胡柯青tql你们说对不队

[Common 3](#_Toc7105872)

[Head 3](#_Toc7105873)

[Read-fast 3](#_Toc7105874)

[图 3](#_Toc7105875)

[最大团 3](#_Toc7105876)

[最大流 4](#_Toc7105877)

[Bfs 5](#_Toc7105878)

[Dijkstra(带路径) 8](#_Toc7105879)

[二分图匹配（匈牙利算法的DFS实现）(邻接矩阵形式) 10](#_Toc7105880)

[二分匹配的bfs 11](#_Toc7105881)

[并查集 12](#_Toc7105882)

[Bellman-ford(解决负边权) 13](#_Toc7105883)

[翻转 13](#_Toc7105884)

[数组线段树 15](#_Toc7105885)

[Math 18](#_Toc7105886)

[dancing links 18](#_Toc7105887)

[Bagage 21](#_Toc7105888)

[01背包--装满背包的方案总数 24](#_Toc7105889)

[完全背包--装满背包的方案总数 25](#_Toc7105890)

[大数 25](#_Toc7105891)

[多边形面积 31](#_Toc7105892)

[高斯消元法(浮点型) 31](#_Toc7105893)

[高斯消元法(整数) 32](#_Toc7105894)

[扩展gcd（中国剩余定理） 34](#_Toc7105895)

[欧拉筛 35](#_Toc7105896)

[判断两条线段相交 36](#_Toc7105897)

[数学公式 37](#_Toc7105898)

[凸包（二维） 37](#_Toc7105899)

[凸包（三维） 39](#_Toc7105900)

[万进制 44](#_Toc7105901)

[因数个数 45](#_Toc7105902)

[矩阵运算 48](#_Toc7105903)

[String 50](#_Toc7105904)

[Kmp 50](#_Toc7105905)

[AC自动机 51](#_Toc7105906)

[Trie 53](#_Toc7105907)

[拓扑排序之确定比赛名次(输出字典序最小的答案) 54](#_Toc7105908)

[Algorithm 56](#_Toc7105909)

[KM 56](#_Toc7105910)

[二分 58](#_Toc7105911)

[归并排序求逆序对 59](#_Toc7105912)

[ST 60](#_Toc7105913)

# Common

## Head

|  |  |  |
| --- | --- | --- |
| // C  #ifndef \_GLIBCXX\_NO\_ASSERT | // C++  **#include <algorithm>** | #if \_\_cplusplus >= 201103L |
| #include <cassert> | #include <bitset> | #include <array> |
| #endif | #include <complex> | #include <atomic> |
| #include <cctype> | #include <deque> | #include <chrono> |
| #include <cerrno> | #include <exception> | #include <condition\_variable> |
| #include <cfloat> | #include <fstream> | #include <forward\_list> |
| #include <ciso646> | #include <functional> | #include <future> |
| #include <climits> | #include <iomanip> | #include <initializer\_list> |
| #include <clocale> | #include <ios> | #include <mutex> |
| **#include <cmath>** | #include <iosfwd> | #include <random> |
| #include <csetjmp> | **#include <iostream>** | #include <ratio> |
| #include <csignal> | #include <istream> | #include <regex> |
| #include <cstdarg> | #include <iterator> | #include <scoped\_allocator> |
| #include <cstddef> | #include <limits> | #include <system\_error> |
| **#include <cstdio>** | **#include <list>** | #include <thread> |
| **#include <cstdlib>** | #include <locale> | #include <tuple> |
| **#include <cstring>** | **#include <map>** | #include <typeindex> |
| #include <ctime> | #include <memory> | #include <type\_traits> |
|  | #include <new> | **#include <unordered\_map>** |
| #if \_\_cplusplus >= 201103L | #include <numeric> | #include <unordered\_set> |
| #include <ccomplex> | #include <ostream> | #endif |
| #include <cfenv> | **#include <queue>** |  |
| #include <cinttypes> | **#include <set>** |  |
| #include <cstdalign> | **#include <sstream>** |  |
| #include <cstdbool> | **#include <stack>** |  |
| #include <cstdint> | #include <stdexcept> |  |
| #include <ctgmath> | #include <streambuf> |  |
| #include <cwchar> | **#include <string>** |  |
| #include <cwctype> | #include <typeinfo> |  |
| #endif | #include <utility> |  |
|  | #include <valarray> |  |
|  | **#include <vector>** |  |

## Read-fast

/\*

FIRST

Fast Input

1、支持正负整数、小数、字符串

2、不能判断是否是文本结束

USE:

int a;

rin>>a;

\*/

struct FastIO {

inline FastIO& operator >> (int& x) {

x = 0; char f = 0, ch = getchar();

while(ch > '9' || ch < '0') {f |= (ch == '-'); ch = getchar();}

while(ch <= '9' && ch >= '0') {x = x \* 10 + ch - 48; ch = getchar();}

x = (f ? -x : x);

return \*this;

}

inline FastIO& operator >> (long long& x) {

x = 0; char f = 0, ch = getchar();

while(ch > '9' || ch < '0') {f |= (ch == '-'); ch = getchar();}

while(ch <= '9' && ch >= '0') {x = x \* 10 + ch - 48; ch = getchar();}

x = (f ? -x : x);

return \*this;

}

inline FastIO& operator >> (double& x) {

x = 0; char f = 0, ch = getchar();

double d = 0.1;

while(ch > '9' || ch < '0') {f |= (ch == '-'); ch = getchar();}

while(ch <= '9' && ch >= '0') {x = x \* 10 + ch - 48; ch = getchar();}

if(ch == '.') {

ch = getchar();

while(ch <= '9' && ch >= '0') {x += d \* (ch - 48); d \*= 0.1; ch = getchar();}

}

x = (f ? -x : x);

return \*this;

}

inline FastIO& operator >> (string& x){

x.clear(); char ch = getchar();

while(ch == ' ' || ch == '\t' || ch == '\n') ch = getchar();

while(ch != ' ' && ch != '\t' && ch != '\n') {x.push\_back(ch); ch = getchar();}

return \*this;

}

}rin;

/\*

Second

Fast Input

1、支持正负整数、小数

2、返回是否是文本结束

USE:

int a;

FastRead (a);

\*/

template <class \_\_T>

inline bool FastRead (\_\_T& x) {

x = 0; char f = 0; int ch = getchar(); double d = 0.1;

if (ch == EOF) return false;

while(ch > '9' || ch < '0') { f |= (ch == '-'); ch = getchar(); }

while(ch <= '9' && ch >= '0') { x = x \* 10 + ch - 48; ch = getchar(); }

if (ch == '.') {

ch = getchar();

while(ch <= '9' && ch >= '0') { x += d \* (ch - 48); d \*= 0.1; ch = getchar(); }

}

return true;

}

# 图

## 最大团

const int N = 55;

int n;//图中的点数

int bestn;//最优解

int a[N][N];//图用邻接表表示

int cn;//当前已放入团中节点数量

bool x[N];//是否将第i个节点加入团中

bool bestx[N];//记录最优解

bool can\_join(int t){//判断是否可以把节点t放入团中

bool can=true;

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

if(x[i]&&a[t][i]==0){//x[i]表示i是被选中的节点，a[t][i]==0表示t和j没有边相连

can=false;

break;

}

}

return can;

}

void dfs(int t){

if(t>n){//到达叶子节点

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

bestx[i]=x[i];

}

bestn=cn;

return;

}

if(can\_join(t)){//满足约束条件，进入左子树，即把节点t加入团

x[t]=1;

cn++;

dfs(t+1);

cn--;

}

if(cn+n-t>bestn){//满足限界条件，进入右子树

x[t]=0;

dfs(t+1);

}

}

int main(){

ios\_base::sync\_with\_stdio(false);//关同步

while(cin>>n,n){

for(int i=1;i<=n;i++){//下标从1开始

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

cin>>a[i][j];

}

}

bestn=0;

cn=0;

dfs(1);

cout<<bestn<<endl;

}

}

## 最大流

//最大流.cpp

//EK算法模板（s为源点，t为汇点，函数返回值为最大流）

//e为邻接矩阵

//p储存路径，a用来计算路径

int EK(int s,int t)

{

queue<int> q;

int p[MAXN\*2],a[MAXN\*2];

int f=0;

while(1)

{

memset(a,0,sizeof(a));

a[s]=inf;

q.push(s);

while(!q.empty())

{

int u=q.front();

q.pop();

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

if(!a[v]&&flow[u][v])

{

p[v]=u,q.push(v);

a[v]=min(a[u],flow[u][v]);

if(v==t) break;

}

}

if(a[t]==0) break;

for(int u=t; u!=s; u=p[u])

{

flow[p[u]][u]-=a[t];

flow[u][p[u]]+=a[t];

}

f+=a[t];

}

return f;

}

## Bfs

#include<iostream>

#include<queue>

using namespace std;

const int INF = 100000000;

const int maxn = 10000;

typedef pair<int, int> P; //储存坐标下x,y

int maze[maxn][maxn];

int d[maxn][maxn]; //储存每个坐标的最短路径

int sx, sy; //起始坐标

int ex, ey; //终点坐标

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

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

int bfs()

{

queue<P> que; //bfs用队列

for (int i = 0; i < maxn; i++) { //初始化所有距离为极大

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

d[i][j] = INF;

}

}

que.push(P(sx, sy));

d[sx][sy] = 0;

while (que.size()) {

P p = que.front();

que.pop();

if (p.first == ex && p.second == ey)

break;

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

int nx = p.first + dx[i];

int ny = p.second + dy[i];

if (nx >= 0 && nx < N && ny >= 0 && ny < N && d[nx][ny] == INF && maze[nx][ny] == ) {

que.push(P(nx, ny));

d[nx][ny] = d[p.first][p.second] + 1;

}

}

}

return d[ex][ey];

}

#include<iostream>

#include<cstring>

using namespace std;

//节点的信息

struct node

{

int x;

int y;

int z;

int num;

} link[30000];

//每个节点包括x,y,z坐标

//num表示开始访问当前节点的时间

int vis[31][31][31];//标记节点是否入队

char mapp[31][31][31];//搜索地图

int pp,qq,t,l,r,c;//当前指针和入队指针,最小时间

int step[6][3]= {{1,0,0},{-1,0,0},{0,0,1},{0,0,-1},{0,1,0},{0,-1,0}};

int flag;//标志有没有搜索到目标节点

bool judge(int x,int y,int z)

{

if(mapp[x][y][z]=='#'||x<0||x>=l||y<0||y>=r||z<0||z>=c)

return false;

else

return true;

}

void bfs()

{

while(pp<qq)//队列非空

{

//访问节点时的操作

//对节点的访问就是对节点周围节点的搜索

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

{

int x=link[pp].x+step[i][0];

int y=link[pp].y+step[i][1];

int z=link[pp].z+step[i][2];

if(judge(x,y,z)&&vis[x][y][z]==0)//尚未入队并符合条件

{

if(mapp[x][y][z]=='.')

{

//节点入队

vis[x][y][z]=1;

link[qq].x=x;

link[qq].y=y;

link[qq].z=z;

link[qq].num=link[pp].num+1;//开始访问邻接点的时间相差为1

qq++;

}

else if(mapp[x][y][z]=='E')//目标节点寻找成功

{

flag=1;

t=link[pp].num+1;

break;

}

}

}

if(flag)

break;

pp++;//pp出队，遍历队列中的下一个节点

}

}

int main()

{

while(cin>>l>>r>>c)

{

if(l==0&&r==0&&c==0)

break;

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

pp=qq=t=0;

link[qq].num=0;

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

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

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

{

cin>>mapp[i][j][k];

if(mapp[i][j][k]=='S')//第一个节点入队

{

vis[i][j][k]=1;

link[qq].x=i;

link[qq].y=j;

link[qq].z=k;

qq++;//对头指针+1，节点入队

}

}

flag=0;

bfs();

if(flag)

cout<<"Escaped in "<<t<<" minute(s)."<<endl;

else

cout<<"Trapped!"<<endl;

}

return 0;

}

#include<iostream>

#include<cstring>

#include<algorithm>

#include<queue>

using namespace std;

const int maxn=(int)1e6+1000;

int vis[maxn];

int d[2]={1,-1};

int ex, ey;

int t;

struct node{

int x;

int num;

};

void bfs(){

queue<node> q;

node now, next;

node nod={ex, 0};

q.push(nod);

vis[ex]=1;

while(!q.empty()){

now=q.front();

if(now.x == ey){

t=now.num;

return;

}

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

if(i==2){

next.x=now.x\*2;

next.num = now.num + 1;

if(vis[next.x] == 0 && next.x>=0 && next.x<=100000){

q.push(next);

vis[next.x] = 1;

}

}else{

next.x = now.x + d[i];

next.num = now.num + 1;

if(vis[next.x] == 0 && next.x>=0 && next.x<=100000){

q.push(next);

vis[next.x] = 1;

}

}

}

q.pop();

}

}

int main(void){

while(cin >> ex >> ey){

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

bfs();

cout << t << endl;

}

}

## Dijkstra(带路径)

#include <bits/stdc++.h>

using namespace std;

const int MAX\_V=20000+1;

const int INF=1e8;

struct edge

{

int to,cost;

};

typedef pair<int,int> P;

int V,pre[MAX\_V];

vector<edge> G[MAX\_V];

int d[MAX\_V];

bool used[MAX\_V];

int n,m,S,T;

int dijkstra(int s,int t)

{

priority\_queue<P,vector<P>,greater<P> > que;

memset(used,false,sizeof(used));

fill(d+1,d+V+1,INT\_MAX);

fill(pre,pre+n+1,-1);

d[s]=0;

que.push(P(0,s));

while(!que.empty())

{

P p=que.top();

que.pop();

int v=p.second;

if(d[v]<p.first||used[v])

continue;

used[v]=true;

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

{

edge e=G[v][i];

if(d[e.to]>d[v]+e.cost)

{

d[e.to]=d[v]+e.cost;

que.push(P(d[e.to],e.to));

pre[i]=v;

}

}

}

return d[t];

}

vector<int> get\_path(int t){

vector<int >path;

for(; t != -1;t=pre[t]){

path.push\_back(t);

}

reverse(path.begin(), path.end());

return path;

}

void ini()

{

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

G[i].clear();

}

int main() {

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

while (cin >> n >> m) {

if(n==0 && m==0){

break;

}

V = n;

ini();

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

int a, b, c;

cin >> a >> b >> c;

G[a].push\_back(edge{b, c});

G[b].push\_back(edge{a, c});

}

int ans = dijkstra(1, n);

if (ans == INT\_MAX)

cout << -1 << endl;

else{

cout << ans << endl;

vector<int> path;

path=get\_path(n);

cout << 1;

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

cout << ' ' << path[i];

}

cout << endl;

}

}

return 0;

}

## 二分图匹配（匈牙利算法的DFS实现）(邻接矩阵形式)

//二分图匹配（匈牙利算法的DFS实现）(邻接矩阵形式)

//初始化：g[][]两侧顶点的可连接情况

//建立g[i][j]表示i->j的有向边就可以了，是左边向右边的匹配

//g没有边相连则初始化为0

//uN是左边的顶点数，vN是右边的顶点数

//lnk[]表示的是右侧点的归属，use是对某次查找时是否修改的标记

//调用：ans=hungary();输出最大匹配数

//优点：适用于稠密图，DFS找增广路，实现简洁易于理解

//时间复杂度:O(VE)

//该模板中 顶点的编号是从1开始的

//变种1：二分图的最小顶点覆盖

//在二分图中求最少的点，让每条边都至少和其中的一个点关联，这就是“二分图的最小顶点覆盖”。

//二分图的最小顶点覆盖数 = 二分图的最大匹配数

//变种2：DAG图（无回路有向图）的最小路径覆盖

//用尽量少的不相交简单路径覆盖有向无环图(DAG)的所有顶点，这就是DAG图的最小路径覆盖问题。

//DAG图的最小路径覆盖数 = 节点数（n）- 最大匹配数（m）

//关键：求二分图的最大匹配数

//变种3: 二分图的最大独立集

//二分图的最大独立集数 = 节点数（n）- 最大匹配数（m）

//关键：求二分图的最大匹配数

int uN,vN; //u,v的数目，使用前面必须赋值

int g[maxn][maxn];

int lnk[maxn],use[maxn];

int dfs(int u)

{

for(int v=1;v<=vN;v++)

if(g[u][v]&&!use[v]) //若 use=1 表示这次的查找 曾试图改变过该i的归属问题，但是没有成功，所以就不用费工夫了

{

use[v]=1;

if(lnk[v]==-1||dfs(lnk[v]))

{

lnk[v]=u;

return true;

}

}

return false;

}

int hungary()

{

int res=0;

mst(lnk,-1);

for(int u=1;u<=uN;u++)

{

mst(use,0);

if(dfs(u)) res++;

}

return res;

}

## 二分匹配的bfs

struct Edge

{

int from;

int to;

int weight;

Edge(int f, int t, int w):from(f), to(t), weight(w) {}

};

vector<int> G[\_\_maxNodes]; /\* G[i] 存储顶点 i 出发的边的编号 \*/

vector<Edge> edges;//记录所有边的信息

typedef vector<int>::iterator iterator\_t;

int num\_nodes;

int num\_left;

int num\_right;

int num\_edges;

int matching[\_\_maxNodes]; /\* 存储求解结果 \*/

int check[\_\_maxNodes];

queue<int> Q;

int prev[\_\_maxNodes];

int Hungarian()

{

int ans = 0;

memset(matching, -1, sizeof(matching));

memset(check, -1, sizeof(check));

for (int i=0; i<num\_left; ++i) {

if (matching[i] == -1) {

while (!Q.empty()) Q.pop();

Q.push(i);

prev[i] = -1; // 设 i 为路径起点

bool flag = false; // 尚未找到增广路

while (!Q.empty() && !flag) {

int u = Q.front();

for (iterator\_t ix = G[u].begin(); ix != G[u].end() && !flag; ++ix) {

int v = edges[\*ix].to;

if (check[v] != i) {

check[v] = i;

Q.push(matching[v]);

if (matching[v] >= 0) { // 此点为匹配点

prev[matching[v]] = u;

} else { // 找到未匹配点，交替路变为增广路

flag = true;

int d=u, e=v;

while (d != -1) {//匹配边与非匹配边互换

int t = matching[d];

matching[d] = e;

matching[e] = d;

d = prev[d];

e = t;

}

}

}

}

Q.pop();

}

if (matching[i] != -1) ++ans;

}

}

return ans;

}

}

## 并查集

const int maxn = 1000000 + 10;

const int INF = int(1e9);

using namespace std;

int fa[maxn]; //父亲结点

int Rank[maxn]; //树的的高度

void init(int n) //初始化所有结点

{

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

{

fa[i] = i;

Rank[i] = 0;

}

}

int find(int x) //查询树结点的根

{

return x == fa[x] ? x : fa[x] = find(fa[x]); //并查集压缩路径

}

void unite(int x, int y) //合并x结点和y结点所属的集合

{

x = find(x);

y = find(y);

if (x == y) return;

if (Rank[x] < Rank[y]) //低的树接到高的树上面

fa[x] = y;

else

{

fa[y] = x;

if (Rank[x] == Rank[y])

Rank[x]++;

}

}

bool same(int x, int y) //判断x和y是否在同一集合上

{

return find(x) == find(y);

}

## Bellman-ford(解决负边权)

#include<bits/stdc++.h>

#include<algorithm>

using namespace std;

const int INF=9999999;

int main(void){

int u[100],v[100],w[100],dis[100],n,m,ck,flag;

cin >> n >> m;

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

cin >> u[i] >> v[i] >> w[i];

}

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

dis[i]=INF;

}

dis[1]=0;

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

ck=0;

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

if(dis[v[i]] > dis[u[i]]+w[i]){

dis[v[i]]=dis[u[i]]+w[i];

ck=1;

}

}

if(ck==0){

break;

}

}

flag=0;

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

if(dis[v[i]] > dis[u[i]]+w[i]){

flag=1;

}

}

if(flag== 1){

cout << "此图包含有负权回路\n";

}else{

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

cout << dis[i] << ' ' ;

}

}

}

## 翻转

#include <iostream>

#include <cstdio>

#include <cstring>

using namespace std;

const int N = 16;

int g[N][N], t[N][N], f[N][N];

int cnt, n, m;

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

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

void flip(int i, int j)//翻转

{

++cnt, f[i][j] = 1;//步数加1，记录翻转了哪个瓷砖

t[i][j] = !t[i][j];//首先翻转自己

for(int k = 0; k < 4; ++k)//向四个方向寻找，找到就翻转，这里使用了异或

if(i + x[k] > -1 && j + y[k] > -1)

t[i + x[k]][j + y[k]] ^= 1;

}

bool ok(int k)//对于第一行的每一种情况，判断是否能够产生最终的结果

{

cnt = 0;//初始化步数

memcpy(t, g, sizeof(t));//初始化临时数组，作为原始数组的副本

for(int j = 0; j < m; ++j)//这里采用了二进制压缩，例如，j从0到3，那么1 << (m - 1 - j)的二进制就是1000,0100,0010,0001

if(k & (1 << (m - 1 - j)))//对于k的每一个取值，如1010，找到不为0的列，因为只需要翻转1就可以了，用到了与运算

flip(0, j);//如果某一列不为0，就翻转第一行的这个位置

for(int i = 1; i < n; ++i)//当第一行全部翻转完了，原来为1的位置肯定是0，原来是0的位置肯定是1，这就需要第二行来解决这些为1位置，以此类推

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

if(t[i - 1][j]) flip(i, j);//如果该列上一个位置是1，那么这个位置需要翻，否则不需要翻

for(int j = 0; j < m; ++j)//因为每一行的1都可以由下一行搞定，但是最后一行没有下一行，所以只需要考察最后一行最后是不是全0就可以了

if(t[n - 1][j]) return false;

return true;

}

int main()

{

int ans, p;

while(~scanf("%d%d", &n, &m))

{

for(int i = 0; i < n; ++i)//数据输入

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

scanf("%d", &g[i][j]);

ans = n \* m + 1, p = -1;//初始化

for(int i = 0; i < (1 << m); ++i)//i表示一个二进制数，用来枚举第一行的各种不同翻法，如0001就是只翻最后一个

if(ok(i) && cnt < ans) //如果找到一种可能并且所用的步数更少的话，记下这种翻法

ans = cnt, p = i;

memset(f, 0, sizeof(f));

if(p >= 0)//最后找到的就是最少的翻法，模拟一遍，然后输出

{

ok(p);

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

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

printf("%d%c", f[i][j], j < m - 1 ? ' ' : '\n');

}

else puts("IMPOSSIBLE");

}

return 0;

}

## 数组线段树

const int MAXM=50000;　　　　　　　　　　//定义 MAXM 为线段最大长度

int a[MAXM+5],st[(MAXM<<2)+5];　　　　// a 数组为 main 函数中读入的内容，st 数组为需要查询的数的信息（如和、最值等），树的空间大小为线段最大长度的四倍

void build(int o,int l,int r){　　　　//传入的参数为 o:当前需要建立的结点；l：当前需要建立的左端点；r：当前需要建立的右端点

if(l==r)st[o]=a[l];　　　　　　//当左端点等于右端点即建立叶子结点时，直接给数组信息赋值

else{

int m=l+((r-l)>>1);　　　　　　// m 为中间点，左儿子结点为 [l,m] ，右儿子结点为 [m+1,r]；

build(o<<1,l,m);　　　　　　　　//构建左儿子结点

build((o<<1)|1,m+1,r);　　　　　//构建右儿子结点

st[o]=st[o<<1]+st[(o<<1)|1];　　//递归返回时用儿子结点更新父节点，此处可进行更新最大值、最小值、区间和等操作

}

}

{　　　　　　　　　　　　　　　　　　　　　　　//在 main 函数中的语句

build(1,1,n);

}

void update(int o,int l,int r,int ind,int ans){　　//o、l、r为当前更新到的结点、左右端点，ind为需要修改的叶子结点左端点，ans为需要修改成的值；

if(l==r){　　　　　　　　　　　　　　　　　　　　　　//若当前更新点的左右端点相等即到叶子结点时，直接更新信息并返回

st[o]=ans;

return;

}

int m=l+((r-l)>>1);

if(ind<=m){　　　　　　　　　　　　　　　　　　　　　　//若需要更新的叶子结点在当前结点的左儿子结点的范围内，则递归更新左儿子结点，否则更新右儿子结点

update(o<<1,l,m,ind,ans);

}

else{

update((o<<1)|1,m+1,r,ind,ans);

}

st[o]=max(st[o<<1],st[(o<<1)|1]);//递归回之后用儿子结点更新父节点（此处是区间最大值）

}

{　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　//在main函数中的语句

update(1,1,n,ind,ans);

}

int query(int o,int l,int r,int ql,int qr){　　　　　　//ql、qr为需要查询的区间左右端点

if(ql>r||qr<l) return -1;　　　　　　　　　　　　　　//若当前结点和需要查找的区间不相交，则返回一个对于区间查询无关的值（如求和时返回0，求最大值时返回-1等）

if(ql<=l&&qr>=r) return st[o];　　　　　　　　//若当前结点的区间被需要查询的区间覆盖，则返回当前结点的信息

int m=l+((r-l)>>1);

int p1=query(o<<1,l,m,ql,qr),p2=query((o<<1)|1,m+1,r,ql,qr);　　//p1为查询左儿子结点得到的信息，p2为查询右儿子结点得到的信息

return max(p1,p2);　　　　//综合两个儿子结点的信息并返回

}

{　　　　//main函数中的语句

printf("%d\n",query(1,1,n,a,b));

}

void pushup(int o){　　　　　　　　　　//pushup函数，该函数本身是将当前结点用左右子节点的信息更新，此处求区间和，用于update中将结点信息传递完返回后更新父节点

st[o]=st[o<<1]+st[o<<1|1];

}

void pushdown(int o,int l,int r){　　//pushdown函数，将o结点的信息传递到左右子节点上

if(add[o]){　　　　　　　　　　　　　//当父节点有更新信息时才向下传递信息

add[o<<1]+=add[o];　　　　　　//左右儿子结点均加上父节点的更新值

add[o<<1|1]+=add[o];

int m=l+((r-l)>>1);

st[o<<1]+=add[o]\*(m-l+1);　　//左右儿子结点均按照需要加的值总和更新结点信息

st[o<<1|1]+=add[o]\*(r-m);

add[o]=0;　　　　　　　　　　　　　　　　//信息传递完之后就可以将父节点的更新信息删除

}

}

void update(int o,int l,int r,int ql,int qr,int addv){　　//ql、qr为需要更新的区间左右端点，addv为需要增加的值

if(ql<=l&&qr>=r){　　　　　　　　　　　　　　　　　　　　　　//与单点更新一样，当当前结点被需要更新的区间覆盖时

add[o]+=addv;　　　　　　　　　　　　　　　　　　　　　　//更新该结点的所需更新信息

st[o]+=addv\*(r-l+1);　　　　　　　　　　　　　　　　//更新该结点信息

return;　　　　　　　　　　　　　　　　　　　　//根据lazy思想，由于不需要遍历到下层结点，因此不需要继续向下更新，直接返回

}

pushdown(o,l,r);　　　　　　　　　　　　　　　　　　//将当前结点的所需更新信息传递到下一层（其左右儿子结点）

int m=l+((r-l)>>1);

if(ql<=m)update(o<<1,l,m,ql,qr,addv);　　　　　//当需更新区间在当前结点的左儿子结点内，则更新左儿子结点

if(qr>=m+1)update(o<<1|1,m+1,r,ql,qr,addv);　　　//当需更新区间在当前结点的右儿子结点内，则更新右儿子结点

pushup(o);　　　　　　　　　　　　　　　　　　//递归回上层时一步一步更新回父节点

}

ll query(int o,int l,int r,int ql,int qr){　　　　//ql、qr为需要查询的区间

if(ql<=l&&qr>=r) return st[o];　　　　　　//若当前结点覆盖区间即为需要查询的区间，则直接返回当前结点的信息

pushdown(o,l,r);　　　　　　　　　　　　　　　　　　//将当前结点的更新信息传递给其左右子节点

int m=l+((r-l)>>1);

ll ans=0;　　　　　　　　　　　　　　　　　　　　　　//所需查询的结果

if(ql<=m)ans+=query(o<<1,l,m,ql,qr);　　　　　//若所需查询的区间与当前结点的左子节点有交集，则结果加上查询其左子节点的结果

if(qr>=m+1)ans+=query(o<<1|1,m+1,r,ql,qr);　//若所需查询的区间与当前结点的右子节点有交集，则结果加上查询其右子节点的结果

　　 return ans;

}

void pushup(int o){

st[o]=st[o<<1]+st[o<<1|1];

}

void pushdown(int o,int l,int r){　　//pushdown和区间加值不同，改值时修改结点信息只需要对修改后的信息求和即可，不用加上原信息

if(change[o]){

int c=change[o];

change[o<<1]=c;

change[o<<1|1]=c;

int m=l+((r-l)>>1);

st[o<<1]=(m-l+1)\*c;

st[o<<1|1]=(r-m)\*c;

change[o]=0;

}

}

void update(int o,int l,int r,int ql,int qr,int c){

if(ql<=l&&qr>=r){　　　　　　　　　//同样更新结点信息和区间加值不同

change[o]=c;

st[o]=(r-l+1)\*c;

return;

}

pushdown(o,l,r);

int m=l+((r-l)>>1);

if(ql<=m)update(o<<1,l,m,ql,qr,c);

if(qr>=m+1)update(o<<1|1,m+1,r,ql,qr,c);

pushup(o);

}

int query(int o,int l,int r,int ql,int qr){

if(ql<=l&&qr>=r) return st[o];

pushdown(o,l,r);

int m=l+((r-l)>>1);

int ans=0;

if(ql<=m)ans+=query(o<<1,l,m,ql,qr);

if(qr>=m+1)ans+=query(o<<1|1,m+1,r,ql,qr);

return ans;

}

# Math

## dancing links

#include <bits/stdc++.h>

using namespace std;

const int maxnode = 5e5 + 10;

const int MAXM = 10010;

const int MAXN = 510;

struct DLX

{

int n, m, size;

int U[maxnode], D[maxnode], R[maxnode], L[maxnode], Row[maxnode], Col[maxnode];

int H[MAXM], S[MAXM];

int ansd;

void init(int \_n, int \_m)

{

n = \_n;

m = \_m;

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

{

S[i] = 0;

U[i] = D[i] = i;

L[i] = i - 1;

R[i] = i + 1;

}

R[m] = 0;

L[0] = m;

size = m;

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

{

H[i] = -1;

}

}

void Link(int r, int c)

{

++S[Col[++size] = c];

Row[size] = r;

D[size] = D[c];

U[D[c]] = size;

U[size] = c;

D[c] = size;

if (H[r] < 0) H[r] = L[size] = R[size] = size;

else {

R[size] = R[H[r]];

L[R[H[r]]] = size;

L[size] = H[r];

R[H[r]] = size;

}

}

void remove (int c) {

L[R[c]] = L[c], R[L[c]] = R[c];

for (int i = D[c]; i != c; i = D[i]) {

for (int j = R[i]; j != i; j = R[j]) {

U[D[j]] = U[j];

D[U[j]] = D[j];

--S[Col[j]];

}

}

}

void resume (int c) {

for (int i = U[c]; i != c; i = U[i])

for (int j = L[i]; j != i; j = L[j])

++S[Col[U[D[j]] = D[U[j]] = j]];

L[R[c]] = R[L[c]] = c;

}

void Dance (int d) {

//jian zhi

if (ansd != -1 && ansd <= d) return;

if (R[0] == 0) {

if (ansd == -1) ansd = d;

else if (d < ansd) ansd = d;

return ;

}

int c = R[0];

for (int i = R[0]; i != 0; i = R[i])

if (S[i] < S[c])

c = i;

remove (c);

for (int i = D[c]; i != c; i = D[i]) {

for (int j = R[i]; j != i; j = R[j]) remove (Col[j]);

Dance (d + 1);

for (int j = L[i]; j != i; j = L[j]) resume (Col[j]);

}

resume (c);

}

};

DLX g;

int main()

{

int t, n, m, p;

cin >> t;

while (t--) {

cin >> n >> m >> p;

g.init(p, n\*m);

int x1, x2, y1, y2;

for (int k = 1; k <= p; k++) {

cin >> x1 >> y1 >> x2 >> y2;

for (int i = x1 + 1; i <= x2; i++) {

for (int j = y1 + 1; j <= y2; j++) {

g.Link (k, j + (i - 1)\*m);//01矩阵，把给你的矩阵转换为线性

}

}

}

g.ansd = -1;

g.Dance (0);

cout << g.ansd << "\n";

}

return 0;

}

## Bagage

//0-1背包

#include <iostream>

#include <cstring>

#define MAXN 10000

using namespace std;

int dp[MAXN];

int w[MAXN] = {0, 2, 1, 3, 2};

int v[MAXN] = {0, 3, 2, 4, 2};

int W = 5, n = 4;

int solve(int n, int W) {

memset(dp, 0, sizeof(dp));

for (int i = 1; i <= n; i++) { // i从1开始，递增

for (int j = W; j >= 0; j--) { // j按递减顺序填表

if (j < w[i]) {

dp[j] = dp[j];

}

else {

dp[j] = max(dp[j], dp[j-w[i]] + v[i]);

}

}

}

return dp[W];

}

int main() {

cout << solve(n, W) << endl;

return 0;

}

//完全背包

#include <iostream>

#include <cstring>

#define MAXN 1000

using namespace std;

int dp[MAXN];

int w[MAXN] = {0, 3, 4, 2};

int v[MAXN] = {0, 4, 5, 3};

int W = 7, n = 3;

int solve(int n, int W) {

memset(dp, 0, sizeof(dp));

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

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

if (j < w[i]) {

dp[j] = dp[j];

}

else {

dp[j] = max(dp[j], dp[j-w[i]] + v[i]);

}

}

}

return dp[W];

}

int main() {

cout << solve(n, W) << endl; // 10

return 0;

}

//完全背包

#include <iostream>

#include <deque>

#include <algorithm>

using namespace std;

struct Pack

{

int sum, cost;

Pack(int s, int c) : sum (s), cost(c) {}

};

const int Maxv = 1001;

deque <Pack> Q;

int N, V, F[Maxv];

int main()

{

cin >> N >> V;

for (int i = 1, p, w, c; i <= N; i ++)

{

cin >> p >> w >> c; p = min(p, V / w);

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

{

Q.clear();

for (int k = 0; k <= (V - j) / w; k ++)

{

int y = F[k \* w + j] - k \* c;

while (Q.size() && Q.back().cost <= y) Q.pop\_back();

Q.push\_back(Pack(k, y));

if (Q.front().sum < k - c) Q.pop\_front();

F[k \* w + j] = Q.front().cost + k \* c;

}

}

}

cout << F[V] << endl;

return 0;

}

//背包

#include <stdio.h>

#include <algorithm>

#include <string.h>

using namespace std;

const int MAX=100000;

int dp[MAX];

int c[MAX],w[MAX];

int v;

void ZeroOnePack(int cost,int wei)//01

{

int i;

for(i = v;i>=cost;i--)

{

dp[i] = max(dp[i],dp[i-cost]+wei);

}

}

void CompletePack(int cost,int wei)//完全

{

int i;

for(i = cost;i<=v;i++)

{

dp[i] = max(dp[i],dp[i-cost]+wei);

}

}

void MultiplePack(int cost,int wei,int cnt)//多重

{

if(v<=cnt\*cost)//如果总容量比这个物品的容量要小，那么这个物品可以直到取完，相当于完全背包

{

CompletePack(cost,wei);

return ;

}

else//否则就将多重背包转化为01背包

{

int k = 1;

while(k<=cnt)

{

ZeroOnePack(k\*cost,k\*wei);

cnt = cnt-k;

k = 2\*k;

}

ZeroOnePack(cnt\*cost,cnt\*wei);

}

}

int main()

{

int n;

while(~scanf("%d%d",&n,&v),n+v)

{

int i;

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

scanf("%d",&c[i]);

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

scanf("%d",&w[i]);

memset(dp,0,sizeof(dp));

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

{

MultiplePack(c[i],c[i],w[i]);

}

int sum = 0;

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

{

if(dp[i]==i)

{

sum++;

}

}

printf("%d\n",sum);

}

return 0;

}

## 01背包--装满背包的方案总数

#include<iostream>

using namespace std;

#define Size 1111

int MethodTable[Size];

int Max(int x,int y)

{

return x>y?x:y;

}

int Package01\_FullOfPackage(int Weight[], int nLen, int nCapacity)

{

MethodTable[0]= 1;//初始化

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

{

for(int j = nCapacity; j >=Weight[i];j--)

{

if(j >= Weight[i])

MethodTable[j] += MethodTable[j-Weight[i]];

}

}

cout << "MethodTable:" << endl;

return MethodTable[nCapacity];

}

int main()

{

int Weight[Size];

int nCapacity;//空间

int n\_goods;//数量

cin>>n\_goods>>nCapacity;

for(int k=1;k<=n\_goods;k++)

cin>>Weight[k];

cout << "AllCount:" << Package01\_FullOfPackage(Weight,n\_goods,nCapacity) << endl;

return 0;

}

## 完全背包--装满背包的方案总数

#define Size 1111

int Package02\_FullOfPackage\_Compress(int Weight[], int nLen, int nCapacity)//nCapacity是背包最大容量。需要保证比Size小。

{

int MethodTable[Size];

memset(MethodTable,0,sizeof(MethodTable));

//initiallize all MethodTable[0] with 1

MethodTable[0] = 1;

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

{

for(int j = Weight[i]; j <= nCapacity; j++) //从第i件物品的重量开始遍历到最大重量。

{

if(j >= Weight[i])

MethodTable[j] += MethodTable[j-Weight[i]];

}

}

int nRet = MethodTable[nCapacity];

delete [] MethodTable;

return nRet;

}

## 大数

#include<bits/stdc++.h>

#include<algorithm>

using namespace std;

#define MAXN 9999

#define MAXSIZE 10

#define DLEN 4

class BigNum

{

private:

int a[500]; //可以控制大数的位数

int len; //大数长度

public:

BigNum(){ len = 1;memset(a,0,sizeof(a)); } //构造函数

BigNum(const int); //将一个int类型的变量转化为大数

BigNum(const char\*); //将一个字符串类型的变量转化为大数

BigNum(const BigNum &); //拷贝构造函数

BigNum &operator=(const BigNum &); //重载赋值运算符，大数之间进行赋值运算

friend istream& operator>>(istream&, BigNum&); //重载输入运算符

friend ostream& operator<<(ostream&, BigNum&); //重载输出运算符

BigNum operator+(const BigNum &) const; //重载加法运算符，两个大数之间的相加运算

BigNum operator-(const BigNum &) const; //重载减法运算符，两个大数之间的相减运算

BigNum operator\*(const BigNum &) const; //重载乘法运算符，两个大数之间的相乘运算

BigNum operator/(const int &) const; //重载除法运算符，大数对一个整数进行相除运算

BigNum operator^(const int &) const; //大数的n次方运算

int operator%(const int &) const; //大数对一个int类型的变量进行取模运算

bool operator>(const BigNum & T)const; //大数和另一个大数的大小比较

bool operator>(const int & t)const; //大数和一个int类型的变量的大小比较

void print(); //输出大数

};

BigNum::BigNum(const int b) //将一个int类型的变量转化为大数

{

int c,d = b;

len = 0;

memset(a,0,sizeof(a));

while(d > MAXN)

{

c = d - (d / (MAXN + 1)) \* (MAXN + 1);

d = d / (MAXN + 1);

a[len++] = c;

}

a[len++] = d;

}

BigNum::BigNum(const char\*s) //将一个字符串类型的变量转化为大数

{

int t,k,index,l,i;

memset(a,0,sizeof(a));

l=strlen(s);

len=l/DLEN;

if(l%DLEN)

len++;

index=0;

for(i=l-1;i>=0;i-=DLEN)

{

t=0;

k=i-DLEN+1;

if(k<0)

k=0;

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

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

a[index++]=t;

}

}

BigNum::BigNum(const BigNum & T) : len(T.len) //拷贝构造函数

{

int i;

memset(a,0,sizeof(a));

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

a[i] = T.a[i];

}

BigNum & BigNum::operator=(const BigNum & n) //重载赋值运算符，大数之间进行赋值运算

{

int i;

len = n.len;

memset(a,0,sizeof(a));

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

a[i] = n.a[i];

return \*this;

}

istream& operator>>(istream & in, BigNum & b) //重载输入运算符

{

char ch[MAXSIZE\*4];

int i = -1;

in>>ch;

int l=strlen(ch);

int count=0,sum=0;

for(i=l-1;i>=0;)

{

sum = 0;

int t=1;

for(int j=0;j<4&&i>=0;j++,i--,t\*=10)

{

sum+=(ch[i]-'0')\*t;

}

b.a[count]=sum;

count++;

}

b.len =count++;

return in;

}

ostream& operator<<(ostream& out, BigNum& b) //重载输出运算符

{

int i;

cout << b.a[b.len - 1];

for(i = b.len - 2 ; i >= 0 ; i--)

{

cout.width(DLEN);

cout.fill('0');

cout << b.a[i];

}

return out;

}

BigNum BigNum::operator+(const BigNum & T) const //两个大数之间的相加运算

{

BigNum t(\*this);

int i,big; //位数

big = T.len > len ? T.len : len;

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

{

t.a[i] +=T.a[i];

if(t.a[i] > MAXN)

{

t.a[i + 1]++;

t.a[i] -=MAXN+1;

}

}

if(t.a[big] != 0)

t.len = big + 1;

else

t.len = big;

return t;

}

BigNum BigNum::operator-(const BigNum & T) const //两个大数之间的相减运算

{

int i,j,big;

bool flag;

BigNum t1,t2;

if(\*this>T)

{

t1=\*this;

t2=T;

flag=0;

}

else

{

t1=T;

t2=\*this;

flag=1;

}

big=t1.len;

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

{

if(t1.a[i] < t2.a[i])

{

j = i + 1;

while(t1.a[j] == 0)

j++;

t1.a[j--]--;

while(j > i)

t1.a[j--] += MAXN;

t1.a[i] += MAXN + 1 - t2.a[i];

}

else

t1.a[i] -= t2.a[i];

}

t1.len = big;

while(t1.a[t1.len - 1] == 0 && t1.len > 1)

{

t1.len--;

big--;

}

if(flag)

t1.a[big-1]=0-t1.a[big-1];

return t1;

}

BigNum BigNum::operator\*(const BigNum & T) const //两个大数之间的相乘运算

{

BigNum ret;

int i,j,up;

int temp,temp1;

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

{

up = 0;

for(j = 0 ; j < T.len ; j++)

{

temp = a[i] \* T.a[j] + ret.a[i + j] + up;

if(temp > MAXN)

{

temp1 = temp - temp / (MAXN + 1) \* (MAXN + 1);

up = temp / (MAXN + 1);

ret.a[i + j] = temp1;

}

else

{

up = 0;

ret.a[i + j] = temp;

}

}

if(up != 0)

ret.a[i + j] = up;

}

ret.len = i + j;

while(ret.a[ret.len - 1] == 0 && ret.len > 1)

ret.len--;

return ret;

}

BigNum BigNum::operator/(const int & b) const //大数对一个整数进行相除运算

{

BigNum ret;

int i,down = 0;

for(i = len - 1 ; i >= 0 ; i--)

{

ret.a[i] = (a[i] + down \* (MAXN + 1)) / b;

down = a[i] + down \* (MAXN + 1) - ret.a[i] \* b;

}

ret.len = len;

while(ret.a[ret.len - 1] == 0 && ret.len > 1)

ret.len--;

return ret;

}

int BigNum::operator %(const int & b) const //大数对一个int类型的变量进行取模运算

{

int i,d=0;

for (i = len-1; i>=0; i--)

{

d = ((d \* (MAXN+1))% b + a[i])% b;

}

return d;

}

BigNum BigNum::operator^(const int & n) const //大数的n次方运算

{

BigNum t,ret(1);

int i;

if(n<0)

exit(-1);

if(n==0)

return 1;

if(n==1)

return \*this;

int m=n;

while(m>1)

{

t=\*this;

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

{

t=t\*t;

}

m-=i;

ret=ret\*t;

if(m==1)

ret=ret\*(\*this);

}

return ret;

}

bool BigNum::operator>(const BigNum & T) const //大数和另一个大数的大小比较

{

int ln;

if(len > T.len)

return true;

else if(len == T.len)

{

ln = len - 1;

while(a[ln] == T.a[ln] && ln >= 0)

ln--;

if(ln >= 0 && a[ln] > T.a[ln])

return true;

else

return false;

}

else

return false;

}

bool BigNum::operator >(const int & t) const //大数和一个int类型的变量的大小比较

{

BigNum b(t);

return \*this>b;

}

void BigNum::print() //输出大数

{

int i;

cout << a[len - 1];

for(i = len - 2 ; i >= 0 ; i--)

{

cout.width(DLEN);

cout.fill('0');

cout << a[i];

}

cout << endl;

}

int main()

{

BigNum a1;

BigNum b1;

BigNum sum;

string a,b;

while(cin>>a>>b){

const char \*aa=a.c\_str();

const char \*bb=b.c\_str();

a1=aa;

b1=bb;

sum=a1+b1;

cout<<sum<<endl;

}

return 0;

}

## 多边形面积

struct node

{

double x, y;

};

double area(node\* lists, int n)

{

step = lists[0].x \* list[n - 1].y - list[n - 1].x \* lists[0].y;

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

{

area += list[i - 1].x \* list[i].y - list[i].x \* list[i - 1].y;

}

area = fabs(area) - step;

area /= 2;

return area;

}

## 高斯消元法(浮点型)

#include<bits/stdc++.h>

#define rg register

#define il inline

#define maxn 500005

#define eps 1e-8

#define ll long long

using namespace std;

bool vis[maxn];

double a[300][300],b[maxn],res[maxn];

bool Gauss(int m,int n){

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

if (fabs(a[i][i]) < eps){//若该项为0，则进行 行交换

int j = i + 1;

while (fabs(a[j][i]) < eps && j<=m) ++j;//在当前列中找到一个非零项

if (j > m) return 0;//如果没有找到，说明没有唯一解，直接return

if (j<=m) {swap(a[i],a[j]);swap(res[i],res[j]);}//进行 行交换

}

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

if (fabs (a[j][i]) > eps){//只操作该列非0元素所在的行

double chu = a[j][i] / a[i][i];//简单的消元

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

a[j][k] -= (a[i][k] \* chu);

}

res[j] -= (res[i] \* chu);

}

}

}

for (int i = n;i>=1;--i){//暴力回代

double x = res[i];

for (int j=n;j>i;--j){

x -= a[i][j] \* b[j];//减去已知值

}

b[i] = x / a[i][i];//得到未知数

}

return 1;

}

int main(){

rg int n,m;

scanf("%d",&m);

n = m;

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

for (rg int j=1;j<=n;++j){

scanf("%lf",&a[i][j]);

}

scanf("%lf",&res[i]);//这里为了结果清晰就没有增广了

}

if (!Gauss(m,n)) {cout<<"No Solution";return 0;}

for (rg int i=1;i<=n;++i) printf("%.2f\n",b[i]);

return 0;

}

## 高斯消元法(整数)

#include<stdio.h>

#include<algorithm>

#include<iostream>

#include<string.h>

#include<math.h>

using namespace std;

const int MAXN=50;

int a[MAXN][MAXN];//增广矩阵

int x[MAXN];//解集

bool free\_x[MAXN];//标记是否是不确定的变元

int gcd(int a,int b){

if(b == 0) return a; else return gcd(b,a%b);

}

inline int lcm(int a,int b){

return a/gcd(a,b)\*b;//先除后乘防溢出

}

// 高斯消元法解方程组(Gauss-Jordan elimination).(-2表示有浮点数解，但无整数解，

//-1表示无解，0表示唯一解，大于0表示无穷解，并返回自由变元的个数)

//有equ个方程，var个变元。增广矩阵行数为equ,分别为0到equ-1,列数为var+1,分别为0到var.

int Gauss(int equ,int var){

int i,j,k;

int max\_r;// 当前这列绝对值最大的行.

int col;//当前处理的列

int ta,tb;

int LCM;

int temp;

int free\_x\_num;

int free\_index;

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

x[i]=0;

free\_x[i]=true;

}

//转换为阶梯阵.

col=0; // 当前处理的列

for(k = 0;k < equ && col < var;k++,col++){// 枚举当前处理的行.

// 找到该col列元素绝对值最大的那行与第k行交换.(为了在除法时减小误差)

max\_r=k;

for(i=k+1;i<equ;i++){

if(abs(a[i][col])>abs(a[max\_r][col])) max\_r=i;

}

if(max\_r!=k){// 与第k行交换.

for(j=k;j<var+1;j++) swap(a[k][j],a[max\_r][j]);

}

if(a[k][col]==0){// 说明该col列第k行以下全是0了，则处理当前行的下一列.

k--;

continue;

}

for(i=k+1;i<equ;i++){// 枚举要删去的行.

if(a[i][col]!=0){

LCM = lcm(abs(a[i][col]),abs(a[k][col]));

ta = LCM/abs(a[i][col]);

tb = LCM/abs(a[k][col]);

if(a[i][col]\*a[k][col]<0)tb=-tb;//异号的情况是相加

for(j=col;j<var+1;j++){

a[i][j] = a[i][j]\*ta-a[k][j]\*tb;

}

}

}

}

// 1. 无解的情况: 化简的增广阵中存在(0, 0, ..., a)这样的行(a != 0).

for (i = k; i < equ; i++){ // 对于无穷解来说，如果要判断哪些是自由变元，那么初等行变换中的交换就会影响，则要记录交换.

if (a[i][col] != 0) return -1;

}

// 2. 无穷解的情况: 在var \* (var + 1)的增广阵中出现(0, 0, ..., 0)这样的行，即说明没有形成严格的上三角阵.

// 且出现的行数即为自由变元的个数.

if (k < var){

return var - k; // 自由变元有var - k个.

}

// 3. 唯一解的情况: 在var \* (var + 1)的增广阵中形成严格的上三角阵.

// 计算出Xn-1, Xn-2 ... X0.

for (i = var - 1; i >= 0; i--){

temp = a[i][var];

for (j = i + 1; j < var; j++){

if (a[i][j] != 0) temp -= a[i][j] \* x[j];

}

if (temp % a[i][i] != 0) return -2; // 说明有浮点数解，但无整数解.

x[i] = temp / a[i][i];

}

return 0;

}

int main(void){

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

// freopen("out.txt","w",stdout);

int i, j;

int equ,var;

while (scanf("%d %d", &equ, &var) != EOF){

memset(a, 0, sizeof(a));

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

for (j = 0; j < var + 1; j++){

scanf("%d", &a[i][j]);

}

}

int free\_num = Gauss(equ,var);

if (free\_num == -1) printf("无解!\n");

else if (free\_num == -2){

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

printf("%.2lf\n",x[i]);

}

}

else if (free\_num > 0){

printf("无穷多解! 自由变元个数为%d\n", free\_num);

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

if (free\_x[i]) printf("x%d 是不确定的\n", i + 1);

else printf("x%d: %d\n", i + 1, x[i]);

}

}else{

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

printf("x%d: %d\n", i + 1, x[i]);

}

}

printf("\n");

}

return 0;

}

## 扩展gcd（中国剩余定理）

# include <bits/stdc++.h>

using namespace std;

typedef long long ll;

ll a, b, c, d, x, y, gd;

ll ex\_gcd(ll a, ll b)

{

if(b == 0)

{

x = 1;

y = 0;

return a;

}

gd = ex\_gcd(b, a%b);

ll t = x;

x = y;

y = t-(a/b)\*y;

return gd;

}

int main()

{

ll A, B, C, D, t, tmp=0;

scanf("%lld%lld%lld%lld",&A,&B,&C,&D);

a = A;

b = -C;

c = D-B;

gd = ex\_gcd(a, b);

if(c % gd != 0)

return 0\*puts("-1");

x = x\*c/gd;

y = y\*c/gd;

ll k = b/gd;

x = (x%k+k)%k;

if(x < 0) x += abs(k);

k = abs(k);

ll l = 0, r = 1e9;

while (l <= r) {

ll mid = (l + r) / 2;

if (x + mid \* k >= 1e9) {

r = mid - 1;

continue;

}

if ((c - a \* (x + mid \* k)) / b >= 0) {

r = mid - 1;

tmp = mid;

} else {

l = mid + 1;

}

}

printf("%lld\n",B+A\*(x+k\*tmp));

return 0;

}

## 欧拉筛

int prime[maxn];

int visit[maxn];

void Prime(){

mem(visit,0);

mem(prime, 0);

for (int i = 2;i <= maxn; i++) {

cout<<" i = "<<i<<endl;

if (!visit[i]) {

prime[++prime[0]] = i; //纪录素数， 这个prime[0] 相当于 cnt，用来计数

}

for (int j = 1; j <=prime[0] && i\*prime[j] <= maxn; j++) {

// cout<<" j = "<<j<<" prime["<<j<<"]"<<" = "<<prime[j]<<" i\*prime[j] = "<<i\*prime[j]<<endl;

visit[i\*prime[j]] = 1;

if (i % prime[j] == 0) {

break;

}

}

}

}

## 判断两条线段相交

#include<bits/stdc++.h>

#include<algorithm>

using namespace std;

const double eps=1e-10;

struct point{

double x,y;

};

double min(double a, double b) {

return a < b ? a : b;

}

double max(double a, double b) {

return a < b ? b : a;

};

bool inter(point a, point b, point c, point d){

if (min(a.x, b.x) > max(c.x, d.x) || min(a.y, b.y) > max(c.y, d.y) || min(c.x, d.x) > max(a.x, b.x) || min(c.y, d.y) > max(a.y, b.y)){

return 0;

}

double h,i,j,k;

h = (b.x - a.x) \* (c.y - a.y) - (b.y - a.y) \* (c.x - a.x);

i = (b.x - a.x) \* (d.y - a.y) - (b.y - a.y) \* (d.x - a.x);

j = (d.x - c.x) \* (a.y - c.y) - (d.y - c.y) \* (a.x - c.x);

k = (d.x - c.x) \* (b.y - c.y) - (d.y - c.y) \* (b.x - c.x);

return h \* i <= eps && j \* k <= eps;

}

## 数学公式

#include<bits/stdc++.h>

using namespace std;

typedef long double ld;

const ld pi(acos(-1));

int main(void){

ld v,h,r,;

v = pi\*h\*h\*(r-h/3);//h是球缺的高，r是球的半径

}

## 凸包（二维）

#include<stdio.h>

#include<limits.h>

#include<string.h>

typedef struct

{

int x;

int y;

int temp; /\*若这点在凸包上,则temp==1,否则temp==0\*/

}Point;

Point s[105];

int n;

void Sechup(int a, int b);

void Sechdown(int a, int b);

int main(void)

{

int T, i, a, b, ax, bx;

scanf("%d", &T);

while(T--)

{

memset(s, 0, sizeof(s));

ax = INT\_MAX;

bx = INT\_MIN;

scanf("%d", &n);

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

{

scanf("%d%d", &s[i].x, &s[i].y);

if(s[i].x<ax)

ax = s[i].x, a = i;

if(s[i].x>bx)

bx = s[i].x, b = i;

}

s[a].temp = s[b].temp = 1;

Sechup(a, b); /\*上包递归\*/

Sechdown(a, b); /\*下包递归\*/

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

{

if(s[i].temp==1)

printf("%d %d\n", s[i].x, s[i].y);

}

}

return 0;

}

/\*若向量叉乘为负,说明点在直线下面,否则在直线上面(参照直线方向为从左到右)\*/

void Sechup(int a, int b)

{

int i, max, c;

max = 0;

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

{

if((s[b].x-s[a].x)\*(s[i].y-s[a].y)-(s[b].y-s[a].y)\*(s[i].x-s[a].x)>max)

{

max = (s[b].x-s[a].x)\*(s[i].y-s[a].y)-(s[b].y-s[a].y)\*(s[i].x-s[a].x);

c = i; /\*用向量叉乘求三点三角形面积,使三角形面积最大的那个(非直线上的)点便是离参照直线最远的点Pmax\*/

}

}

if(max!=0)/\*参照直线上方有点\*/

{

s[c].temp = 1;

Sechup(a, c);

Sechup(c, b);

}

}

void Sechdown(int a, int b)

{

int i, max, c;

max = 0;

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

{

if((s[b].x-s[a].x)\*(s[i].y-s[a].y)-(s[b].y-s[a].y)\*(s[i].x-s[a].x)<max)

{

max = (s[b].x-s[a].x)\*(s[i].y-s[a].y)-(s[b].y-s[a].y)\*(s[i].x-s[a].x);

c = i;

}

}

if(max!=0)

{

s[c].temp = 1;

Sechdown(a, c);

Sechdown(c, b);

}

}

## 凸包（三维）

#include <cstring>

#include <cstdio>

#include <cmath>

#include <algorithm>

#include<iostream>

using namespace std;

#define inf 0x7fffffff

#define max(a,b) (a>b?a:b)

#define min(a,b) (a<b?a:b)

#define eps 1e-7

#define MAXV 505

//三维点

struct pt {

double x, y, z;

pt() {}

pt(double \_x, double \_y, double \_z): x(\_x), y(\_y), z(\_z) {}

pt operator - (const pt p1) {

return pt(x - p1.x, y - p1.y, z - p1.z);

}

pt operator \* (pt p) {

return pt(y\*p.z-z\*p.y, z\*p.x-x\*p.z, x\*p.y-y\*p.x); //叉乘

}

double operator ^ (pt p) {

return x\*p.x+y\*p.y+z\*p.z; //点乘

}

};

struct \_3DCH {

struct fac {

int a, b, c; //表示凸包一个面上三个点的编号

bool ok; //表示该面是否属于最终凸包中的面

};

int n; //初始点数

pt P[MAXV]; //初始点

int cnt; //凸包表面的三角形数

fac F[MAXV\*8]; //凸包表面的三角形

int to[MAXV][MAXV];

double vlen(pt a) {

return sqrt(a.x\*a.x+a.y\*a.y+a.z\*a.z); //向量长度

}

double area(pt a, pt b, pt c) {

return vlen((b-a)\*(c-a)); //三角形面积\*2

}

double volume(pt a, pt b, pt c, pt d) {

return (b-a)\*(c-a)^(d-a); //四面体有向体积\*6

}

//正：点在面同向

double ptof(pt &p, fac &f) {

pt m = P[f.b]-P[f.a], n = P[f.c]-P[f.a], t = p-P[f.a];

return (m \* n) ^ t;

}

void deal(int p, int a, int b) {

int f = to[a][b];

fac add;

if (F[f].ok) {

if (ptof(P[p], F[f]) > eps)

dfs(p, f);

else {

add.a = b, add.b = a, add.c = p, add.ok = 1;

to[p][b] = to[a][p] = to[b][a] = cnt;

F[cnt++] = add;

}

}

}

void dfs(int p, int cur) {

F[cur].ok = 0;

deal(p, F[cur].b, F[cur].a);

deal(p, F[cur].c, F[cur].b);

deal(p, F[cur].a, F[cur].c);

}

bool same(int s, int t) {

pt &a = P[F[s].a], &b = P[F[s].b], &c = P[F[s].c];

return fabs(volume(a, b, c, P[F[t].a])) < eps && fabs(volume(a, b, c, P[F[t].b])) < eps && fabs(volume(a, b, c, P[F[t].c])) < eps;

}

//构建三维凸包

void construct() {

cnt = 0;

if (n < 4)

return;

/\*\*\*\*\*\*\*\*\*此段是为了保证前四个点不公面，若已保证，可去掉\*\*\*\*\*\*\*\*/

bool sb = 1;

//使前两点不公点

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

if (vlen(P[0] - P[i]) > eps) {

swap(P[1], P[i]);

sb = 0;

break;

}

}

if (sb)return;

sb = 1;

//使前三点不公线

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

if (vlen((P[0] - P[1]) \* (P[1] - P[i])) > eps) {

swap(P[2], P[i]);

sb = 0;

break;

}

}

if (sb)return;

sb = 1;

//使前四点不共面

for (int i = 3; i < n; i++) {

if (fabs((P[0] - P[1]) \* (P[1] - P[2]) ^ (P[0] - P[i])) > eps) {

swap(P[3], P[i]);

sb = 0;

break;

}

}

if (sb)return;

/\*\*\*\*\*\*\*\*\*此段是为了保证前四个点不公面\*\*\*\*\*\*\*\*/

fac add;

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

add.a = (i+1)%4, add.b = (i+2)%4, add.c = (i+3)%4, add.ok = 1;

if (ptof(P[i], add) > 0)

swap(add.b, add.c);

to[add.a][add.b] = to[add.b][add.c] = to[add.c][add.a] = cnt;

F[cnt++] = add;

}

for (int i = 4; i < n; i++) {

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

if (F[j].ok && ptof(P[i], F[j]) > eps) {

dfs(i, j);

break;

}

}

}

int tmp = cnt;

cnt = 0;

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

if (F[i].ok) {

F[cnt++] = F[i];

}

}

}

//表面积

double area() {

double ret = 0.0;

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

ret += area(P[F[i].a], P[F[i].b], P[F[i].c]);

}

return ret / 2.0;

}

//体积

double volume() {

pt O(0, 0, 0);

double ret = 0.0;

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

ret += volume(O, P[F[i].a], P[F[i].b], P[F[i].c]);

}

return fabs(ret / 6.0);

}

//表面三角形数

int facetCnt\_tri() {

return cnt;

}

//表面多边形数

int facetCnt() {

int ans = 0;

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

bool nb = 1;

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

if (same(i, j)) {

nb = 0;

break;

}

}

ans += nb;

}

return ans;

}

};

\_3DCH hull;

int main() {

scanf("%d",&hull.n);

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

scanf("%lf%lf%lf",&hull.P[i].x,&hull.P[i].y,&hull.P[i].z);

hull.construct();

printf("%.3lf\n",hull.area());

return 0;

}

## 万进制

#include<iostream>

#include<iomanip>

using namespace std;

int main(void){

void factorial(int n);//阶乘函数||多次幂函数

int n;

while(cin >> n){

factorial(n);

}

}

void factorial(int n){

int a[10001];

int places,carry,i,j;

a[0]=1;

places=0;//当前数的总位数

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

carry=0;

for(j=0;j<=places;j++){

a[j]=a[j]\*i+carry;//如果是多次幂函数，将i改成数字即可

carry=a[j]/10000;

a[j]%=10000;

}

if(carry>0){

places++;

a[places]=carry;

}

}

/\*

\* 输出

\* 最高位原样输出

\* 其他位小于1000的，高位补0

\*/

cout <<a[places];

for(i=places-1;i>=0;i--){

cout << setw(4) << setfill('0') << a[i];

}

cout << endl;

## 因数个数

#include<iostream>

#include<cstring>

#include<string>

#include<cmath>

#include<algorithm>

#include<vector>

#include<queue>

using namespace std;

typedef long long ll;

const int inf = (int)1e9;

const int maxn = (int)1e6 + 5;

const ll mod = (ll)998244353;

int m, n;

int num[maxn] = { 0 };

int prim[maxn] = { 0 };

bool vis[maxn >> 1] = { false };

void initprime() {

m = 0;

for (int i = 2; i < maxn; i++) {

if (!vis[i]) {

prim[m++] = i;

}

for (int j = 0; j < maxn&&i\*prim[j] < maxn; j++) {

vis[i\*prim[j]] = 1;

if (i % prim[j] == 0) {

break;

}

}

}

}

int cnt[maxn], ans[maxn];

int cal(int x) {

int ed = int(sqrt(x));

int as = 1;

for (int i = 0; i < m&&prim[i] <= ed; i++) {

int cnt = 0;

while (x%prim[i] == 0) {

x /= prim[i];

cnt++;

}

ed = int(sqrt(x));

as \*= (cnt + 1);

if (as > 1000000) {

return -1;

}

}

if (x > 1) {

as \*= 2;

}if (as > 1000000) {

return -1;

}

return as;

}

void prime();

void solve() {

prime();

num[1] = 1;

num[2] = 2;

num[3] = 4;

for (int i = 5; i <= maxn; i++) {

int tmpe = i;

int res = 1;

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

int tmp = 0;

while (tmpe%prim[j] == 0) {

tmp++;

tmpe /= prim[j];

}

res \*= (tmp + 1);

if (tmpe == 1) {

break;

}

}

if (num[res] == 0) {

num[res] = tmpe;

}

}

}

void prime() {

int sq = (int)sqrt(maxn);

prim[0] = 2;

m = 1;

for (int i = 3; i < sq; i += 2) {

if (vis[i >> 1]) {

continue;

}

for (int j = i \* i; j <= maxn; j += (i << 1)) {

vis[j >> 1] = 1;

}

}

for (int i = 1; i <= (maxn >> 1); i++) {

if (!vis[i]) {

prim[m++] = i << 1 | 1;

}

}

}

void init() {

initprime();

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

int t = cal(i);

if (~t) {

cnt[i] = t;

}

}

memset(ans, -1, sizeof(ans));

for (int i = maxn - 1; i >= 1; i--) {

if (cnt[i] <= 1000000) {

ans[cnt[i]] = i;

}

}

}

int main(void) {

init();

int T;

scanf("%d", &T);

while (T--) {

int a;

scanf("%d", &a);

printf("%d\n", ans[a] == -1 ? -1 : ans[a]);

}

}

## 矩阵运算

struct Matrix

{

#define MAXN 2

#define MOD 1

int mat[MAXN][MAXN];

Matrix() {}

Matrix operator\*(Matrix const &b)const

{

Matrix res;

memset(res.mat, 0, sizeof(res.mat));

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

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

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

res.mat[i][j] = (res.mat[i][j]+this->mat[i][k] \* b.mat[k][j]) % MOD;

return res;

}

Matrix operator+(Matrix const &b)const

{

Matrix res;

memset(res.mat, 0, sizeof(res.mat));

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

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

res.mat[i][j] = (this->mat[i][j] + b.mat[i][j]) % MOD;

return res;

}

};

Matrix pow\_mod(Matrix base, int n)

{

Matrix res;

memset(res.mat, 0, sizeof(res.mat));

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

res.mat[i][i] = 1;

while (n > 0)

{

if (n & 1) res = res\*base;

base = base\*base;

n >>= 1;

}

return res;

}

#undef MAXN

#undef MOD

/\*

struct Matrix

{

int col, row;

double \*mat;

Matrix(int x, int y) : col(y), row(x), mat(new double[col \* row]) { }

Matrix(int x, int y, double\* t) : col(y), row(x), mat(new double[col \* row]) { memcpy(mat, t, sizeof(double) \* col \* row); }

double& getNumber(int x, int y) { return \*(this->mat + col \* x + y); }

Matrix operator\*(Matrix &b)

{

Matrix res(this->col, b.col );

memset(res.mat, 0, sizeof(double) \* this->col \* b.row);

for (int i = 0 ;i < this->row; i++)

for (int j = 0; j < b.col; j++)

for (int k = 0; k < this->col; k++)

res.getNumber(i, j) = (res.getNumber(i, j) + getNumber(i, k) \* b.getNumber(k, j));

return res;

}

void operator = (Matrix k)

{

col = k.col; row = k.row; mat = new double[col \* row];

memcpy(mat, k.mat, sizeof(double) \* col \* row);

}

};

\*/

# String

## Kmp

#include <bits/stdc++.h>

using namespace std;

void get\_next(int len, int\* next\_int, string a)

{

int i=0,j=-1;

next\_int[0]=-1;

while(i<len)

{

if(j==-1||a[i]==a[j])

{

i++;

j++;

next\_int[i]=j;

}

else

{

j=next\_int[j];

}

}

}

int main()

{

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

string a,b;

while(cin>>a)

{

cin>>b;

int next[100000];

get\_next((int)b.size(), next, b);

int cnt = 0;

int k = -1;

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

{

while(k > -1 && b[k+1] != a[i])

{

k = next[k];

}

if(b[k+1] == a[i])

{

k++;

}

if(k == b.size() - 1)

{

k=-1;

cnt++;

}

}

cout<<cnt<<endl;

}

return 0;

}

## AC自动机

//ac自动机

struct acnode{

int sum;

acnode\* next[26];

acnode\* fail;

acnode(){

for(int i =0;i<26;i++)

next[i]=NULL;

fail= NULL;

sum=0;

}

};

acnode \*root;

int cnt;

//插入函数

void Insert(char \*s) {

acnode \*p = root;

for(int i = 0; s[i]; i++) {

int x = s[i] - 'a';

if(p->next[x]==NULL) {

acnode \*nn = new acnode();

p->next[x]=nn;

}

p = p->next[x];

}

p->sum++;

}

//获取fail指针，在插入结束之后使用

void getfail(){

queue<acnode\*> q;

q.push(root);

while(!q.empty()){

acnode\* tem = q.front();

q.pop();

for(int i = 0;i<26;i++) {

if(tem->next[i]!=NULL) {

acnode \*p;

p = tem->fail;

while(p!=NULL) {//xuyaojianhua

if(p->next[i]!=NULL){

tem->next[i]->fail = p->next[i];

break;

}

p=p->fail;

}

if(p==NULL) {

tem->next[i]->fail = root;

}

q.push(tem->next[i]);

}

}

}

}

//匹配函数

void ac\_automation(char \*ch) {

acnode \*p = root;

int len = strlen(ch);

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

int x = ch[i] - 'a';

while(p->next[x]==NULL && p != root)//没匹配到，那么就找fail指针。

p = p->fail;

p = p->next[x];

if(!p)

p = root;

acnode \*temp = p;

while(temp != root)

{

if(temp->sum >= 0)

/\*

在这里已经匹配成功了，执行想执行的操作即可，怎么改看题目需求+

\*/

{

cnt += temp->sum;

temp->sum = -1;

}

else break;

temp = temp->fail;

}

}

}

int main()

{

cnt = 0;

int n;

cin>>n;

char c[101];

root = newnode();

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

scanf("%s",c);

Insert(c);

}

getfail();

int m ;

cin>> m;

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

scanf("%s",c);

ac\_automation(c);

}

cout<<cnt<<endl;

return 0;

}

## Trie

//trie字典树

struct tnode{

int sum;//用来判断是否是终点的

tnode\* next[26];

tnode(){

for(int i =0;i<26;i++)

next[i]=NULL;

sum=0;

}

};

tnode \*root;

//插入函数

void Insert(char \*s) {

tnode \*p = root;

for(int i = 0 ; s[i] ; i++) {

int x = s[i] - 'a';

if(p->next[x]==NULL) {

tnode \*nn = new tnode();

p->next[x]=nn;

}

p = p->next[x];

}

p->sum++;//这个单词终止啦

}

//匹配函数

bool Compare(char \*ch) {

tnode \*p = root;

int len = strlen(ch);

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

int x = ch[i] - 'a';

p = p->next[x];

if(p==NULL) {

return false;

}

}

if(p->sum>0 ){

return true;

}

return false;

}

void DELETE(tnode \* &top){

if(top==NULL)

return;

for(int i =0;i<26;i++)

DELETE(top->next[i]);

delete top;

}

int main()

{

int n,m;

cin>>n;

char s[20];

root = new tnode();

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

scanf("%s",s);

Insert(s);

}

cin>>m;

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

scanf("%s",s);

if(Compare(s))

cout<<"YES"<<endl;

else

cout<<"NO"<<endl;

}

DELETE(root);//看见指针就要想到释放,然而这东西会花时间，所以网上很多人写ACM题就不delete了，我很看不惯这一点。

return 0;

}

## 拓扑排序之确定比赛名次(输出字典序最小的答案)

const int maxn = 100005;

int n;

struct node{

int s;

int e;

bool operator<(const node& a){

return s==a.s?e<a.e:s<a.s;

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*类声明\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class Graph

{

int V; // 顶点个数

list<int> \*adj; // 邻接表

priority\_queue<int,vector<int>,greater<int> > q; // 优先队列，优先级低的数字在顶部，维护一个入度为0的顶点的集合

int\* indegree; // 记录每个顶点的入度

public:

Graph(int V); // 构造函数

~Graph(); // 析构函数

void addEdge(int v, int w); // 添加边

bool topological\_sort(); // 拓扑排序

};

queue<int> ans;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*类定义\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Graph::Graph(int V)

{

this->V = V;

adj = new list<int>[V+1];

indegree = new int[V+1]; // 入度全部初始化为0

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

indegree[i] = 0;

}

Graph::~Graph()

{

delete [] adj;

delete [] indegree;

}

void Graph::addEdge(int v, int w)

{

adj[v].push\_back(w);

++indegree[w];

}

bool Graph::topological\_sort()

{

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

if(indegree[i] == 0)

q.push(i); // 将所有入度为0的顶点入队

int count = 0; // 计数，记录当前ans个数

while(!q.empty())

{

int v = q.top(); // 取优先队列顶部元素

q.pop();

ans.push(v);

++count;

// 将所有v指向的顶点的入度减1，并将入度减为0的顶点入栈

list<int>::iterator beg = adj[v].begin();

for( ; beg!=adj[v].end(); ++beg)

if(!(--indegree[\*beg]))

q.push(\*beg); // 若入度为0，则入队

}

if(count < V)

return false; // 没有输出全部顶点，有向图中有回路

else

return true; // 拓扑排序成功

}

int main(){

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

struct node point[maxn];

int m,T;

while(cin>>n>>m){//n个队伍,m场比赛

if(n==1){

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

cin>>point[i].s>>point[i].e;//s胜e

}

cout<<'1'<<endl;

}

else{

Graph g(n);

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

cin>>point[i].s>>point[i].e;//s胜e

}

sort(point+1,point+m);

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

if(point[i].s!=point[i-1].s||point[i].e!=point[i-1].e)

g.addEdge(point[i].s,point[i].e);

}

g.topological\_sort();

while(!ans.empty()){

int v=ans.front();

cout<<v;

ans.pop();

if(!ans.empty())

cout<<' ';

else

cout<<endl;

}

}

}

return 0;

}

# Algorithm

## KM

#include <iostream>

#include <cstring>

#include <cstdio>

using namespace std;

const int MAXN = 305;

const int INF = 0x3f3f3f3f;

int love[MAXN][MAXN]; // 记录每个妹子和每个男生的好感度

int ex\_girl[MAXN]; // 每个妹子的期望值

int ex\_boy[MAXN]; // 每个男生的期望值

bool vis\_girl[MAXN]; // 记录每一轮匹配匹配过的女生

bool vis\_boy[MAXN]; // 记录每一轮匹配匹配过的男生

int match[MAXN]; // 记录每个男生匹配到的妹子 如果没有则为-1

int slack[MAXN]; // 记录每个汉子如果能被妹子倾心最少还需要多少期望值

int N;

bool dfs(int girl)

{

vis\_girl[girl] = true;

for (int boy = 0; boy < N; ++boy) {

if (vis\_boy[boy]) continue; // 每一轮匹配 每个男生只尝试一次

int gap = ex\_girl[girl] + ex\_boy[boy] - love[girl][boy];

if (gap == 0) { // 如果符合要求

vis\_boy[boy] = true;

if (match[boy] == -1 || dfs( match[boy] )) { // 找到一个没有匹配的男生 或者该男生的妹子可以找到其他人

match[boy] = girl;

return true;

}

} else {

slack[boy] = min(slack[boy], gap); // slack 可以理解为该男生要得到女生的倾心 还需多少期望值 取最小值 备胎的样子【捂脸

}

}

return false;

}

int KM()

{

memset(match, -1, sizeof match); // 初始每个男生都没有匹配的女生

memset(ex\_boy, 0, sizeof ex\_boy); // 初始每个男生的期望值为0

// 每个女生的初始期望值是与她相连的男生最大的好感度

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

ex\_girl[i] = love[i][0];

for (int j = 1; j < N; ++j) {

ex\_girl[i] = max(ex\_girl[i], love[i][j]);

}

}

// 尝试为每一个女生解决归宿问题

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

fill(slack, slack + N, INF); // 因为要取最小值 初始化为无穷大

while (1) {

// 为每个女生解决归宿问题的方法是 ：如果找不到就降低期望值，直到找到为止

// 记录每轮匹配中男生女生是否被尝试匹配过

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

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

if (dfs(i)) break; // 找到归宿 退出

// 如果不能找到 就降低期望值

// 最小可降低的期望值

int d = INF;

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

if (!vis\_boy[j]) d = min(d, slack[j]);

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

// 所有访问过的女生降低期望值

if (vis\_girl[j]) ex\_girl[j] -= d;

// 所有访问过的男生增加期望值

if (vis\_boy[j]) ex\_boy[j] += d;

// 没有访问过的boy 因为girl们的期望值降低，距离得到女生倾心又进了一步！

else slack[j] -= d;

}

}

}

// 匹配完成 求出所有配对的好感度的和

int res = 0;

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

res += love[ match[i] ][i];

return res;

}

int main()

{

while (~scanf("%d", &N)) {

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

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

scanf("%d", &love[i][j]);

printf("%d\n", KM());

}

return 0;

}

## 二分

#include <algorithm>

#include <iostream>

#include <iterator>

#include <vector>

template<class ForwardIt, class T, class Compare=std::less<>>

ForwardIt binary\_find(ForwardIt first, ForwardIt last, const T& value, Compare comp={})

{

// 注意：类型 T 和 Forward 解引用后的类型都必须可隐式转换为

// 用于 Compare 的 Type1 和 Type2 。

// 这严格于 lower\_bound 要求（见上述）

first = std::lower\_bound(first, last, value, comp);

return first != last && !comp(value, \*first) ? first : last;

}

int main()

{

std::vector<int> data = { 1, 1, 2, 3, 3, 3, 3, 4, 4, 4, 5, 5, 6 };

auto lower = std::lower\_bound(data.begin(), data.end(), 4);

auto upper = std::upper\_bound(data.begin(), data.end(), 4);

std::copy(lower, upper, std::ostream\_iterator<int>(std::cout, " "));

std::cout << '\n';

// 经典二分搜索，仅若存在才返回值

data = { 1, 2, 4, 6, 9, 10 };

auto it = binary\_find(data.cbegin(), data.cend(), 4); // 选择 '5' 的 < 将返回 end()

if(it != data.cend())

std::cout << \*it << " found at index "<< std::distance(data.cbegin(), it);

return 0;

}

/\*

equal\_range

返回匹配特定键值的元素区间

partition

把一个区间的元素分为两组

upper\_bound

返回指向第一个大于给定值的元素的迭代器

binary\_search

检查等价于 value 的元素是否出现于范围 [first, last) 中。

\*/

## 归并排序求逆序对

//归并算法求逆序对.cpp

//merge\_sort

const int maxn(1e6 + 7);

int n;

ll times;

int row[maxn];

int col[maxn];

void mergec(int l, int r) {

if (l != r) {

int mid = (l + r) / 2;

mergec(l, mid);

mergec(mid + 1, r);

int leftPoint = l, rightPoint = mid + 1, colPoint = 0;

while (leftPoint < mid + 1 && rightPoint < r + 1) {

if (row[leftPoint] < row[rightPoint]) {

col[colPoint++] = row[leftPoint++];

}

else {

col[colPoint++] = row[rightPoint++];

times += mid + 1 - leftPoint;

}

}

while (leftPoint < mid + 1) {

col[colPoint++] = row[leftPoint++];

}

while (rightPoint < r + 1) {

col[colPoint++] = row[rightPoint++];

}

per(i, l, r + 1) {

row[i] = col[i - l];

}

}

}

int main() {

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

while (cin >> n) {

times = 0;

per(i, 1, n + 1) {

cin >> row[i];

}

mergec(1, n);

cout << times << endl;

}

}

## ST

//ST±í

#include<cstdio>

#include<cmath>

#include<algorithm>

using namespace std;

const int maxn = 50005;

int n, q;

//???,???st,???st

int a[maxn], mn[maxn][16], mx[maxn][16];

void pre()

{

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

mx[i][0] = mn[i][0] = a[i];

int t = log(n) / log(2);

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

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

{

mx[j][i] = mx[j][i - 1];

if (j + (1 << (i - 1)) <= n)

mx[j][i] = max(mx[j][i], mx[j + (1 << (i - 1))][i - 1]);

}

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

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

{

mn[j][i] = mn[j][i - 1];

if (j + (1 << (i - 1)) <= n)

mn[j][i] = min(mn[j][i], mn[j + (1 << (i - 1))][i - 1]);

}

}

//????????

int rmq(int l, int r)

{

int m = log(r - l + 1) / log(2);

int a = max(mx[l][m], mx[r - (1 << m) + 1][m]);

int b = min(mn[l][m], mn[r - (1 << m) + 1][m]);

return a - b;

}

int main()

{

scanf("%d%d", &n, &q);

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

scanf("%d", &a[i]);

pre();

int x, y;

while (q--)

{

scanf("%d%d", &x, &y);

printf("%d\n", rmq(x, y));

}

return 0;

}

# 博弈论

// 巴什博奕, 是否先手必胜  
inline bool bash\_game(int n, int m) {  
    //一堆东西, n个物品,最多选m个  
    return n % (m + 1);  
}  
  
// 威佐夫博弈, 是否先手必胜  
// 有两堆各若干的物品，两人轮流从其中一堆取至少一件物品，至多不限，或从两堆中同时取相同件物品，规定最后取完者胜利。  
inline bool wythoff\_game(int n, int m) {  
    if (n > m) {  
        swap(n, m);  
    }  
    int temp = floor((n2 - n1) \* (1 + sqrt(5.0)) / 2.0);  
    return temp != n1;  
}  
// SG函数  
#define N 1001  
//f[]：可以取走的石子个数  
//sg[]:0~n的SG函数值  
int f[N], sg[N], mex[N];  
  
void getSG(int n) {  
    int i, j;  
    memset(sg, 0, sizeof(sg));  
    for (i = 1; i <= n; i++) {  
        memset(mex, 0, sizeof(mex));  
        for (j = 1; f[j] <= i; j++)  
            mex[sg[i - f[j]]] = 1;  
        for (j = 0; j <= n; j++) { //求mes{}中未出现的最小的非负整数  
            if (mex[j] == 0) {  
                sg[i] = j;  
                break;  
            }  
        }  
    }  
}