# Project 200 : Algorithm Analysis

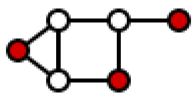
## Dominating Set :

In graph theory, a dominating set for a graph G = (V, E) is a subset D of V such that every vertex not in D is joined to at least one member of D by some edge. The domination number γ(G) is the number of vertices in a smallest dominating set for G.

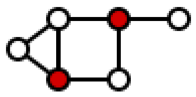
## Complexity :

The dominating set problem concerns testing whether γ(G) ≤ K for a given graph G and input K; it is a classical NP-complete decision problem in computational complexity theory (Garey & Johnson 1979). Therefore it is believed that there is no efficient algorithm that finds a smallest dominating set for a given graph.

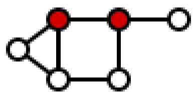
(a)



(b)



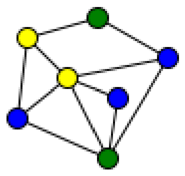
(c)



Figures (a)–(c) on the right show three examples of dominating sets for a graph. In each example, each white vertex is adjacent to at least one red vertex, and it is said that the white vertex is dominated by the red vertex. The domination number of this graph is 2: the examples (b) and (c) show that there is a dominating set with 2 vertices

## Domatic Number :

In graph theory, a domatic partition of a graph  G=(V, E) is a partition of V  into disjoint sets ,V1 ,V2 ,....,Vk,  such that each Vi is a dominating set for G.



The figure above  shows a domatic partition of a graph; here the dominating set  consists of the yellow vertices, consists of the green vertices, and  consists of the blue vertices.
The domatic number is the maximum size of a domatic partition, that is, the maximum number of disjoint dominating sets. The graph in the figure has domatic number 3.

## Problem 1

**LightOj – 1406 – Assacin's Creed**

## Problem Description:

Altair is in great danger as he broke the three tenets of the assassin creed. The three tenets are: 1) never kill an innocent people, 2) always be discrete and 3) never compromise the brotherhood. As a result Altair is given another chance to prove that he is still a true assassin. Altair has to kill n targets located in n different cities. Now as time is short, Altair can send a massage along with the map to the assassin's bureau to send some assassins who will start visiting cities and killing the targets. An assassin can start from any city, but he cannot visit a city which is already visited by any other assassin except him (because they do not like each other's work). He can visit a city multiple times though. Now Altair wants to find the minimum number of assassins needed to kill all the targets. That's why he is seeking your help.

## Input

Input starts with an integer T (≤ 50), denoting the number of test cases.

Each case starts with a blank line. Next line contains two integers n (1 ≤ n ≤ 15) and m (0 ≤ m ≤ 50), where n denotes the number of cities and m denotes the number of one way roads. Each of the next m lines contains two integers u v (1 ≤ u, v ≤ n, u ≠ v) meaning that there is a road from u to v. Assume that there can be at most one road from a city u to v.

## Output

For each case, print the case number and the minimum number of assassins needed to kill all the targets.

## Analysis:

We can simplify this problem into Dominating set.First of all We need to figure out the sets of Vertices which can be visited by a single assacin.If Sets like these can be figured out then we will do a brute force method to figure out combination of smallest number of sets which creates the set of the entire Vertice.

Here The Dfs() function creates the sets which are valid for a single assacin to travel.

And the rec() method uses a brute force approach to to check how minimum number of sets needed to create the set of vertice V.

## Problem Solution :

```
#include <map>
#include <queue>
#include <stack>
#include <cmath>
#include <cctype>
#include <set>
#include <bitset>
#include <algorithm>
```

```
#include <list>
#include <vector>
#include <sstream>
#include <iostream>

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>

using namespace std;


#define mem(a,b) memset(a,b,sizeof a)
#define pb push_back
#define SZ(x) (int)x.size()
#define oo (1<<25)

vector<int>v[16];
int con[1<<15],n;
int col[16][1<<15];
int dp[1<<15];


void dfs(int source,int mask)
{
    col[source][mask]=1;
    con[mask]=true;
    for(int i=0;i<SZ( v[source] );i++ )
    {
        int now=v[source][i];
        if(col[now][mask|(1<<(now-1))]==0)
        {
            dfs(now,mask|(1<<(now-1)) );
        }
    }
    return ;
}


int rec(int mask)
{
    if(mask==0) return 0;
    int &ret=dp[mask];
    if(ret!=-1) return ret;
    ret=oo;
    for(int cur=mask;cur>=0; )
    {
        if(con[cur]) ret=min(ret,rec(mask^cur) + 1);
        cur--;
        if(cur>=0) cur&=mask;
    }
    return ret;
}
int main(void)
{
    int cas,loop=0,m,a,b,ans;
    scanf("%d",&cas);
    while(cas--)
    {
        scanf("%d %d",&n,&m);
        for(int i=0;i<=n;i++) v[i].clear() ;
        for(int i=0;i<m;i++)
        {
            scanf("%d %d",&a,&b);
            v[a].pb(b);
        }
        mem(con,0);mem(col,0);
        for(int i=1;i<=n;i++)
        {
            dfs(i,1<<(i-1));
        }
        mem(dp,-1);
        ans=rec((1<<n)-1);
        printf("Case %d: %d\n",++loop,ans);
    }
    return 0;
}
```

## Problem 2

**Single Round Match 388 Round 1 - Division I, Level Two [Topcoder]**

**Problem Statement :**

You wish to share as many facts as possible with a group of N people, but you only have time to tell one fact to each person in the group. When you tell someone a fact, you also instruct them to tell all their friends. However, the friends do not tell their friends: if A and B are friends, B and C are friends, but A and C are not friends, then after telling the fact to A it will be passed on to B but not to C. You must tell each fact to enough people so that every person either hears the fact from you, or is a friend of someone who heard it from you.

friends contains N strings of N characters, each of which is either 'Y' or 'N'. The jth character of the ith element is 'Y' if members i and j are friends, and 'N' otherwise. Determine the maximum number of facts that can be shared with every person in the group.

**Class: InformFriends**

Method: maximumGroups

Parameters: String[]

Returns: int

Method signature: int maximumGroups(String[] friends)

(be sure your method is public)

**Constraints**

- friends will contain exactly N elements, where N is between 1 and 15, inclusive.
- Each element of friends will contain exactly N characters.
- Each character in friends will be either 'Y' or 'N'.
- For i and j between 0 and N - 1, inclusive, character j of element i of friends will be the same as character i of element j.
- For i between 0 and N - 1, inclusive, character i of element i of friends will be 'N'.

## Analysis :

This is also similar with the Dominating set. Here For each friend we have made a list i.e. The people who will be informed if this person gets the message which is stored in con[] array.
Now For every possbile bit pattern I checked if this pattern is valid.
In the rec() method the input parameter is the people which can be informed.
I brute force all possible subsets of the possible pattern and checked if it is valid or not and made the recursion call which ensures that I will find the maximum number of sets which are dominating.

## Problem Solution

```
#include <map>
#include <queue>
#include <stack>
#include <cmath>
```

```
#include <cctype>
#include <set>
#include <bitset>
#include <algorithm>
#include <list>
#include <vector>
#include <sstream>
#include <iostream>

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>

using namespace std;

typedef long long ll;
typedef vector<int> vi;
typedef pair<int,int> paii;

#define REP(i,n) for (i=0;i<n;i++)

#define PI (2*acos(0))
#define ERR 1e-5
#define mem(a,b) memset(a,b,sizeof a)
#define pb push_back
#define popb pop_back
#define all(x) (x).begin(),(x).end()
#define mp make_pair
#define SZ(x) (int)x.size()
#define oo (1<<25)
#define rep(x,n)        for(int x=0;x<(int)(n);x++)
#define FOREACH(it,x) for(__typeof((x).begin()) it=(x.begin()); it!=(x).end(); +
+it)
#define Contains(X,item)        ((X).find(item) != (X).end())
#define popc(i) (__builtin_popcount(i))
#define fs      first
#define sc      second
#define EQ(a,b)     (fabs(a-b)<ERR)
#define MAX 100


template<class T> T Abs(T x) {return x > 0 ? x : -x;}
template<class T> inline T sqr(T x){return x*x;}
ll Pow(ll B,ll P){      ll R=1; while(P>0)      {if(P%2==1)
R=(R*B);P/=2;B=(B*B);}return R;}
int BigMod(ll B,ll P,ll M){     ll R=1; while(P>0)      {if(P%2==1){R=(R*B)
%M;}P/=2;B=(B*B)%M;} return (int)R;} /// (B^P)%M


#include <string>
#include <vector>

int dp[1<<15],con[15];
bool chk[1<<15];

int rec(int mask)
{
    if(mask==0) return 0;
    int &ret=dp[mask];
    if(ret!=-1) return ret;
    ret=0;

    for(int now=mask;now>=0; )
    {
        if(chk[now]) ret=max(ret,rec(mask^now) + 1);
        now--; if(now>=0) now&=mask;
    }
    return ret;
}
class InformFriends {
public:
int maximumGroups(vector <string> friends)
{
    int n=SZ(friends);
    for(int i=0;i<SZ(friends);i++ )
    {
        con[i]=(1<<i);
        for(int j=0;j<SZ( friends[i] );j++ ) if(friends[i][j]=='Y') con[i]|=(1<<j);
    }
    for(int mask=0;mask<(1<<n);mask++)
    {
        int now=0;
        for(int j=0;j<n;j++) if(mask&(1<<j)) now|=con[j];
        if(now==(1<<n)-1) chk[mask]=true;
    }
    mem(dp,-1);
    return rec( (1<<n) -1);
}
};
```

## Problem 3

**Single Round Match 386 Round 1 - Division I, Level Two [Topcoder]**

**Problem Statement :**

You're given several points in the cartesian plane. Return the smallest possible total sum of areas of a set of convex polygons such that each point is covered by at least one polygon. Moreover, the vertices of each polygon must all lie on the given points, and each polygon must have at least three sides. A point is covered by a polygon if the point lies in the polygon's interior or on its boundary.

The points are described by int[]s x and y, where (x[i],y[i]) is the location of the ith point.

**Definition**

Class: PolygonCover

Method: getArea

Parameters: int[], int[]

Returns: double

Method signature: double getArea(int[] x, int[] y)

(be sure your method is public)

**Notes**

- The returned value must be accurate to within a relative or absolute value of 1E-9.
- A polygon is convex if its edges only intersect at its vertices with each vertex sharing exactly two edges, and it's possible to complete a walk around the polygon by only making left turns.
- If two polygons with areas A and B overlap, then an area of A+B is contributed to the result.

**Constraints**

x and y will each contain between 3 and 15 elements, inclusive.

x and y will contain the same number of elements.

Each element of x and y will be between -1000 and 1000, inclusive.

No three points represented by x and y will lie on a common line.

## Analysis :

To solve this problem, we'll use polygon triangulation. Every convex polygon can be decomposed into a set of non-overlapping triangles, with all three vertices of each triangle lying on the original polygon's vertices. This fact allows us to only consider triangles for covering our points. We can make implementing this problem easier by noticing that we actually only need to consider triangles that don't contain any points, besides the points that define its vertices. This is true because we can take any triangle that contains a point in its interior and decompose it into three nonoverlapping triangles whose union has the same area as the original triangle .

We simply iterate through each triple of points {a,b,c} (such that at least one such point must be covered), and consider rec(total_set -a-b-c )+area(a,b,c)) as a possible return value. Our final result is the smallest such value.

**Solution :**

```
#include <map>
#include <queue>
#include <stack>
#include <cmath>
#include <cctype>
#include <set>
#include <bitset>
#include <algorithm>
#include <list>
#include <vector>
#include <sstream>
#include <iostream>

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>

using namespace std;

typedef long long ll;
typedef vector<int> vi;
typedef pair<int,int> paii;

#define mem(a,b) memset(a,b,sizeof a)
#define pb push_back
#define all(x) (x).begin(),(x).end()
#define mp make_pair
#define SZ(x) (int)x.size()
#define oo (1<<25)
#define rep(x,n)        for(int x=0;x<(int)(n);x++)
#define FOREACH(it,x) for(__typeof((x).begin()) it=(x.begin()); it!=(x).end(); +
+it)
#define Contains(X,item)        ((X).find(item) != (X).end())
#define popc(i) (__builtin_popcount(i))
#define fs      first
#define sc      second
#define EQ(a,b)     (fabs(a-b)<ERR)
#define MAX 100



#include <string>
#include <vector>
struct point
```

```
{
    int x,y;
    point(int a,int b)
    {
        x=a;y=b;
    }
};
vector<point>v;

int n;
double dp[1<<16];
double area(point a, point b,point c)
{
    int x1=b.x-a.x , y1= b.y-a.y;
    int x2=c.x-a.x , y2= c.y-a.y;
    return ( 0.5*(double)abs(x1*y2 - x2*y1) );
}

double rec(int mask)
{
    if(mask==0) return 0.0;
    double &ret=dp[mask];
    if(ret>-.5) return ret;
    ret=(double)oo;

    for(int i=0;i<n;i++)
    {
        if(mask&(1<<i))
        {
            for(int j=0;j<n;j++)
            {
                if(i==j) continue;
                for(int k=0;k<n;k++)
                {
                    if(i==k || j==k) continue;
                    ret=min(ret,rec(mask& (~((1<<i)|(1<<j)|(1<<k))) )  +
area(v[i],v[j],v[k]) );
                }
            }
            break;
        }

    }
    return ret;

}
class PolygonCover {
public:
double getArea(vector <int> x, vector <int> y)
{
    for(int i=0;i<SZ(x);i++)
        v.pb(point(x[i],y[i]));
    n=SZ(x);
    mem(dp,-1);
    return rec((1<<n)-1);
}
};
```