of\_month.size(); di++)

{

string curday\_of\_month = vday\_of\_month[di];

string wd = getweekday(year, curmonth, curday\_of\_month);

if (wekexist.count(wd) == 0 || stoi(curday\_of\_month) > mtharray[stoi(curmonth)])

continue;

for (size\_t hi = 0; hi < vhours.size(); hi++)

{

for (size\_t mi = 0; mi < vminutes.size(); mi++)

{

string datetime = year + curmonth + curday\_of\_month + vhours[hi] + vminutes[mi];

if (datetime < t && datetime >= s)

mrt[datetime].push\_back(command);

}

}

}

}

curyear++;

}

}

for (map<string, vector<string>>::iterator it = mrt.begin(); it != mrt.end(); it++)

{

map<string, int> isprt;

for (size\_t i = 0; i < it->second.size(); i++)

{

string dis = it->first + " " + it->second[i];

cout << dis << endl;

}

}

return 0;

}

// Travel Way.cpp: 定义控制台应用程序的入口点。

//

//问题描述

//小明和小芳出去乡村玩，小明负责开车，小芳来导航。

//小芳将可能的道路分为大道和小道。大道比较好走，每走1公里小明会增加1的疲劳度。

//小道不好走，如果连续走小道，小明的疲劳值会快速增加，连续走s公里小明会增加s2的疲劳度。

//例如：有5个路口，1号路口到2号路口为小道，2号路口到3号路口为小道，3号路口到4号路口为大道，

//4号路口到5号路口为小道，相邻路口之间的距离都是2公里。

//如果小明从1号路口到5号路口，则总疲劳值为(2 + 2)2 + 2 + 22 = 16 + 2 + 4 = 22。

//现在小芳拿到了地图，请帮助她规划一个开车的路线，使得按这个路线开车小明的疲劳度最小。

//输入格式

//输入的第一行包含两个整数n, m，分别表示路口的数量和道路的数量。路口由1至n编号，小明需要开车从1号路口到n号路口。

//接下来m行描述道路，每行包含四个整数t, a, b, c，表示一条类型为t，连接a与b两个路口，长度为c公里的双向道路。

//其中t为0表示大道，t为1表示小道。保证1号路口和n号路口是连通的。

//输出格式

//输出一个整数，表示最优路线下小明的疲劳度。

//样例输入

//6 7

//1 1 2 3

//1 2 3 2

//0 1 3 30

//0 3 4 20

//0 4 5 30

//1 3 5 6

//1 5 6 1

//样例输出

//76

//样例说明

//从1走小道到2，再走小道到3，疲劳度为52 = 25；然后从3走大道经过4到达5，疲劳度为20 + 30 = 50；最后从5走小道到6，疲劳度为1。总共为76。

//数据规模和约定

//对于30%的评测用例，1 ≤ n ≤ 8，1 ≤ m ≤ 10；

//对于另外20%的评测用例，不存在小道；

//对于另外20%的评测用例，所有的小道不相交；

//对于所有评测用例，1 ≤ n ≤ 500，1 ≤ m ≤ 105，1 ≤ a, b ≤ n，t是0或1，c ≤ 105。保证答案不超过106。

#include "stdafx.h"

#include <iostream>

#include <cstring>

#include <vector>

#include <algorithm>

#include <cmath>

#include <queue>

using namespace std;

struct way

{

int typ, sou, des, len;

way(int t, int s, int d, int l) :typ(t), sou(s), des(d), len(l) {};

};

int n, m;

vector<way> vec;

vector<int> v[100010];

int vis[100010];

int pre[100010];

int leng[100010];

bool islastsma[100010];

int lastsma[100010];

int ans = 1 << 30;

bool u(int i, int j, int k)

{

if (vec[k].typ == 1)

{

if (islastsma[i])

{

int t = leng[i] - lastsma[i] \* lastsma[i];

if (t + pow(vec[k].len + lastsma[i], 2) < leng[j])

{

leng[j] = t + pow(vec[k].len + lastsma[i], 2);

lastsma[j] = lastsma[i] + vec[k].len;

islastsma[j] = true;

return true;

}

}

else

{

if (leng[i] + pow(vec[k].len, 2) < leng[j])

{

leng[j] = leng[i] + pow(vec[k].len, 2);

lastsma[j] = vec[k].len;

islastsma[j] = true;

return true;

}

}

}

else

{

if (leng[i] + vec[k].len < leng[j])

{

leng[j] = leng[i] + vec[k].len;

islastsma[j] = false;

lastsma[j] = 0;

return true;

}

}

return false;

}

void bfs()

{

queue<int> q;

q.push(1);

vis[1] = 1;

while (!q.empty())

{

int w = q.front();

q.pop();

vis[w] = 0;

for (int i = 0; i < v[w].size(); i++)

{

if (u(vec[v[w][i]].sou, vec[v[w][i]].des, v[w][i]))

if (!vis[vec[v[w][i]].des])

{

q.push(vec[v[w][i]].des);

vis[vec[v[w][i]].des] = 1;

}

}

}

}

int main()

{

cin >> n >> m;

memset(pre, 0xff, sizeof(pre));

memset(lastsma, 0, sizeof(lastsma));

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

{

leng[i] = 1 << 30;

islastsma[i] = false;

}

leng[1] = 0;

while (m--)

{

int a, b, c, d;

cin >> a >> b >> c >> d;

vec.push\_back(way(a, b, c, d));

vec.push\_back(way(a, c, b, d));

v[b].push\_back(vec.size() - 2);

v[c].push\_back(vec.size() - 1);

}

bfs();

cout << leng[n] << endl;

return 0;

}