/\* Lets have a brief summary what i am gonna to do here

Region cut will form a bipartite graph

and as we know its NP hard for maximum independent set in graph

but its somewhat easy when we talk about bipartite graph \*/

// All header files included :p

#include <iostream>

#include <vector>

#include <map>

#include <set>

#include <cstring>

#include <stdio.h>

#include <string.h>

#include <string>

#include <algorithm>

#include <bits/stdc++.h>

#include <cstdio>

using namespace std;

// All of the macros

#define UP 1

#define DOWN 2

#define LEFT 4

#define RIGHT 8

#define INF 1<<30

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

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

typedef pair<int,int> PII;

typedef pair<PII,PII> SIDE; // as side contains 2 coordinates :p i.e. ((x1,y1),(x2,y2))

int n,m;

int xsize,ysize;

int edge[2000][2000];

int label[2000][2000];

int V;

bool adjacent[2000][2000];

int color[2000];

int area[2000];

int cap[2000][2000];

int flow[2000][2000];

int source,sink;

bool visited[2000];

map<int,int> SetToMap(set<int> s) // make set to map for further

{

int cnt = 0;

map<int,int> ret;

set<int>::iterator it;

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

{

ret[\*it] = cnt;

++cnt;

}

return ret;

}

// checking for bipartiteness or one can say we are applying colouring to it

void dfscolor(int v,int c)

{

color[v] = c;

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

{

if(adjacent[v][i])

{

if(color[i]==-1)

{

dfscolor(i,1-c);

}

else

{

if(color[i]!=1-c)

{

//it should never happen

cerr<<"The graph is not bipartite\n";

}

}

}

}

}

/\*

// for checking bipartiteness we can use this formally

// but as we know it shd be bi[arite graph already so we shdnt using this

bool check\_bipartiteness(int v) // v is total n umber of vertices

{

int i,j,temp;

bool ans = true; // let it be initially bipartite

for(i=1;i<=v;i++) // vertices from 1 to v

{ // we have run this loop for the disconnected graph

if(visit[i]==0)

{

queue<int> q; // now running simple code for bfs

q.push(i);

color[i]=1;

while(!q.empty())

{

temp=q.front();

visit[temp]=1;

q.pop();

for(j=0;j<graph[temp].size();j++)

{

if(color[graph[temp][j]]==-1) // if not coloured

color[graph[temp][j]]=1-color[temp];

else if(color[graph[temp][j]]==color[temp]) // if already coloured and color equals to the

{ // its color of parent hence breaks rule of bipartiteness

ans=false; // so break it.... and no eed to chq further

break;

}

if(visit[graph[temp][j]]==0)

q.push(graph[temp][j]);

}

if(!ans)

break;

}

}

if(!ans)

break;

}

return ans;

}\*/

void bicolor()

{

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

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

{

if(color[i]==-1)

{

dfscolor(i,0);

}

}

}

void dfs(int r,int c,int cnt)

{

if(r<0||r>=xsize)

return;

if(c<0||c>=ysize)

return;

if(label[r][c]!=-1)

return;

label[r][c] = cnt;

// Imp point : as humne phle or liya hai isliye abhi and lene se i can get phli value na

// up wala chq kro

if((edge[r][c]&UP)==0)

dfs(r-1,c,cnt);

// down wala chq kro

if((edge[r][c]&DOWN)==0)

dfs(r+1,c,cnt);

// left wala chq kro

if((edge[r][c]&LEFT)==0)

dfs(r,c-1,cnt);

// if right wala chq kro

if((edge[r][c]&RIGHT)==0)

dfs(r,c+1,cnt);

}

int dfsFlow(int cur,int cc)

{

if(cur==sink)

return cc;

if(visited[cur])

return 0;

visited[cur] = true;

for(int i=0;i<V+2;++i)

{

if(cap[cur][i]-flow[cur][i]<=0) // residual graph ka chq portiion

continue;

int t1 = dfsFlow(i,min(cc,(cap[cur][i]-flow[cur][i])));

if(t1==0)

continue;

flow[cur][i]+=t1;

flow[i][cur]-=t1;

return t1;

}

return 0;

}

void solve\_cake()

{

int i,j,side,x,y,cnt=0;

cout<<"\n\n\n\n\n\n\t";

cout<<"Enter the square size of the cake and number of sides :";

cin>>n>>m; // n-> (0,0) to (n,n) assume cake to be of square size

// m-> belong number of cuts

set<int> XS; // as we know set do already sorting

set<int> YS;

// put initial annd final vertex or one can say coordinate

XS.insert(0);

XS.insert(n);

YS.insert(0);

YS.insert(n);

vector<SIDE> sides; // make all sides vector

// taking input

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

{

cout<<"\tEnter the side the piece you want to cut."<<endl;

cin>>side; // sides belong to how many sides are there in a polygon to be cutted

vector<PII> polygon; // storing polygon

cout<<"\tEnter all the pairs or coordinates of the piece yyou want to cut :p."<<endl;

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

{

cin>>x>>y; // input all the coordinates one by one

polygon.push\_back(make\_pair(x,y)); // making pair

XS.insert(x);

YS.insert(y);

}

// generating polygon hereby

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

{

sides.push\_back(make\_pair(polygon[j],polygon[(j+1)%side]));

}

}

// set x coordinate and y coordinate to corresponding map B|

map<int,int> XM = SetToMap(XS);

map<int,int> YM = SetToMap(YS);

xsize = XM.size();

ysize = YM.size();

// make all edges to 0

memset(edge,0,sizeof(edge));

// use of geometry to cut the the all polygons so formed in final pieces ;)

// taken from stackoverflow

// didnot get how to break polygons into all cutted part

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

{

if(sides[i].first.first==sides[i].second.first) //((x1,y1);(x2,y2))

{ // side ka x1 == x2

int xi = XM[sides[i].first.first]; // x1

int y1 = YM[sides[i].first.second];

int y2 = YM[sides[i].second.second];

int ymin = min(y1,y2);

int ymax = max(y1,y2);

for(j=ymin;j<ymax;++j)

{

edge[xi-1][j]|= DOWN; // down = 2

edge[xi][j]|= UP; // up = 1

}

}

else

{

int yi = YM[sides[i].first.second];

int x1 = XM[sides[i].first.first];

int x2 = XM[sides[i].second.first];

int xmin = min(x1,x2);

int xmax = max(x1,x2);

for(j=xmin;j<xmax;++j)

{

edge[j][yi-1]|= RIGHT;

edge[j][yi]|= LEFT;

}

}

}

for(i=0;i<2001;i++)

for(j=0;j<2001;j++)

label[i][j]=-1;

// for all the coordinates running dfs

// also counting total number of components

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

{

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

{

if(label[i][j]==-1)

{

dfs(i,j,cnt);

++cnt;

}

}

}

V = cnt;

// cout<<V<<endl;

for(i=0;i<2001;i++)

for(j=0;j<2001;j++)

adjacent[i][j]=0;

// checking sides are adjacent or not

// if they are just putting out there 1

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

{

if(sides[i].first.first==sides[i].second.first)

{

int xi = XM[sides[i].first.first];

int y1 = YM[sides[i].first.second];

int y2 = YM[sides[i].second.second];

int ymin = min(y1,y2);

int ymax = max(y1,y2);

for(j=ymin;j<ymax;++j)

{

int l1 =label[xi-1][j];

int l2 = label[xi][j];

if(l1!=l2)

{

adjacent[l1][l2] = true;

adjacent[l2][l1] = true;

}

}

}

else if(sides[i].first.second==sides[i].second.second)

{

int yi = YM[sides[i].first.second];

int x1 = XM[sides[i].first.first];

int x2 = XM[sides[i].second.first];

int xmin = MIN(x1,x2);

int xmax = MAX(x1,x2);

for(j=xmin;j<xmax;++j)

{

int l1 = label[j][yi-1];

int l2 = label[j][yi];

if(l1!=l2)

{

adjacent[l1][l2] = true;

adjacent[l2][l1] = true;

}

}

}

}

//cout<<"1"<<endl;

bicolor();

// cout<<"2"<<endl;

// copying all the vector to vertexes of set to set

// as we cant modify our set easily

vector<int> XV(XS.begin(),XS.end());

vector<int> YV(YS.begin(),YS.end());

for(i=0;i<2001;i++)

area[i]=0;

// cout<<"3"<<endl;

// now our main area of all pollygon is here

// also we have considered bipartiteness and all

for(i=0;i<xsize-1;++i)

{

for(int j=0;j<ysize-1;++j)

{

area[label[i][j]]+=(XV[i+1]-XV[i])\*(YV[j+1]-YV[j]);

}

}

// main program now starts

// cout<<"4"<<endl;

for(i=0;i<2001;i++)

for(j=0;j<2001;j++)

cap[i][j]=0;

// Maximum flow and minimum cut now starts hereby

source=V;

sink=V+1;

// cout<<"5"<<endl;

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

{

if(color[i]==0)

{

cap[source][i] = area[i];

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

{

if(color[j]==1 && adjacent[i][j])

cap[i][j] = INF;

}

}

else cap[i][sink] = area[i];

}

// cout<<"6"<<endl;

memset(flow,0,sizeof(flow));

/\*for(i=0;i<2001;i++)

for(j=0;j<2001;j++)

flow[i][j]=0;\*/

int res=n\*n;

//cout<<"7"<<endl;

while(true)

{

memset(visited,0,sizeof(visited));

int t1 = dfsFlow(source,INF);

if(t1==0)

break;

res-=t1;

}

system("pause");

system("cls");

cout<<"\n\n\n\n\n\n\t\t";

cout<<"Here is it you are allowed to eat cake :p :";

cout<<res<<endl; // maximum area i can get from cake :p

system("pause");

system("cls");

}

void solve\_digits()

{

vector<int> graph[10];

bool visited[100001];

cout<<"\n\n\n\n\n\n\t\t";

cout<<"Enter the string :"<<endl;

string str;

cin>>str;

int i,len=str.length(),t1;

int ans[len+1];

memset(visited,0,sizeof(visited));

for(i=0;i<10;i++)

graph[i].clear();

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

graph[str[i]-'0'].push\_back(i);

queue<int> Q;

// make initial index markings

ans[0]=0;

Q.push(0);

visited[0]=true;

while(!Q.empty())

{

t1=Q.front();

Q.pop();

// make all updates for which this value is repeated in sequence

for(i=0;i<graph[str[t1]-'0'].size();i++)

{

if(!visited[graph[str[t1]-'0'][i]])

{

ans[graph[str[t1]-'0'][i]]=ans[t1]+1;

visited[graph[str[t1]-'0'][i]]=true;

Q.push(graph[str[t1]-'0'][i]);

}

}

// to make sure it is not repeated again :p

graph[str[t1]-'0'].clear();

// now make moves for right and left wala ;)

if(!visited[t1+1] && t1+1<len)

{

visited[t1+1]=true;

Q.push(t1+1);

ans[t1+1]=ans[t1]+1; // update the answer

}

if(!visited[t1-1] && t1-1>=0)

{

visited[t1-1]=true;

Q.push(t1-1);

ans[t1-1]=ans[t1]+1;

}

}

cout<<"\n\n\n\n\n\n\t";

cout<<"Hurray!! .. This is minimum number of steps you can reach at final step :";

cout<<ans[len-1]<<endl;

system("pause");

system("cls");

}

int main()

{

cout<<"\n\n\n\n\n\n\t\t";

cout<<"Lets Begin with world of logics :p "<<endl;

cout<<"\n\n\n\n\n\n\n\n\n";

system("pause");

system("cls");

cout<<"\n\n\n\n\n\n\t\t";

cout<<"Few intresting Daily Life examples :"<<endl<<endl;

cout<<"\t\t";

cout<<"1) Cake cutting and mine to have maximum piece."<<endl;

cout<<"\t\t";

cout<<"2) Finding minimum steps to reach end."<<endl;

int kp=0;

cin>>kp;

cout<<"\n\n\n\n\n\n\n\n\n";

system("pause");

system("cls");

if(kp==1)

solve\_cake();

else if(kp==2)

solve\_digits();

else

cout<<"Enter another choice .... Sorry :) \n";

return 0;

}