#include <bits/stdc++.h> // 201912-4

using namespace std;

int n, m, ux, vx, t, k;

// 操作优先级 接收-产生-查询

map<int, unordered\_map<int, array<vector<int>, 2> > > ans; // 保存操作 时刻-顶点-（0接受链,生成的块）

vector<vector<int> > graph(502); // 图的邻接表

vector<vector<int> > lis(502, {0}); // current vertex's list

bool canAccept(vector<int>& Old, vector<int>& New){

return Old.size()==New.size()? Old.back()>New.back(): Old.size()<New.size();

}

// 顶点v辐射

void cast(int v, int time){ // v辐射时刻time的状态

for(int u: graph[v]){

auto& chain = ans[time][u][0];

if((chain.empty()&&canAccept(lis[u], lis[v]))||(!chain.empty()&&canAccept(chain, lis[v])))

chain = lis[v];

}

}

void query(int a, int b){ // 时刻a顶点b的主链

for(auto& e: ans){

int currTime = e.first;

if(currTime>b)

break;

for(auto& vertex: e.second){

int v = vertex.first; // 顶点 v

auto& chain = vertex.second[0], &block = vertex.second[1];

bool flag = canAccept(lis[v], chain)|| !block.empty()? true: false; //v是否辐射

if(canAccept(lis[v], chain))

lis[v] = chain;

for(int& u: block)

lis[v].push\_back(u);

if(flag)

cast(v, currTime+t);

}

}

ans.erase(ans.begin(), ans.upper\_bound(b));

cout<< lis[a].size();

for(int& e: lis[a])

cout<< " "<< e;

cout<< endl;

}

int main(){

ios::sync\_with\_stdio(false);

cin>> n>> m;

while(m--){

cin>> ux>> vx;

graph[ux].push\_back(vx);

graph[vx].push\_back(ux);

}

cin>> t>> k;

while(k--){

int a, b;

if(cin.get()=='\n' || cin.eof())

query(a, b);

else{

int c;

cin>> c;

ans[b][a][1].push\_back(c);

}

}

return 0;

}

#include <bits/stdc++.h>

using namespace std;

int la = INT\_MAX;

void dijkstra(vector<vector<int> >&a, int v) { // start from zero

int n = a.size();

int dis[n],

pre[n],

s[n];

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

dis[i] = a[v][i];

s[i] = 0;

if(dis[i]==la)

pre[i] = -1;

else

pre[i] = v;

}

dis[v] = 0;

s[v] = 1;

for(int i=0; i<n-1; i++) {

int tmp = la;

int u = v;

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

if(s[j]==0&&dis[j]<tmp) {

u = j;

tmp = dis[j];

}

s[u] = 1;

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

if(s[j]==0&&a[u][j]<la) {

int newdis = dis[u]+a[u][j];

if(newdis<dis[j]) {

dis[j] = newdis;

pre[j] = u;

}

}

}

}

Kruskal:

struct Edge{//边的类

int v1,v2,cost;

Edge(int vv1,int vv2,int c):v1(vv1),v2(vv2),cost(c){}

bool operator <(const Edge&e)const{//重载小于运算符

return this->cost>e.cost;

}

};

priority\_queue<Edge>edges;

int father[50005];//并查集

int findFather(int x){//查找根结点并进行路径压缩

if(father[x]==x)

return x;

int temp=findFather(father[x]);

father[x]=temp;

return temp;

}

int main(){

int n,m,root,ans=0;

scanf("%d%d%d",&n,&m,&root);

iota(father,father+n+1,0);//初始化并查集

while(m--){

int a,b,c;

scanf("%d%d%d",&a,&b,&c);

edges.push(Edge(a,b,c));

}

while(!edges.empty()){

Edge e=edges.top();

edges.pop();

int ua=findFather(e.v1),ub=findFather(e.v2);

if(ua!=ub){//边的两个端点不属于同一个集合

father[ua]=ub;

ans=e.cost;//更新最长边

}

}

printf("%d",ans);

return 0;

}

int stoi(stirng s,0,int radix) 将radix进制字符串s转换成10进制

**char**\* \_itoa(int value,char\*s,int radix) 10进制转radix, 存于s；

#define maxSize 100

#define INF 100000

using namespace std;

bool vis[maxSize] = {false};

int n, edges, G[maxSize][maxSize];

void dfs(int u){

vis[u] = true;

cout << u << " ";

for(int v = 0; v < n; v++){

if(vis[v] == false && G[u][v] != INF){

dfs(v);

}

}

}

int main(void){

cout << "输入图中的顶点数: ";

cin >> n;

fill (G[0], G[0] + maxSize \* maxSize, INF);

cout << endl;

cout << "输入图中的边数: ";

cin >> edges;

cout << endl << "输入图中的起始顶点, 结束顶点和边的权重: " << endl;

int u = -1, v = -1, weight = -1;

for(int i = 0; i < edges; i++){

cin >> u >> v >> weight;

G[u][v] = weight;

}

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

if(vis[i] == false){

dfs(0);

}

}

return 0;

}

String:

void output(string& s, array<int, 3> rgb = {0, 0, 0}) { //输出

for (char c : s)

if (c == 'R' || c == 'G' || c == 'B') { //是RGB数值

string t = to\_string(rgb[c == 'R' ? 0 : c == 'G' ? 1 : 2]); //将数值转换成字符串

for (char cc : t) //遍历字符串

printf("\\x%02X", cc); //输出16进制数

} else

printf("\\x%02X", c); //输出字符的16进制数

}

int main() {

string back = "\x1b[48;2;R;G;Bm", reset = "\x1b[0m"; //背景色和重置默认值字符串

int m, n, p, q;

cin >> m >> n >> p >> q;

vector<vector<array<int, 3>>> image(n); //图像像素

string rgb;

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

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

cin >> rgb;

if (rgb.size() == 2) //只有2位字符

rgb += string(5, rgb.back()); //字符串末尾添加5个末尾字符

else if (rgb.size() == 4) //只有4位字符

rgb = "#" + string(2, rgb[1]) + string(2, rgb[2]) + string(2, rgb[3]); //添加成为6位

image[i].push\_back({0, 0, 0});

for (int t = 0; t < 3; ++t) //计算RGB数值，将16进制数转换成10进制

image[i].back()[t] = stoi(rgb.substr(2 \* t + 1, 2), 0, 16);

}

}

array<int, 3> last = {0, 0, 0}, start = {0, 0, 0};

for (int i = 0; i < n / q; ++i) { //遍历所有像素

for (int j = 0; j < m / p; ++j) {

array<int, 3> cur = {0, 0, 0};

for (int r = 0; r < q; ++r) { //计算块内RGB颜色分量之和

for (int s = 0; s < p; ++s) {

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

cur[t] += image[i \* q + r][j \* p + s][t];

}

}

for (int t = 0; t < 3; ++t) //计算RGB颜色分量平均值

cur[t] /= p \* q;

if (cur != last) //和上一个块颜色分量不同

if (cur == start) //和默认颜色分量一致，输出重置转义序列

output(reset);

else

output(back, cur);

last = cur; //更新上一个状态为当前状态

printf("\\x%02X", ' '); //每一个块后输出一个空格

}

if (last != start) //每一行结束后恢复默认状态

output(reset);

last = start; //每一行结束后更新上一个状态位默认状态

printf("\\x%02X", '\n'); //每一行字符后输出一个换行

}

return 0; }

struct Commodity { //商品类

long long id, score; //id和分数

Commodity(long long i, long long s) : id(i), score(s) {}

bool operator<(const Commodity& c) const { //重载小于运算符

return this->score != c.score ? this->score > c.score : this->id < c.id;

}

};

int main() {

const long long mul = (long long)(1e9);

int m, n, id, score, op, t, c, k;

cin >> m >> n;

vector<int> K(m); //存储每类商品被选中的最多件数

set<Commodity> commodities; //对所有商品进行排序

unordered\_map<long long, set<Commodity>::iterator> um; //存储商品id和对应的set中的迭代器

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

cin >> id >> score;

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

long long a = j \* mul + id;

um[a] = commodities.insert(Commodity(a, score)).first;

}

}

cin >> op;

while (op--) {

cin >> c;

if (c == 1) { //添加商品

cin >> t >> id >> score;

long long a = t \* mul + id;

um[a] = commodities.insert(Commodity(a, score)).first;

} else if (c == 2) { //删除商品

cin >> t >> id;

long long a = t \* mul + id;

commodities.erase(um[a]);

um.erase(a);

} else {

vector<vector<int>> ans(m);

cin >> k;

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

cin >> K[i];

for (auto& i : commodities) { //遍历整个set

t = i.id / mul;

if (ans[t].size() < K[t]) {

ans[t].push\_back(i.id % mul);

if (--k == 0) //商品已选满k件，结束遍历

break;

}

}

for (auto& i : ans)

if (i.empty()) { //没有选中的商品，输出-1

cout << "-1\n";

} else {

// sort(i.begin(), i.end());//从小到大排序，注释掉这行代码能拿到满分，不注释掉只有60分

for (auto j : i)

cout << j << " ";

cout << "\n"; }}} return 0;}

损坏的RAID5

#include<bits/stdc++.h>

using namespace std;

const int MAX=10005;

char temp[9],disks[1005][85000]={""};//存储所有磁盘数据

int main(){

int n,s,l,m,a,maxBlock=0;

scanf("%d%d%d",&n,&s,&l);

for(int i=0;i<l;++i){//读取l块磁盘的数据

scanf("%d%\*c",&a);

fgets(disks[a],85000,stdin);

maxBlock=strlen(disks[a])/8;//maxBlock存储一块磁盘上的块数

}

scanf("%d",&m);

while(m--){

scanf("%d",&a);

int band=a/s,row=band/(n-1);//计算条带号、单磁盘上的条带号

int diskNo=(n-row%n+band%(n-1))%n,block=row\*s+a%s;//计算所在磁盘号、所在磁盘上的块号

if(block>=maxBlock||(disks[diskNo][0]=='\0'&&n-l>1))//块号超过磁盘上的块数或者该磁盘被损坏且坏掉的磁盘超过1个

puts("-");//该磁盘上对应块的数据无法获取

else if(disks[diskNo][0]!='\0'){//该磁盘数据完好，直接输出对应块的数据即可

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

putchar(disks[diskNo][block\*8+i]);

puts("");

}else{//该磁盘被损坏，但数据可恢复

int ans=0,k;

for(int i=0;i<n;++i)//遍历其他的块

if(diskNo!=i){

for(int j=0;j<8;++j)//将对应的8个字符复制粘贴到temp中

temp[j]=disks[i][block\*8+j];

temp[8]='\0';//temp末尾字符置\0

sscanf(temp,"%x",&k);//将16进制字符串转换为10进制数

ans^=k;//进行异或运算

}

printf("%08X\n",ans);//输出8位16进制字符串，不够8位在高位补0

}

}

return 0;

}

消息传递接口

int main(){

int T, n, i, x, y;

scanf("%d%d%\*c", &T, &n);

while(T--){

x = y = 0;

list<queue<int> > process;

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

process.push\_back(queue<int>());

char s[100];

gets(s);

for(int j=0; s[j]&&s[j]!='\n'; j++)

if(s[j]=='S'||s[j]=='R'){

int ans = 0, k;

for(k=j+1; s[k]&&s[k]!=' '; k++)

ans = ans\*10+s[k]-'0';

if(s[j]=='S'){

process.back().push(i\*10000+ans);

x++;

}

else{

process.back().push(-(ans\*10000+i));

y++;

}

j = k-1;

}

}

if(x!=y){

cout<< 1<< endl;

continue;

}

unordered\_set<int> commands;

bool f;

while(!process.empty()){

f = true;

for(auto it = process.begin(); it!=process.end();){

auto t = commands.find(-(it->front()));

if(t==commands.end())

commands.insert(it->front());

else{

commands.erase(-(it->front()));

f = false;

}

it->pop();

if(it->empty())

it = process.erase(it);

else

it++;

}

if(f){

commands.insert(0);

break;

}

}

cout<< (commands.empty()? 0: 1)<< endl;

}

return 0;}

// CIDR合并

struct IP{//IP地址类

string ip="";//IP地址32位的二进制字符串

int length=-1;//IP地址的前缀长度

};

IP stringToIp(string&input){//将输入的IP地址转换为32位二进制IP地址，并求出前缀长度

IP ip;

string s="";

vector<int>pow2={1,2,4,8,16,32,64,128};//2的0~7次幂

for(int i=0;i<=input.size();++i)//遍历输入的IP地址

if(i==input.size()||!isdigit(input[i])){//到达字符串末尾或当前字符不是数字字符

int k=stoi(s);//将字符串s转换为整数

for(int ii=7;ii>=0;--ii)//求出当前整数的二进制表示

if(k>=pow2[ii]){

ip.ip+="1";

k-=pow2[ii];

}else

ip.ip+="0";

s="";

if(input[i]=='/'){//遇到了/字符，其后面的整数就代表了前缀长度

ip.length=stoi(input.substr(i+1));

break;

}

}else

s+=input[i];

if(ip.length==-1)//输入的IP地址中不包含前缀长度

ip.length=ip.ip.size();//前缀长度即为2进制字符串长度

while(ip.ip.size()<32)//IP地址小于32位，在末尾补0

ip.ip+="0";

return ip;

}

bool isChildCollection(IP&a,IP&b){//判断IP地址b是不是IP地址a的匹配集的子集

if(a.length>b.length)

return false;

for(int i=0;i<a.length;++i)

if(a.ip[i]!=b.ip[i])

return false;

return true;

}

void Merge1(list<IP>&ipAddress){//第一步合并，移除匹配集是前一IP地址子集的IP地址

auto i=ipAddress.begin(),j=ipAddress.begin();

for(++j;j!=ipAddress.end();){

if(isChildCollection(\*i,\*j)){

j=ipAddress.erase(j);

}else{

++i;

++j;

}

}

}

bool unionCollection(IP&a,IP&b){//判断IP地址a和b的匹配集的并集是否等于a'的匹配集

if(a.length!=b.length)

return false;

for(int i=0;i<a.length-1;++i)

if(a.ip[i]!=b.ip[i])

return false;

return a.ip[a.length-1]!=b.ip[a.length-1];

}

void Merge2(list<IP>&ipAddress){//第二步合并，同级合并

auto i=ipAddress.begin(),j=ipAddress.begin();

for(++j;j!=ipAddress.end();){

if(unionCollection(\*i,\*j)){

j=ipAddress.erase(j);

--(\*i).length;

if(i!=ipAddress.begin()){

--i;

--j;

}

}else{

++i;

++j;

}

}

}

int main(){

int N;

cin>>N;

list<IP>ipAddress;

while(N--){

string input;

cin>>input;

ipAddress.push\_back(stringToIp(input));

}

ipAddress.sort([](const IP&a,const IP&b){

if(a.ip!=b.ip)

return a.ip<b.ip;

return a.length<b.length;

});

Merge1(ipAddress);//第一步合并

Merge2(ipAddress);//第二步合并

for(auto&i:ipAddress){//输出IP地址

for(int j=0;j<4;++j){//求出每8位2进制字符串代表的整数并输出

int k=0;

for(int ii=0;ii<8;++ii)

k=k\*2+(i.ip[ii+j\*8]-'0');

printf("%d%s",k,j<3?".":"/");

}

printf("%d\n",i.length);//输出前缀长度

}

return 0;

}

// 元素选择器

struct element{

int father;//父元素所在行号

string label,id;//标签、id属性

};

vector<element>elements(105);//从下标1开始存储所有元素，下标代表每个元素所在行号

vector<vector<int>>series;

int main(){

int n,m;

string line;

scanf("%d%d%\*c",&n,&m);

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

getline(cin,line);

int j=0,k=0;

while(line[j]=='.')//循环结束后j就表示小数点数量，line[j]表示第一个非小数点的字符

++j;

for(k=j;k<line.size()&&line[k]!=' ';++k);//循环结束后,k就表示字符串末尾或字符串中空格位置

if(series.size()<=j/2)//series还没有下标为j/2的元素

series.push\_back({i});//压入一个初始化为当前元素所在行号的vector

else//series有下标为j/2的元素

series[j/2].push\_back(i);//直接把行号压入

elements[i].label=line.substr(j,k-j);//取标签字符串

for(auto&c:elements[i].label)//将元素的标签中的英文字符都变为小写字符

c=tolower(c);

elements[i].id=(k==line.size())?"":line.substr(k+1);//取id属性字符串

elements[i].father=j/2-1<0?-1:series[j/2-1].back();//求父元素所在行号

}

while(m--){

getline(cin,line);

vector<string>query;//存储查询的元素

for(int i=0;i<line.size();++i){//遍历读取的一行字符串

int j=i;

while(j<line.size()&&line[j]!=' ')

++j;

string s=line.substr(i,j-i);

if(s[0]!='#')//将标签中所有字符转换为小写字符

for(auto&c:s)

c=tolower(c);

query.push\_back(s);

i=j;

}

vector<int>ans;

for(int i=query.size()-1;i<series.size();++i)//从query.size()-1开始遍历

for(int j:series[i])//遍历当前series[i]中的元素行号

if(elements[j].label==query.back()||elements[j].id==query.back()){//如果标签或id等于最低查询元素

int k=query.size()-2;//k指向查询的倒数第2个元素

for(int p=elements[j].father;k>=0&&p!=-1;p=elements[p].father)//遍历行号为p元素的祖先元素行号

if(elements[p].label==query[k]||elements[p].id==query[k])//如果与当前查询的元素相同

--k;//继续查询前面的元素

if(k<0)//所有元素均已查询到，该元素符合要求

ans.push\_back(j);//压入ans中

}

printf("%d ",ans.size());//输出

for(auto i:ans)

printf("%d ",i);

puts("");//换行

}

return 0;

}

#include <iostream>

#include <vector>

#include <string>

#include <regex>

using namespace std;

int main()

{

//FILE \*stream;

//freopen\_s(&stream, "data.txt", "r", stdin);

int n, m;

cin >> n >> m;

vector<pair<string, regex>> rules; //<名字，规则>存放最终的映射规则

vector<pair<regex, string>> trans = {

{regex("<int>"), "([\\d]+)"}, {regex("<str>"), "([^/]+)"}, {regex("<path>"), "(.+)"}

}; //将每条规则先翻译成正则表达式的形式

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

{

string p, r;

cin >> p >> r;

for(auto &e : trans)

p = regex\_replace(p, e.first, e.second);

rules.push\_back({r, regex(p)});

}

smatch result; //存放匹配结果

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

{

string url;

cin >> url;

bool success = false;

unsigned j = 0;

for(; j < rules.size(); j++) //依次进行规则匹配

if(success = regex\_match(url, result, rules[j].second)) break; //若完全匹配成功则不必再进行下去

if(success)

{

cout << rules[j].first << ' ';

for(unsigned k = 1; k < result.size(); k++)

{

string temp = result[k];

while(temp[0] == '0') temp.erase(0, 1); //去掉数字的前导0

cout << temp << ' ';

}

cout << endl;

}

else cout << 404 << endl;

}

//fclose(stream);

return 0;

}