ACM模板整理

快速幂取模

long long power( long long a , long long b , long long mod )

{

long long acc = 1 ,q;

for( q = b ; q ; q >>=1 )

{

if(q&1)

acc = acc\*a%mod;

a = a\*a%mod;

}

return acc;

}

//用于求a^b%mod.

//////////////////////////////////////////////////////////////

矩阵快速幂

#include <iostream>

#include <cstring>

#define mod 10007

using namespace std;

const int N = 4;//为N×N矩阵

struct Mat{

int mat[N][N];

void clear()

{

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

}

};

Mat mul( Mat A , Mat B )

{

Mat C;

C.clear();

int i , j , k;

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

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

for( k = 0 ; k < N ; k++ )

C.mat[i][j] = (C.mat[i][j] + A.mat[i][k]\*B.mat[k][j])%mod;

return C;

}

Mat fastm( Mat C , int n )

{

if( n == 1 )

return C;

else if( n&1 )

return mul(fastm( mul( C , C) , n >> 1) , C);

else return fastm(mul(C,C) , n>>1);

}

//////////////////////////////////////////////////////////////

Dijkstra算法

//dijkstra算法用于求源点到图中各点的最短路径

void Dijkstra( int t , int n )//t表示源点,n表示边的个数

{

int k;

bool vis[MAXN];//MAXN代表点的个数；

int h[MAXN];//h[i]用于储存i点到源点的距离

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

memset( h , 0x7F , sizeof(h));//将h中的值初始为最大

h[t] = 0;//源点到它自己的距离为0

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

{

k = 0;

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

if(!vis[i]&&(!k||h[i]<h[k]))

k = i;

vis[k] = 1;

for( i = tail[k] ; ~i ; i = edge[i].next)

{

int v = edge[i].to;

h[v] = min( h[v] , h[k] + edge[i].w );

}

}

}

//////////////////////////////////////////////////////////////

//dijkstra算法的优先队列优化

struct cmp

{

bool operator() ( int a , int b )

{

return h[a]>h[b];

}

};//使h值小的优先级较高

void Dijkstra(int t, int n)

{

priority\_queue<int,vector<int>,cmp> q;

memset(h,0x7f,sizeof(h));

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

h[t] = 0;

q.push(t);

while(!q.empty())

{

int u = q.top();

q.pop();

vis[u] = 1;

for( int i = head[u] ; ~i ; i = edge[i].next )

{

int v = edge[i].to;

int w = edge[i].w;

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

{

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

q.push(v);

}

}

}

}

//////////////////////////////////////////////////////////////

spfa算法

struct Edge{

int to,w,next;

}edge[maxn];

addEdge(int u , int v , int w)

{

edge[cnt].to = v;

edge[cnt].next = head[u];

edge[cnt].w = w;

head[u] = cnt++;

}

int vis[maxn],h[maxn];

void spfa( int t )//t为源点

{

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

memset(h,0x7f,sizeof(h));

queue<int> q;

h[t] = 0;

q.push(t);

while(!q.empty())

{

int u = q.front();

q.pop();

vis[u] = 0;

for( int i = head[u] ; ~i ; i = edge[i].next)

{

int v = edge[i].to;

int w = edge[i].w;

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

{

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

if(!vis[v])

{

q.push(v);

vis[v] = 1;

}

}

}

}

}

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

POJ1716

/\*

node1：对于区间放置元素问题，要注意区间开闭性，也就是说要关注对点的约束。

特别注意每个点上放置元素个数的限制，这里一般都是隐含关系的考察点（详见下文）。

node2：对于差分不等式，a - b <= c ，建一条 b 到 a 的权值为 c 的边，求的是最短路，得到的是最大值；对于不等式 a - b >= c ，建一条 b 到 a 的权值为 c 的边，求的是最长路，得到的是最小值。存在负环的话是无解，求不出最短路（dist[ ]没有得到更新）的话是任意解。

node3：建图中有时候会用到一个虚点，这个点到图中每个实点的距离（dist[ ]）为0，当然这个点的作用是方便图中的点入队（spfa算法），然后使这些实点的dist[ ]值得到更新，其实有时候我们可以省略这个点，手动把所有实点入队，同时更新这些实点的 dist[ ] 值和 visit[ ] 值。

\*/

#include <iostream>

#include <cstdio>

#include <cstring>

#include <queue>

#include <algorithm>

using namespace std;

struct Edge

{

int to,next,w;

}edge[40000];

int head[10005],h[10005],cnt,n,c[100005],Min,Max;

bool vis[10005];

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

{

edge[cnt].to = v;

edge[cnt].w = w;

edge[cnt].next = head[u];

head[u] = cnt++;

}

bool spfa()

{

memset( h , 0 , sizeof(h));

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

memset( c , 0 , sizeof(c));

queue<int> q;

//将每个点入队一次,同时更新vis值；

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

{

vis[i] = 1;

q.push(i);

}

while(!q.empty())

{

int u = q.front();

q.pop();

vis[u] = 0;

for( int i = head[u] ; ~i ; i = edge[i].next )

{

int v = edge[i].to , w = edge[i].w;

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

{

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

if(!vis[v])

{

vis[v] = 1;

q.push(v);

}

if( ++c[v] > n )

return false;

}

}

}

return true;

}

int main()

{

scanf("%d",&n);

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

Min = ~0u>>1 , Max = 0 ;

cnt = 1;

int a , b;

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

{

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

Min = min( Min , a);

Max = max( Max , b + 1);

addEdge( a , b + 1 , 2 );

}

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

{

addEdge( i , i + 1 , 0 );

addEdge( i + 1 , i , -1 );

}

if(spfa())

printf("%d\n",h[Max]);

else

printf("0\n");

}

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

POJ1201

/\*

设x[i]是{i}这个集合跟所求未知集合的交集元素个数，明显最大只能是1

再设s[i] = x[0] + x[1] + …… + x[i]

明显的，s[i]表示集合{0,1,2,3,……,i}与所求未知集合的交集元素个数

那么就有x[i] = s[i] - s[i-1]

∵0 <= x[i] <= 1 ∴0 <= s[i] - s[i-1] <= 1

由于题目求最小值，所以是最长路，用的是a - b >= c这种形式

即有：①s[i] - s[i-1] >= 0; ②s[i-1] - s[i] >= -1;

按照题目输入a, b, c：

表示{a,a+1,a+2,……,b}(设这个集合是Q)与所求未知集合的交集元素个数至少为c

而s[a-1]表示{1,2,3,……,a-1}与所求未知集合的交集元素个数

s[b]表示{1,2,3,……,a-1,a,a+1,a+2,……,b}与所求未知集合的交集元素个数

∴Q = s[b] - s[a-1];

即可建立关系: ③s[b] - s[a-1] >= c;

但是还有一个问题a >= 0，那么a-1有可能不合法

解决方法：所有元素+1就可以了。实现：把③变成 s[b+1] - s[a] >= c;

\*/

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <queue>

using namespace std;

struct Edge

{

int to,next,w;

}edge[150005];

int head[50005],h[50005],cnt,n,Min,Max;

bool vis[50005];

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

{

edge[cnt].to = v;

edge[cnt].w = w;

edge[cnt].next = head[u];

head[u] = cnt++;

}

void spfa()

{

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

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

h[Min] = 0;

vis[Min] = 1;

queue<int>q;

q.push(Min);

while(!q.empty())

{

int u = q.front();

q.pop();

vis[u] = 0;

for( int i = head[u] ; ~i ; i = edge[i].next )

{

int v = edge[i].to , w = edge[i].w;

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

{

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

if( !vis[v] )

{

vis[v] = 1;

q.push(v);

}

}

}

}

}

int main()

{

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

{

int a,b,c;

Min = ~0u >> 1,Max = 0;

cnt = 1;

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

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

{

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

Min = min( Min , a );

Max = max( Max , b+1 );

addEdge( a , b + 1 , c );

}

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

{

addEdge( i , i + 1 , 0 );

addEdge( i+1 , i , -1 );

}

spfa();

printf("%d\n",h[Max]);

}

}

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

HDU1529

/\*

先说明一下，这里我把编号设定为1-24，而不是题目的0-23，并且设0为虚点

设x[i]是实际雇用的i时刻开始工作的员工数，R[i]是题目需要的i时刻正在工作的最少员工数，注意了，i时刻开始工作跟i时刻正在工作是完全不同的

设s[i] = x[1] + x[2] + …… + x[i]

则s[i]表示1-i这段时间开始工作的员工数

再设num[i]表示i时刻开始工作的最多可以雇用的员工数

∴有0 <= x[i] <= num[i]，即 0 <= s[i] - s[i-1] <= num[i]

由于是求最小值，所以用最长路

则有：①s[i] - s[i-1] >= 0；②s[i-1] - s[i] >= -num[i]；

(1 <= i <= 24，虽然0是虚点，但是s[1] - s[0]也是必要的！因为x[1]也是有范围的！)由于员工可以持续工作8个小时(R[i]是i时刻正在工作的最少人数)

∴x[i-7] + x[i-6] + …… + x[i] >= R[i]【i-7开始工作的人在i时刻也在工作，其他同理】

即：③s[i] - s[i-8] >= R[i] (8 <= i <= 24)

但是有个特殊情况，就是从夜晚到凌晨的一段8小时工作时间

(x[i+17] + …… + x[24]) + (x[1] + x[2] + …… + x[i]) >= R[i]；

则：s[24] - s[i+16] + s[i] >= R[i];

整理一下：④s[i] - s[i+16] >= R[i] - s[24];

(1 <= i < 8，注意i=0是没有意义的，因为R[0]没有意义)

由于s[24]就是全天实际雇用的人数，而一共有n个员工可以雇用，所以设ans = s[24]

则：⑤s[i] - s[i+16] >= R[i] - ans;( 1 <= i < 8 )

所以就可以从小到大暴力枚举ans【或二分枚举】，通过spfa检验是否有解即可【存在负环无解】

但是还有一个问题，起点在哪里……

这时候虚点0就起作用了，我称它为超级起点

于是建图后直接判断spfa(0)是否有解就可以了

PS：还有另外一个条件必须用到……：⑥s[24] - s[0] >= ans]

不用这个条件二分枚举ans可以AC，但这只是数据问题，暴力从小到大枚举木有这条件就会错，所以说这个条件最关键而又难找……要仔细找特殊点和虚点的约束关系

\*/

#include <iostream>

#include <cstdio>

#include <cstring>

#include <queue>

#include <algorithm>

#define INF 0x7fffffff

using namespace std;

struct Edge

{

int to , next , w;

}edge[1000];

int h[100],head[100],cnt[100],c,t,r[30],num[30],n;

bool vis[1000];

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

{

edge[c].to = v;

edge[c].w = w;

edge[c].next = head[u];

head[u] = c++;

}

bool spfa()

{

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

for( int i = 0 ; i <= 24 ; i++ )

h[i] = -INF;

memset( cnt , 0 ,sizeof(cnt));

h[0] = 0;

vis[0] = 1;

queue<int>q;

q.push(0);

while(!q.empty())

{

int u = q.front();

q.pop();

vis[u] = 0;

for( int i = head[u] ; ~i ; i = edge[i].next )

{

int v = edge[i].to , w = edge[i].w;

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

{

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

if(!vis[v])

{

vis[v] = 1;

q.push(v);

}

if( ++cnt[v] > 24 )

return false;

}

}

}

return true;

}

int main()

{

scanf("%d",&t);

while(t--)

{

memset( num , 0 , sizeof(num));

memset( r , 0 , sizeof(r));

bool f = false;

for( int i = 1 ; i <= 24 ; i++ ){

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

if(r[i]) f = true;

}

scanf("%d",&n);

int temp;

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

{

scanf("%d",&temp);

num[temp+1]++;

}

if( !f )

{

printf("0\n");

continue;

}

int ans;

bool flag = false;

//枚举s[24]所有可能，s[24]为1-24时间内需要开始工作的人数，即所求

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

{

//注意每次枚举都需要初始化数据

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

c = 1;

for( int j = 1 ; j <= 24 ; j++)

{

addEdge( j - 1 , j , 0); //s[i] - s[i-1] >= 0

addEdge( j , j - 1 , -num[j]); //s[i-1] - s[i] >= num[i]

if( j >= 8 )

addEdge( j - 8 , j , r[j]); //s[i] - s[i-8] >= r[i]

}

//s[i] - s[i+16] >= r[i] - s[24]

for( int j = 1 ; j < 8 ; j++ )

addEdge( j + 16 , j , r[j] - i );

addEdge( 0 , 24 , i ); //s[24] - s[0] >= s[24]

if(spfa())

{

flag = true;

ans = i;

break;

}

}

if(flag) printf("%d\n",ans);

else printf("No Solution\n");

}

return 0;

}

//////////////////////////////////////////////////////////////

Tarjan算法

#include <cstdio>

#include <stack>

using namespace std;

struct Edge

{

int to,from,next;

}edge[MAXM];

int head[MAXN],vis[MAXN],dfn[MAXN],low[MAXN],cnt,ans,deep;

void addEdge( int u , int v )

{

edge[cnt].from = u;

edge[cnt].to = v;

edge[cnt].next = head[u];

head[u] = cnt++;

}

stack<int> st;

void tarjan( int x )

{

int i;

dfn[x] = low[x] = ++deep;

vis[x] = 1;

st.push(x);

for( i = head[u] ; ~i ; i = edge[i].next )

{

int v = edge[i].to;

if( dfn[v] == -1 )

{

tarjan(v);

if( low[x] > low[v] )

low[x] = low[v];

}

else if( vis[v] && low[x] > dfn[v] )

low[x] = dfn[v];

}

if( low[x] == dfn[x] )

{

ans++;

while(!st.empty())//循环中取出的为一个强连通分量

{

i = st.top();

st.pop();

vis[i] = 0;

if( i == x )

break;

}

}

}

int main()

{

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

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

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

memset(low,0,sizeof(low));

cnt = 0 , deep = 0 ,ans = 0;

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

if(dfn[i] == -1)

tarjan(i);

//ans为强连通分量的个数；

}

//////////////////////////////////////////////////////////////

网络流

（1）方格取数

1.首先是方格内有固定的权值，可以取不相邻的数，问怎样取使权值最大。

这样我们奇偶建图，源点掌管奇属性点，汇点掌管偶属性点，然后相邻的两点建边容量无穷大，源汇向自己掌管的点建边，容量为权值。这样建图的意义在于，如果某一条边被割掉，那么久说明这个点被抛弃了（不选这个点），那么可不可能选到两个相邻的点？由于相邻两点的边为无穷大，那么这条边肯定不会被割掉，也就是说要么和源点相连的点被割掉，要么和汇点相连的被割掉！这样建图，被割掉一定是和源点或汇点相连的边，切记。结果是权值总和减去最小割。

两点间建边无穷大，可以这么认为，目的是为了最大化取的点是相邻点的代价，通过这种方式使得相邻两点必定不能同时取（否则这个代价太大）。

2.方格内有固定权值，可以取相邻的数，如果相邻的数被取走，那么需要付出一个这两个数同时被取走的代价，问怎样取使得权值最大。

同样奇偶建图，源汇分别和自己掌管的点建边，容量为权值。现在重点来了，因为可以同时取走相邻的点，那么我们相邻点之间就不能建边无穷大来约束了。其实约束条件其实就是相邻两点被取走的代价。怎么理解？如果同时取走两点的代价超过取走这两点带来的价值，那么说明这个方案不是最优的，因为显然我们可以只取其中一点。第一类问题其实就是通过将约束条件无穷大化来约束使得不能同时选择相邻的两点而已。

3.方格内有两种属性的权值，每个方格可以取其中一个属性的权值，可以取相邻的数，如果取走的相邻方格属于同一属性，需要花费相应的代价，问怎样取使得权值最大。

建图稍微不同：奇属性的格子的第一种属性和源点建边，第二种属性和汇点建边，容量为该属性在该格子对应的权值。偶属性的格子的第一种属性和汇点建边，第二种属性和源点建边，容量为该属性在该格子对应的权值。这样由于同一个格子我们只能取走其中一个属性的权值，那么对这两个属性建边无穷大约束一下就好。相邻两点取走相同属性怎么办？其实只要对该点的和源点相连的属性对相邻点的同一属性建边就行了（相邻点和该点同一属性则相邻点的该属性必定和汇点相连），约束条件就是代价。这样建图以后，每个点要么选取其中一个属性，要么都不选。

总结一下这一类问题，基本是奇偶建图，然后如果要求能取相邻点，那么相邻点建边，容量为代价，否则建边无穷大约束。建边无穷大可以约束两个条件使至多只能选取一个。

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

HDU3820

#include <cstdio>

#include <cstring>

#include <queue>

#define INF 0x7fffffff

#define MAXM 1000000

#define MAXN 6000

using namespace std;

int dir[4][2] = {{0,1},{0,-1},{1,0},{-1,0}};

struct Edge

{

int to,next,cap;

}edge[MAXM];

int head[MAXN],cnt,dis[MAXN],n,m,T,G,S;

int board1[60][60],board2[60][60];

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

{

edge[cnt].to = v;

edge[cnt].cap = cap;

edge[cnt].next = head[u];

head[u] = cnt++;

edge[cnt].to = u;

edge[cnt].cap = 0;

edge[cnt].next = head[v];

head[v] = cnt++;

}

bool bfs()

{

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

queue<int>q;

q.push(0);

dis[0] = 0;

while(!q.empty())

{

int u = q.front();

q.pop();

for( int i = head[u] ; ~i ; i = edge[i].next )

{

int v = edge[i].to , cap = edge[i].cap;

if( dis[v] < 0 && cap )

{

dis[v] = dis[u] + 1;

q.push(v);

}

}

}

return dis[T] > 0;

}

int dfs( int x , int low )

{

if( x == T )

return low;

int i , a;

for( i = head[x] ; ~i ; i = edge[i].next )

{

int v = edge[i].to , cap = edge[i].cap;

if( dis[v] == dis[x] + 1 && cap > 0 && ( a = dfs(v,min(cap,low))))

{

edge[i].cap -= a;

edge[i^1].cap += a;

return a;

}

}

dis[x] = -1;///此句话很关键，不再TLE!!!

return 0;

}

int dinic()

{

int ans = 0 , tans ;

while(bfs())

while( tans = dfs( 0 , INF ))

ans += tans;

return ans;

}

int main()

{

int t,num = 1;

scanf("%d",&t);

while(t--)

{

scanf("%d%d%d%d",&n,&m,&G,&S);

int i , j , k , temp , tp = n\*m , sum = 0 ,ax , ay , cnt = 0;

T = 2\*tp+1;

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

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

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

scanf("%d",&board1[i][j]);

sum+=board1[i][j];

}

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

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

scanf("%d",&board2[i][j]);

sum+=board2[i][j];

}

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

{

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

{

temp = (i-1)\*m+j;

if( (i+j)%2 == 0 )

{

addEdge( 0 , temp , board1[i][j]);

addEdge( temp + tp , T , board2[i][j]);

addEdge( temp , temp + tp , INF );

for( k = 0 ; k < 4 ; k++ )

{

ax = i + dir[k][0] , ay = j + dir[k][1];

if( ax > 0 && ax <= n && ay > 0 && ay <= m )

addEdge( temp , (ax-1)\*m+ay , G );

//tab[temp][(ax-1)\*m+ay] += G;

}

}

else

{

addEdge( 0 , temp+tp , board2[i][j]);

addEdge( temp , T , board1[i][j]);

addEdge( temp + tp , temp , INF );

//tab[0][temp+tp] += board2[i][j];

//tab[temp][T] += board1[i][j];

//tab[temp+tp][temp] = INF;

for( k = 0 ; k < 4 ; k++ )

{

ax = i + dir[k][0] , ay = j + dir[k][1];

if( ax > 0 && ax <= n && ay > 0 && ay <= m )

addEdge( temp + tp , (ax-1)\*m+ay+tp , S );

//tab[temp][(ax-1)\*m+ay+tp] += S;

}

}

}

}

printf("Case %d: %d\n",num++,sum-dinic());

}

return 0;

}

//////////////////////////////////////////////////////////////(2)DINIC算法

/\*邻接矩阵模板\*/

int tab[250][250];//邻接矩阵

int dis[250];//距源点距离，分层图

int q[2000],h,r;//BFS队列，首，尾

int N,M,ANS;//N点数，M边数，ANS答案

int BFS()

{

int i,j;

memset( dis , 0xff , sizeof(dis));//以-1填充

dis[1] = 0;

h = 0 ;

r = 1;

q[1] = 1;

while( h < r )

{

j = q[++h];

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

if( dis[i] < 0 && tab[j][i]>0)

{

dis[i] = dis[j] + 1;

q[++r] = i;

}

}

if( dis[N] > 0 )

return 1;

else

return 0;//汇点的dis小于零，表明BFS不到汇点

}

//dinic代表一次增广，函数返回本次增广的流量，返回零表示无法增广

int dfs( int x , int low )//low是源点到现在剩余流量最小的边的剩余量

{

int i , a = 0;

if( x == N )

return low;

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

if( tab[x][i] > 0 //联通

&& dis[i] == dis[x] + 1 //是分层图的下一层

&&(a = dfs(i,min(low,tab[x][i]))))//能到汇点

{

tab[x][i] -= a;

tab[i][x] += a;

return a;

}

return 0;

}

int dinic()

{

int tans ,ans = 0;

while( BFS() )//不停地建立分层图，如果BFS不到汇点才结束

{

while( tans = dfs( 1 , 0x7fffffff))//一次BFS要不停寻找增广路

ans += tans; //直到找不到为止

}

return ans;

}

int main()

{

int i,j,f,t,flow;

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

{

memset( tab , 0 , sizeof(tab));

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

{

scanf("%d%d%d",&f,&t,&flow);

tab[f][t] += flow;

}

ANS = dinic();

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

}

return 0;

}

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

/\*邻接表模板\*/

struct Edge

{

int to,next,cap;

}edge[MAXM];

int head[MAXN],dis[MAXN],cnt,

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

{

edge[cnt].to = v;

edge[cnt].cap = w;

edge[cnt].next = head[u];

head[u] = cnt++;

edge[cnt].to = u;

edge[cnt].cap = 0;

edge[cnt].next = head[v];

head[v] = cnt++;

}

bool bfs()

{

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

queue<int> q;

dis[S] = 0;

q.push(S);

while(!q.empty())

{

int u = q.front();

q.pop();

for( int i = head[u] ; ~i ; i = edge[i].next )

{

int v = edge[i].to , cap = edge[i].cap;

if( dis[v] < 0 && cap > 0)

{

dis[v] = dis[u] + 1;

q.push(v);

}

}

}

return dis[T] > 0;

}

int dfs( int x , int low )

{

if( x == T )

return low;

int a , i;

for( i = head[x] ; ~i ; i = edge[i].next )

{

int v = edge[i].to , cap = edge[i].cap;

if(dis[v]==dis[x] + 1 && cap && (a = dfs( v,min(low,cap))))

{

edge[i].cap -= a;

edge[i^1].cap += a;

return a;

}

}

return 0;

}

int dinic()

{

int ans = 0;

while( bfs() )

ans += dfs( S , 0x7fffffff );

return ans;

}

//////////////////////////////////////////////////////////////

(3)SAP算法

int gap[MAXN],cur[MAXN],aug[MAXN],pre[MAXN];

int sap( int x , int T , int S )// x = T + 1

{

int maxflow = 0 , u = S , v;

int id , mindis;

aug[S] = INF;

pre[S] = -1;

memset( dis , 0 , sizeof(dis));

memset( gap , 0 , sizeof(gap));

gap[S] = x;

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

cur[i] = head[i];

while(dis[S] < x )

{

int flag = 0;

if( u == T )

{

maxflow += aug[T];

for( v = pre[T] ; ~v ; v = pre[v])

{

id = cur[v];

edge[id].cap -= aug[T];

edge[id^1].cap += aug[T];

aug[v] -= aug[T];

if( edge[id].cap == 0)

u = v;

}

}

for( int i = cur[u] ; ~i ; i = edge[i].next )

{

v = edge[i].to;

if( edge[i].cap > 0 && dis[u] == dis[v] + 1)

{

flag = 1;

pre[v] = u;

cur[u] = i;

aug[v] = min( aug[u] , edge[i].cap );

u = v;

break;

}

}

if( !flag )

{

if(--gap[dis[u]] == 0 )

break;

mindis = x;

cur[u] = head[u];

for( int i = head[u] ; ~i ; i = edge[i].next )

{

v = edge[i].to;

if( edge[i].cap > 0 && dis[v] < mindis )

{

mindis = dis[v];

cur[u] = i;

}

}

dis[u] = mindis+1;

gap[dis[u]]++;

if( u != S )

u = pre[u];

}

}

return maxflow;

}

//////////////////////////////////////////////////////////////

(3)费用流

#define MAXN

#define MAXM

#define INF 0x7fffffff

#define M 2139062143

using namespace std;

struct Edge

{

int u,v,w,next,cap;

}edge[MAXM<<2];

int cnt,maxFlow,minCost,head[MAXN],dis[MAXN],path[MAXN];

bool vis[MAXN];

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

{

edge[cnt].v = v , edge[cnt].u = u , edge[cnt].cap = cap;

edge[cnt].w = w , edge[cnt].next = head[u];

head[u] = cnt++;

edge[cnt].u = v , edge[cnt].v = u , edge[cnt].w = -w;

edge[cnt].cap = 0 , edge[cnt].next = head[v];

head[v] = cnt++;

}

bool spfa( int s , int t , int n )

{

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

memset( dis , 0x7f , sizeof(dis));

memset( path , 0xff , sizeof(pp));

dis[s] = 0 , vis[s] = 1;

queue<int> q;

q.push(s);

while(!q.empty())

{

int u = q.front();

q.pop();

vis[u] = 0;

for( int i = head[u] ; ~i ; i = edge[i].next )

{

int v = edge[i].v , w = edge[i].w , cap = edge[i].cap;

if( dis[v] > dis[u] + w && cap )

{

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

path[v] = i;

if(!vis[v])

{

vis[v] = 1;

q.push(v);

}

}

}

}

if( dis[t] == M )

return false;

return true;

}

void mcmf( int s , int t , int n )

{

int i , minflow ;

maxFlow = 0 , minCost = 0;

while( spfa( s , t , n ))

{

minflow = INF;

for( i = path[t] ; ~i ; i = path[edge[i].u])

minflow = min ( edge[i].cap , minflow );

maxFlow += minflow;

for( i = path[t] ; ~i ; i = path[edge[i].u])

{

edge[i].cap -= minflow;

edge[i^1].cap += minflow;

}

minCost += dis[t]\*minflow;

}

}

int main()

{

//输入数据

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

cnt = 0;//必须为0！！

/\*

......建图

\*/

int S , T ; //S为源点，T为汇点

mcmf( S , T , T + 1 );

//输出数据，最大流为maxFlow,最小费用为minCost

return 0;

}

//////////////////////////////////////////////////////////////

并查集

void makeSet(int n)

{

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

{

a[i] = i;

num[i] = 1;

}

}

int find( int x )

{

return a[x] == x ? x : a[x] = find(a[x]);

}

void unionSet( int ax , int bx )

{

int x = find(ax), y = find(bx);

if( x == y )

return;

if( num[x] <= num[y] )

{

a[x] = y;

num[y] += num[x];

}

else

{

a[y] = x;

num[x] += num[y];

}

}

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

void makeSet( int n )

{

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

a[i] = i;

}

int find( int x )

{

return a[x] == x ? x : a[x] = find(a[x]);

}

void unionSet( int x , int y )

{

if( a[x] != a[y] )

a[find(y)] = find(x);

}

//////////////////////////////////////////////////////////////

最小生成树

（1）prim算法

#include <cstdio>

#include <cstring>

#define INF 0x7fffffff

#define MAXN 100

using namespace std;

int map[MAXN][MAXN],low[MAXN],vis[MAXN];

int n;

int prim()

{

int i , j , pos , min , result = 0;

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

vis[1] = 1 , pos = 1;

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

if( i!= pos )

low[i] = map[pos][i];

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

{

min = INF;

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

if( !vis[j] && min > low[j])

{

min = low[j];

pos = j;

}

result += min;

vis[pos] = 1;

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

if( !vis[j] && low[j] > map[pos][j])

low[j] = map[pos][j];

}

return result;

}

int main()

{

memset( map , 0x7f , sizeof(map));

return 0;

}

//////////////////////////////////////////////////////////////

(2)kruskal算法

#define MAXM

#define MAXN

using namespace std;

struct Edge

{

int from , to , w;

}edge[MAXM];

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

{

edge[cnt].to = v;

edge[cnt].from = u;

edge[cnt++].w = w;

}

bool cmp( edge a , edge b )

{

return a.w < b.w;

}

int a[MAXN],cnt;

int n , m ;

void makeSet( int n )

{

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

a[i] = i;

}

int find( int x )

{

return a[x] == x ? x : a[x] = find(a[x]);

}

void unionSet( int x , int y )

{

if( a[x] != a[y] )

a[find(y)] = find(x);

}

int kruskal( int n , int m )//n为顶点个数，m为边数

{

sort( edge , edge + m , cmp );

makeSet(n);

int i , j , sum = 0 ;

for( i = 0 ， j = 0 ; j < n - 1 && i < m ; i++ )

if( find(edge[i].from) != find(edge[i].to) )

{

unionSet( edge[i].from , edge[i].to );

sum += edge[i].w;

j++;

}

return sum;

}

//////////////////////////////////////////////////////////////

(3)次小生成树

POJ1679

判断生成树的唯一性，唯一则输出权值，不唯一输出Not Unique!(POJ1679)

显然，可以转化为求次小生成树，次小生成树权值=最小生成树，则不唯一。

#include <iostream>

#include <algorithm>

#include <queue>

const int INF = 0x7fffffff;

const int MAX = 10001;

int n,m,num;

int p[MAX];

int max[101][101];

struct Edge //原始图

{

int from;

int to;

int w;

bool flag;

}e[MAX];

struct Tree //最小生成树

{

int to;

int w;

int next;

}tree[202];

int index[101];

struct Node //生成树的结点

{

int seq; //结点编号

int max; //从某个点到它的路径中的最大边的长度

};

bool cmp(const Edge &a, const Edge &b)

{

return a.w < b.w;

}

void makeSet()

{

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

{

p[i] = i;

}

}

int findSet(int x)

{

if(x != p[x])

p[x] = findSet(p[x]);

return p[x];

}

void addEdge(int from, int to, int w)

{

tree[num].to = to;

tree[num].w = w;

tree[num].next = index[from];

index[from] = num++;

}

int kruscal()

{

int i,j;

int x, y;

int edgeNum = 0;

int result = 0;

makeSet();

std::sort(e,e+m,cmp);

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

{

x = findSet(e[i].from);

y = findSet(e[i].to);

if(x != y)

{

edgeNum++;

addEdge(e[i].from,e[i].to,e[i].w);

addEdge(e[i].to,e[i].from,e[i].w);

e[i].flag = true;

p[x] = y;

result += e[i].w;

}

}

return edgeNum == n-1 ? result : -1;

}

void bfs(int p)

{

int i,j;

bool used[101];

memset(used,0,sizeof(used));

std::queue<Node> que;

Node now,adj;

now.max = 0;

now.seq = p;

que.push(now);

used[p] = true;

while(!que.empty())

{

Node q = que.front();

que.pop();

for(i = index[q.seq]; i != -1; i = tree[i].next)

{

adj.seq = tree[i].to;

adj.max = tree[i].w;

if(!used[adj.seq])

{

if(q.max > adj.max)

adj.max = q.max;

max[p][adj.seq] = adj.max;

used[adj.seq] = true;

que.push(adj);

}

}

}

}

void second\_MST()

{

int i,j;

int mst = kruscal();

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

bfs(i);

int smst = INF;

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

{

if(!e[i].flag)

{

if(mst + e[i].w - max[e[i].from][e[i].to] < smst)

smst = mst + e[i].w - max[e[i].from][e[i].to];

}

}

if(smst == mst)

printf("Not Unique!\n");

else

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

}

int main()

{

int i,j;

int cases;

int a,b,w;

scanf("%d",&cases);

while(cases--)

{

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

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

{

scanf("%d %d %d",&e[i].from,&e[i].to,&e[i].w);

e[i].flag = false;

}

num = 0;

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

second\_MST();

}

return 0;

}

//////////////////////////////////////////////////////////////

康拓展开

**/\***

**X=an\*(n-1)!+an-1\*(n-2)!+...+ai\*(i-1)!+...+a2\*1!+a1\*0!**

**ai为整数，并且0<=ai<i(1<=i<=n)**

**应用实例：**

**{1,2,3,4,...,n}的排列总共有n!种，将它们从小到大排序，怎样知道其中一种排列是有序序列中的第几个？**

**如 {1,2,3} 按从小到大排列一共6个：123 132 213 231 312 321。想知道321是{1,2,3}中第几个大的数。**

**这样考虑：第一位是3，小于3的数有1、2 。所以有2\*2!个。再看小于第二位，小于2的数只有一个就是1 ，所以有1\*1!=1 所以小于32的{1,2,3}排列数2\*2!+1\*1!=5个。所以321是第6个大的数。2\*2!+1\*1!是康托展开。**

**再举个例子：1324是{1,2,3,4}排列数中第几个大的数：第一位是1小于1的数没有，是0个，0\*3!，第二位是3小于3的数有1和2，但1已经在第一位了，**

**所以只有一个数2，1\*2! 。第三位是2小于2的数是1，但1在第一位，所以有0个数，0\*1!，所以比1324小的排列有0\*3!+1\*2!+0\*1!=2个，1324是第三个大数。\*/**

**int fac[] = {1,1,2,6,24,120,720,5040,40320}; //i的阶乘为fac[i]**

**/\* 康托展开.**

**{1...n}的全排列由小到大有序，s[]为第几个数 \*/**

int KT ( int n , int s[] )

{

int i , j , t , sum = 0 ;

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

{

t = 0;

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

if ( s[j] < s[i] )

t++;

sum += t\*fac[n-i-1];

}

return sum + 1 ;

}

//////////////////////////////////////////////////////////////

约瑟夫问题

**/\***

**已知n个人（以编号0,1，2，3,…,n-1分别表示）围坐在一张圆桌周围；从编号为k的人开始报数，数到m的那个人出列,如此循环，直至只剩一人。**

**约瑟夫环的递推公式：f[1]=0;f[i]=(f[i-1]+m)%i;**

**(i>1) , f[i]为胜利者**

**公式的推导：**

**给出一个序列，从0～n-1编号。**

**其中，k代表出列的序号的下一个，即k-1出列。**

**A: 0, 1, …, k-1, k, k+1, …, n-1**

**那么，出列的序号是(m-1)%n，k=m%n。**

**出列k-1后，序列变为**

**B: 0, 1, …, k-2, k, k+1, …, n-1**

**然后，我们继续从n-1后延长这个序列，可以得到**

**C`: 0, 1, …, k-2, k, k+1, …, n-1, n, n+1, …, n+k-2**

**我们取从k开始直到n+k-2这段序列。其实这段序列可以看作将序列b的0~k-2段移到了b序列的后面。这样，得到一个新的序列**

**C: k, k+1, …, n-1, n, n+1, …, n+k-2**

**好了，整个序列c都减除一个k，得到**

**D: 0, 1, …, n-2**

**C序列中的n-1, n, n+1都减除个k是什么？这个不需要关心，反正c序列是连续的，我们知道了头和尾，就能知道d序列是什么样的**

**。**

**这样你看，从序列a到序列d，就是一个n序列到n-1序列的变化，约瑟夫环可以通过递推来获得最终结果。ok，继续向下。**

**剩下的就是根据n-1序列递推到n序列。假设在n-1序列中，也就是序列d中，我们知道了最终剩下的一个序号是x，那么如果知道了x转换到序列a中的编号x`，**

**不就是知道了最终的结果了么？**

**下面我们就开始推导出序列A中x的序号是什么。**

**D->C，这个变换很容易，就是x+k；**

**C->B，其实就是0~k-2这段序列转换为n~n+k-2这段序列，那么再翻转回去，简单的就是%n，即(x+k)%n。%n以后，k~n-1这段序列值不会发生变化，**

**而n~n+k-2这段序列则变成了0～k-2；这两段序列合起来，就是序列b。**

**于是乎，x`=(x+k)%n。并且，k=m%n，所以x`=(x+m%n)%n=(x+m)%n。公式1就出来了：f[i]=(f[i-1]+m)%i。当然，i=1就是特殊情况了，**

**f[1]=0。这里还有一个小问题要注意。f[i]=(f[i-1]+m)%i 最后MOD的i是个变量，而不是n。**

**特别注意以上公式是从0～n-1，若从1～n,则公式应为**

**f[1]=1;**

**f[i]=(f[i-1]+m)%i (i>1);**

**if(f[i]==0) f[i]=i;**

**或者**

**ans[i]=(ans[i-1]+m-1)%(n-i+1)**

**此时ans[i]为被杀掉的人**

**\*/**

#include <iostream>

#include <cstring>

#include <cstdio>

using namespace std;

int table[14];

int main()

{

memset(table,0,sizeof(table));

int k;

while(~scanf("%d",&k)&&k)

{

if(table[k])

{

printf("%d\n",table[k]);

continue;

}

int n = 2\*k,m = 1;

int f[30] = {0};

f[0] = 0;

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

{

f[i] = ( f[i-1] + m - 1 ) % ( n - i + 1);

if( f[i] < k )

{

i = 0;

m++;

}

}

table[k] = m;

printf("%d\n",table[k]);

}

}

//////////////////////////////////////////////////////////////

IDA\*

八数码问题

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

#define SIZE 3

using namespace std;

int bound , sx , sy ,board[SIZE][SIZE] , mem[100];

int dir[4][2] = { {-1,0} , {0,1} , {0,-1} , {1,0}};

char op[] = {'u','r','l','d'};

int goal\_state[9][2] = {{0,0},{0,1},{0,2},{1,0},{1,1},{1,2},{2,0},{2,1},{2,1}};

bool flag;

int h( int board[][SIZE] )

{

int cost = 0;

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

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

if( board[i][j] != SIZE\*SIZE )

cost += abs ( i - goal\_state[ board[i][j] - 1 ][0] ) + abs( j - goal\_state[ board[i][j] - 1 ][1] );

return cost;

}

int dfs ( int x , int y , int dv , int pre )

{

int hv = h(board);

if( hv == 0 )

{

flag = true;

return dv;

}

if ( hv + dv > bound )

return hv + dv ;

int next = 1e9;

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

{

if( pre + i == 3 )

continue;

int ax = x + dir[i][0] , ay = y + dir[i][1];

if( ax >=0 && ay>=0 && ax < SIZE && ay < SIZE )

{

mem[dv] = i;

swap ( board[x][y] , board[ax][ay] );

int new\_bound = dfs ( ax , ay , dv + 1 , i );

if( flag )

return new\_bound;

next = min ( new\_bound , next );

swap ( board[x][y] , board[ax][ay] );

}

}

return next;

}

//判断是否可以到达最终状态,求逆序对数，当逆序对为偶数时可以，为奇数时不可以

bool judge ( int board[][SIZE] )

{

int i , j , sum = 0 , b[10] , k = 0;

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

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

if( board[i][j] != SIZE\*SIZE )

b[++k] = board[i][j];

for ( i = 1 ; i < SIZE\*SIZE ; i++ )

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

if( b[i] < b[j])

sum++;

if( sum % 2 == 0 )

return true;

return false;

}

void IDA\_STAR()

{

flag = false;

bound = h(board);

while( !flag && bound <= 100 )

bound = dfs ( sx , sy , 0 , -1 );

}

int main()

{

char c;

while ( cin >> c )

{

memset ( board , 0 , sizeof( board ));

memset ( mem , 0 , sizeof ( mem ) );

if( c == 'x' ){

board[0][0] = SIZE\*SIZE;

sx = 0;

sy = 0;

}

else

board[0][0] = c - '0';

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

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

{

if(i||j)

{

cin >> c;

if( c == 'x' ){

board[i][j] = SIZE\*SIZE;

sx = i;

sy = j;

}

else

board[i][j] = c - '0';

}

}

if( !judge( board ) ){

printf("unsolvable\n");

continue;

}

IDA\_STAR();

if(flag)

{

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

printf("%c",op[mem[i]]);

printf("\n");

}

else

printf("unsolvable\n");

}

return 0;

}

//////////////////////////////////////////////////////////////

归并排序求逆序对

#include <iostream>

#include <cstdio>

#include <algorithm>

#include <cstdio>

using namespace std;

long long cnt = 0;

int temp[500001] , a[500001];

void merge\_( int a[] , int left , int mid , int right )

{

int i1 = left , i2 = mid + 1 , i3 = 0;

while( i1 <= mid && i2 <= right )

{

if( a[i1] > a[i2] )

{

cnt+=mid-i1+1;

temp[i3++] = a[i2++];

}

else

temp[i3++] = a[i1++];

}

while( i1 <= mid )

temp[i3++] = a[i1++];

while( i2<= right)

temp[i3++] = a[i2++];

for( int i = 0 , j = left + i ; i < i3 ; i++ , j++ )

a[j] = temp[i];

}

void mergeSort( int a[] , int left , int right )

{

if( left + 1 > right )

return;

int mid = left + right >> 1;

mergeSort( a , left , mid );

mergeSort( a , mid+1 , right );

merge\_( a , left , mid , right );

}

int main()

{

int num;

while( scanf( "%d" , &num ) && num!=0 )

{

cnt = 0;

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

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

mergeSort( a , 0 , num-1 );

printf( "%lld\n" , cnt );

}

return 0;

}