BIT

```
struct BIT
{
    int N;
    vector<int> bit;

    void init(int n)
    {
        N = n;
        bit.assign(n + 1, 0);
    }

    void update(int idx, int val)
    {
        while(idx <= N)
        {
            bit[idx] += val;
            idx += idx & -idx;
        }
    }

    void updateMax(int idx, int val)
    {
        while(idx <= N)
        {
            bit[idx] = max(bit[idx], val);
            idx += idx & -idx;
        }
    }

    int pref(int idx)
    {
        int ans = 0;
        while(idx > 0)
        {
            ans += bit[idx];
            idx -= idx & -idx;
        }
        return ans;
    }

    int rsum(int l, int r)
    {
        return pref(r) - pref(l - 1);
    }

    int prefMax(int idx)
    {
        int ans = -2e9;
        while(idx > 0)
        {
            ans = max(ans, bit[idx]);
            idx -= idx & -idx;
        }
        return ans;
    }
};
BIT b;
b.init(10);
```

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

DSU

```
int const N = 1e5 + 10 ;
int par[N];
int sz[N];
void init(){
    for(int i=1;i<N;i++){
        par[i] = i ;
        sz[i] = 1 ;
    }
}
int fp(int i){
    if(par[i]==i){
        return i ;
    }
    return fp(par[i]);
}
void merge(int x,int y){
    int X = fp(x);
    int Y = fp(y);
    if(X == Y )return ;
    if(sz[X] > sz[Y]){
        swap(X,Y);
    }
    par[X] = par[Y];
    sz[Y] += sz[X];
    sz[X] = 0 ;
}
```

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

Fenwick

```
int bit[100001];
int n;
#define LSB(i) ((i) & -(i))
int  sum(int i)
{
        int j=0; while(i>0) {j+=bit[i];i-=LSB(i);} return j;
}
void add(int i, int k)
{
        while (i <= n)
      bit[i] += k, i += LSB(i);
}
////////////////////////
```

Gauss

```
#include<bits/stdc++.h>
using namespace std;
#define MAX_N 100 // adjust this value as needed
struct AugmentedMatrix { double mat[MAX_N][MAX_N + 1]; };
struct ColumnVector { double vec[MAX_N]; };
ColumnVector GaussianElimination(int N, AugmentedMatrix Aug) { // O(N^3) // input: N, Augmented Matrix Aug, output: Column vector X, the answer
        int i, j, k, l; double t; ColumnVector X;
        for (j = 0; j < N - 1; j++) { // the forward elimination phase
                l = j;
                for (i = j + 1; i < N; i++) // which row has largest column value
                        if (fabs(Aug.mat[i][j]) > fabs(Aug.mat[l][j]))
                        l = i; // remember this row l // swap this pivot row, reason: to minimize floating point error
                        for (k = j; k <= N; k++) // t is a temporary double variable
                                t = Aug.mat[j][k], Aug.mat[j][k] = Aug.mat[l][k], Aug.mat[l][k] = t; for (i = j + 1; i < N; i++) // the actual forward elimination phase
                        for (k = N; k >= j; k--)
                                Aug.mat[i][k] -= Aug.mat[j][k] * Aug.mat[i][j] / Aug.mat[j][j];
        }
        for (j = N - 1; j >= 0; j--) {
                // the back substitution phase
                 for (t = 0.0, k = j + 1; k < N; k++) t += Aug.mat[j][k] * X.vec[k]; X.vec[j] = (Aug.mat[j][N] - t) / Aug.mat[j][j]; // the answer is here
        }
return X;
}
int gauss(){
   // https://cp-algorithms.com/linear_algebra/linear-system-gauss.html
   int row=0,col=0;
   for (int j=0;j<m;j++){
        for (int i=row;i<n;i++){
            if(b[i][j]){
                swap(b[i],b[row]);break;
            }
        }
        if(!b[row][j])continue;
        int p=row+1;
        for (int i=p;i<n;i++){
            if (b[i][j]){
                b[i]^=b[row];
            }
        }
        row++;col++;
    }
    return col;// number of pivot cols
}
int main(){
            AugmentedMatrix A;
            double m[3][4] = {{1,1,2,9},{2,4,-3,1},{3,6,-5,0}};
            for(int i = 0;i<3;i++){
                        for(int j=0;j<4;j++){
                                    A.mat[i][j] = m[i][j];;
                        }
            }
            ColumnVector c = GaussianElimination(3,A);
            for(int i=0;i<3;i++){
                        cout << c.vec[i] << " ";
            }
            return 0;
}
```

Geometry

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

const double EPS = 1e-10,PI = acos(-1);

double DEG_to_RAD(double theta){
  return ((theta*PI)/180.0);
}
//************** Point *****************
class point{
public:
  double x,y;
  point(){x=0;y=0;}
  point(double _x,double _y):x(_x),y(_y){}
  bool operator < (point other) const{
    if(fabs(x - other.x)>EPS)return x<other.x;
    return y<other.y;
  }
  bool operator == (point other) const{
    return ((fabs(x-other.x)<EPS )&&(fabs(y-other.y)<EPS));
  }
};
double dist(point p1,point p2){
  return hypot(p1.x-p2.x,p1.y-p2.y);
}
//rotate p by theta "degrees" counter clockwise wrt origin (0,0)
point rotate(point p,double theta){
  double rad = DEG_to_RAD(theta);
  return point(p.x*cos(rad)-p.y*sin(rad) , p.x*sin(rad)+p.y*cos(rad));
}

//************** LINE ******************
//Represented as ax+by+c=0 where b=1 i.e y=(-A)x+(-C)
class line{
public:
  double a,b,c;line(){a=b=c=0;}
  line(point p1,point p2){
    if(fabs(p1.x-p2.x)<EPS)//vertical line
      a=1.0,b=0.0,c=-p1.x;
    else
      a=-(p2.y-p1.y)/(p2.x-p1.x),b=1,c=-(p1.y + a*p1.x);
  }//slope : m and point p on line
  line(point p,double m){a=-m;b=1;c=-(a*p.x+b*p.y);}

  int sub(point p) {
            double k = a*p.x + b*p.y + c;
            if(k <= -EPS) return -1;
            else if(k >= +EPS) return +1;
            else return 0;
  }
};

bool areParallel(line l1,line l2){
  return ((fabs(l1.a-l2.a)<EPS) && (fabs(l1.b-l2.b)<EPS));
}
bool areSame(line l1,line l2){
  return (areParallel(l1,l2) && (fabs(l1.c-l2.c)<EPS));
}
bool areIntersect(line l1,line l2,point &p){
  if(areParallel(l1,l2)) return false;
  //solve system of 2 linear algebraic equations with 2 unknowns
  p.x = (l2.b*l1.c - l1.b*l2.c)/(l2.a*l1.b - l1.a*l2.b);
  //special case for vertical line to avoid division by zero.
  if(fabs(l1.b)>EPS)p.y= -(l1.a*p.x + l1.c);
  else p.y = -(l2.a*p.x + l2.c);
  return true;
}

/************* Vector Algebra *************/
class vect{
  public:
    double x,y;vect():x(0),y(0){}
    vect(double _x,double _y):x(_x),y(_y){}
    vect(point a,point b):x(b.x-a.x),y(b.y-a.y){}
};
double dot(vect a,vect b){
  return (a.x*b.x + a.y*b.y);
}
double cross(vect a,vect b){
  return a.x*b.y - a.y*b.x;
}
//Returns vector of length s along v
vect scale(vect v,double s){
  double k = s/sqrt(dot(v,v));
  return vect(k*v.x,k*v.y);
}
//Move point p along direction v by length of v
point translate(point p,vect v){
  return point(p.x+v.x,p.y+v.y);
}

/************* Miscellaneous **************/
//dist of p from line(a,b).c:closest point to p on line l
double distToLine(point p,point a,point b,point &c){
  vect ap(a,p),ab(a,b);double u = dot(ap,ab)/sqrt(dot(ab,ab));
  c = translate(a,scale(ab,u));
  return dist(p,c);
}

//dist of p from line-segment(a,b).c:closest point to p on line-segment
double distToLineSegment(point p,point a,point b,point &c){
  vect ap(a,p),ab(a,b);double u = dot(ap,ab)/dot(ab,ab);
  if(u<0.0){c=a;return dist(p,a);}//closer to a
  if(u>1.0){c=b;return dist(p,b);}//closer to b
  return distToLine(p,a,b,c);//run dist to line as above
}

//returns angle aob in rad
double angle(point a,point o,point b){
  vect oa(o,a),ob(o,b);
  return acos(dot(oa,ob)/sqrt(dot(oa,oa)*dot(ob,ob)));
}

//returns true if point r is on left side of line pq
int ccw(point p,point q,point r){//>= to accept
  double k = cross(vect(p,q),vect(p,r));//colliner points
  if(k <= -EPS) return -1;
  else if(k >= +EPS) return +1;
  else return 0;
}

//returns true if point r is on same line as p,q
bool collinear(point p,point q,point r){
  return fabs(cross(vect(p,q),vect(p,r)))<EPS;
}

/************* Polygons ***************/
//Polygon is represented as vector of counter-clockwise points,
//with last point equal to first point.
double area(vector<point> &P){
  double res=0;
  for(int i=0;i<P.size()-1;i++)
    res+=(P[i].x*P[i+1].y-P[i+1].x*P[i].y);
  return fabs(res/2);
}

//Check if given polygon is Convex
bool isConvex(vector<point> &P){
  if(P.size()<=3)return false;
  bool isLeft=ccw(P[0],P[1],P[2]);
  for(int i=1;i<P.size()-1;i++)
    if(ccw(P[i],P[i+1],P[(i+2)==P.size()?1:i+2]) != isLeft)
      return false;
  return true;
}

//Returns true if point p is inside (concave/convex)P
bool inPolygon(point p,vector<point>& P){
  if(!P.size())return false;
  double sum=0;
  for(int i=0;i<P.size()-1;i++)
    if(ccw(p,P[i],P[i+1]))sum+=angle(P[i],p,P[i+1]);
    else sum-=angle(P[i],p,P[i+1]);
  return fabs(fabs(sum)-2*PI)<EPS;
}

/************* Convex Hull **************/
```

```
point pivot(0,0);
bool angleCmp(point a,point b){
  if(collinear(pivot,a,b))
    return dist(pivot,a)<dist(pivot,b);
  double d1x=a.x-pivot.x,d1y=a.y-pivot.y;
  double d2x=b.x-pivot.x,d2y=b.y-pivot.y;
  return (atan2(d1y,d1x)-atan2(d2y,d2x))<0;
}

//returns convex hull of polygon.
vector<point> ConvexHull(vector<point> P){
  int j,n=P.size();
  if(n<=3){if(!(P[0]==P[n-1]))P.push_back(P[0]);return P;}
  //Find P0=point with lowest Y and if tie:rightmost X
  int P0=0;
  for(int i=1;i<n;i++)
    if(P[i].y<P[P0].y||(P[i].y==P[P0].y&&P[i].x>P[P0].x))
      P0=i;
  swap(P[P0],P[0]);
  pivot=P[0];
  sort(P.begin()+1,P.end(),angleCmp);
  vector<point>S;
  S.push_back(P[n-1]);S.push_back(P[0]);S.push_back(P[1]);int i=2;
  while(i<n){
    //change ccw to accept collinear point if required
    j=S.size()-1;if(ccw(S[j-1],S[j],P[i]))S.push_back(P[i++]);
    else S.pop_back();
  }
  return S;
}
int main(){
	return 0;
}
```

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

Extended Euclid:

```
int xgcd(int a, int b, int &x, int &y) //Returns GCD of A, B
{
	if(a==0)
	{
		x=0;
		y=1;
		return b;
	}
	int x1, y1;
	int d = xgcd(b % a, a, x1, y1);
	x = y1 - (b/a)*x1;
```

```
    y = x1;
    return d;
}

int modular_inverse(int a, int m)
{
    int x, y;
    int g=xgcd(a, m, x, y);
    if(g!=1)
        return -1;
    else
    {
        x=(x%m + m)%m;
        return x;
    }
}

void shift_solution(int &x, int &y, int a, int b, int cnt)
{
    x+=cnt*b;
    y-=cnt*a;
}

bool find_any_solution(int a, int b, int c, int &x0, int &y0)
{
    int g=xgcd(abs(a), abs(b), x0, y0);
    if(c%g!=0)
        return false;
    x0 *= c/g;
    y0 *= c/g;
    if(a<0)
        x0*=-1;
    if(b<0)
        y0*=-1;
    return true;
}

int find_all_solutions(int a, int b, int c, int minx, int maxx, int miny, int maxy) //Returns number of solutions with x∈[minx, maxx], y∈[miny, maxy]
{
    int x, y, g;
    if(!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    int sign_a = a>0 ? +1 : -1;
    int sign_b = b>0 ? +1 : -1;
    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx) shift_solution(x, y, a, b, sign_b);
    if (x > maxx) return 0;
    int lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx) shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;

    shift_solution(x, y, a, b, - (miny - y) / a);
    if (y < miny) shift_solution(x, y, a, b, -sign_a);
    if (y > maxy) return 0;
    int lx2 = x;

    shift_solution(x, y, a, b, - (maxy - y) / a);
    if (y > maxy) shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap (lx2, rx2);
    int lx = max (lx1, lx2);
    int rx = min (rx1, rx2);

    return (rx - lx) / abs(b) + 1;
}
////////////////////////
```

Floyd Warshall

```
int dist[N][N];

void FloydWarshall()
{
    for(int k=1;k<=n;k++)
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                dist[i][j]=min(dist[i][j], dist[i][k] + dist[k][j]);
}
```

HASHING

```
struct Hashs
{
    vector<int> hashs;
    vector<int> pows;
    int P;
    int MOD;
    Hashs() {}

    Hashs(string &s, int P, int MOD) : P(P), MOD(MOD)
    {
        int n = s.size();
        pows.resize(n+1, 0);
        hashs.resize(n+1, 0);
        pows[0] = 1;
        for(int i=n-1;i>=0;i--)
        {
            hashs[i]=(1LL * hashs[i+1] * P + s[i] - 'a' + 1) % MOD;
            pows[n-i]=(1LL * pows[n-i-1] * P) % MOD;
        }
        pows[n] = (1LL * pows[n-1] * P)%MOD;
    }
    int get_hash(int l, int r)
    {
        int ans=hashs[l] + MOD - (1LL*hashs[r+1]*pows[r-l+1])%MOD;
        ans%=MOD;
        return ans;
    }
};
```

KMP

String:

```
vector<int> prefix_function(string &s)
{
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++)
    {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}

vector<int> find_occurences(string &text, string &pattern)
{
    string cur=pattern + '#' + text;
    int sz1=text.size(), sz2=pattern.size();
    vector<int> v;
    vector<int> lps=prefix_function(cur);
    for(int i=sz2+1;i<=sz1+sz2;i++)
    {
        if(lps[i]==sz2)
            v.push_back(i-2*sz2);
    }
    return v;
}
```

Vector:

```
vector<int> prefix_function(vector<int> &v)
{
    int n = (int)v.size();
    vector<int> pi(n);
    for (int i = 1; i < n; i++)
    {
        int j = pi[i-1];
        while (j > 0 && v[i] != v[j])
            j = pi[j-1];
        if (v[i] == v[j])
            j++;
        pi[i] = j;
    }
    return pi;
}

vector<int> find_occurences(vector<int> &text, vector<int> &pattern)
{
    vector<int> v=pattern;
    v.push_back(-1);
    for(auto &it:text)
        v.push_back(it);
    int sz1=text.size(), sz2=pattern.size();
    vector<int> lps=prefix_function(v);
    vector<int> store;
    for(int i=sz2+1;i<=sz1+sz2;i++)
    {
        if(lps[i]==sz2)
            store.push_back(i-sz*2);
    }
    return v;
}
////////////////////////
```

Topsort

```
int indeg[N];
vector<int> topo; //Stores lexicographically smallest toposort
vector<int> g[N];

bool toposort() //Returns 1 if there exists a toposort, 0 if there is a cycle
{
	priority_queue<int, vector<int>, greater<int> > pq;
	for(int i=1;i<=n;i++)
		for(auto &it:g[i])
			indeg[it]++;
	for(int i=1;i<=n;i++)
	{
		if(!indeg[i])
			pq.push(i);
	}
	while(!pq.empty())
	{
		int u=pq.top();
		pq.pop();
		topo.push_back(u);
		for(auto &v:g[u])
		{
			indeg[v]--;
			if(!indeg[v])
				pq.push(v);
		}
	}
	if(topo.size()<n)
		return 0;
	return 1;
}
```

LCA

```
int tim=0;
int parent[LG][N];
int tin[N], tout[N], level[N];

void dfs(int k, int par, int lvl)
{
	tin[k]=++tim;
	parent[0][k]=par;
	level[k]=lvl;
	for(auto it:g[k])
	{
		if(it==par)
			continue;
		dfs(it, k, lvl+1);
	}
	tout[k]=tim;
}

int walk(int u, int h)
{
	for(int i=LG-1;i>=0;i--)
	{
		if((h>>i) & 1)
			u = parent[i][u];
	}
	return u;
}

void precompute()
{
	for(int i=1;i<LG;i++)
		for(int j=1;j<=n;j++)
			if(parent[i-1][j])
				parent[i][j]=parent[i-1][parent[i-1][j]];
}

int LCA(int u, int v)
{
	if(level[u]<level[v])
		swap(u,v);
	int diff=level[u]-level[v];
	for(int i=LG-1;i>=0;i--)
	{
		if((1<<i) & diff)
		{
			u=parent[i][u];
		}
	}
	if(u==v)
		return u;
	for(int i=LG-1;i>=0;i--)
	{
		if(parent[i][u] && parent[i][u]!=parent[i][v])
		{
			u=parent[i][u];
			v=parent[i][v];
		}
	}
	return parent[0][u];
}

int dist(int u, int v)
{
	return level[u] + level[v] - 2 * level[LCA(u, v)];
}
```

Matrix Multiplication:

```
/*input
10 2 1
*/
#include <bits/stdc++.h>
using namespace std;

#define int long long
#define pii pair<int,int>
#define pb push_back
#define f first
#define s second
#define IOS ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
int const SZ = 2;
int MOD = 1e6;
int add(int a, int b)
{
	int res = a + b;
	if(res >= MOD)
		return res - MOD;
	return res;
}

int mult(int a, int b)
{
	long long res = a;
	res *= b;
	if(res >= MOD)
		return res % MOD;
	return res;
}

struct matrix
{
	int arr[SZ][SZ];

	void reset()
	{
		memset(arr, 0, sizeof(arr));
	}

	void makeiden()
	{
		reset();
```

```
        for(int i=0;i<SZ;i++)
        {
            arr[i][i] = 1;
        }
    }

    matrix operator + (const matrix &o) const
    {
        matrix res;
        for(int i=0;i<SZ;i++)
        {
            for(int j=0;j<SZ;j++)
            {
                res.arr[i][j] = add(arr[i][j], o.arr[i][j]);
            }
        }
        return res;
    }

    matrix operator * (const matrix &o) const
    {
        matrix res;
        for(int i=0;i<SZ;i++)
        {
            for(int j=0;j<SZ;j++)
            {
                res.arr[i][j] = 0;
                for(int k=0;k<SZ;k++)
                {
                    res.arr[i][j] = add(res.arr[i][j] , mult(arr[i][k] , o.arr[k][j]));
                }
            }
        }
        return res;
    }
};

matrix power(matrix a, int b)
{
    matrix res;
    res.makeiden();
    while(b)
    {
        if(b & 1)
        {
            res = res * a;
        }
        a = a * a;
        b >>= 1;
    }
    return res;
}
signed main() {
    IOS;
    int n,l,k;
    cin>>n>>k>>l;
    matrix mat;
    n/=5;
    mat.reset();
    k%=MOD;
    l%=MOD;
    mat.arr[0][0]=k;
    mat.arr[0][1]=l;

    mat.arr[1][0]=1;
    matrix mat2;
    mat2.reset();
    mat2.arr[0][0]=k;
    mat2.arr[1][0]=1;
    matrix res;
    res = power(mat,n-1);
    res = res * mat2;
    int ans = res.arr[0][0] + res.arr[0][1];
    ans%=MOD;
    string s = to_string(ans);
    string t = s;
    reverse(t.begin(), t.end());
    while(t.size()  < 6 ){
        t.pb('0');
    }
    reverse(t.begin(), t.end());
    cout << t << endl;
    return 0;
}
```

MAX FLOW:

```
// Adjacency list implementation of Dinic's blocking flow algorithm.
// This is very fast in practice, and only loses to push-relabel flow.
//
// Running time:
//     O(|V|^2 |E|)
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source and sink
//
// OUTPUT:
//     - maximum flow value
//     - To obtain actual flow values, look at edges with capacity > 0
//       (zero capacity edges are residual edges).

#include<cstdio>
#include<vector>
#include<queue>
using namespace std;
typedef long long LL;

struct Edge {
  int u, v;
  LL cap, flow;
  Edge() {}
  Edge(int u, int v, LL cap): u(u), v(v), cap(cap), flow(0) {}
};

struct Dinic {
  int N;
  vector<Edge> E;
  vector<vector<int>> g;
  vector<int> d, pt;

  Dinic(int N): N(N), E(0), g(N), d(N), pt(N) {}

  void AddEdge(int u, int v, LL cap) {
    if (u != v) {
      E.emplace_back(u, v, cap);
      g[u].emplace_back(E.size() - 1);
      E.emplace_back(v, u, 0);
      g[v].emplace_back(E.size() - 1);
    }
  }

  bool BFS(int S, int T) {
    queue<int> q({S});
    fill(d.begin(), d.end(), N + 1);
    d[S] = 0;
    while(!q.empty()) {
      int u = q.front(); q.pop();
      if (u == T) break;
      for (int k: g[u]) {
        Edge &e = E[k];
        if (e.flow < e.cap && d[e.v] > d[e.u] + 1) {
          d[e.v] = d[e.u] + 1;
          q.emplace(e.v);
        }
      }
    }
    return d[T] != N + 1;
  }

  LL DFS(int u, int T, LL flow = -1) {
    if (u == T || flow == 0) return flow;
    for (int &i = pt[u]; i < g[u].size(); ++i) {
      Edge &e = E[g[u][i]];
      Edge &oe = E[g[u][i]^1];
      if (d[e.v] == d[e.u] + 1) {
        LL amt = e.cap - e.flow;
        if (flow != -1 && amt > flow) amt = flow;
        if (LL pushed = DFS(e.v, T, amt)) {
          e.flow += pushed;
          oe.flow -= pushed;
          return pushed;
        }
      }
    }
    return 0;
  }

  LL MaxFlow(int S, int T) {
    LL total = 0;
    while (BFS(S, T)) {
      fill(pt.begin(), pt.end(), 0);
      while (LL flow = DFS(S, T))
        total += flow;
    }
    return total;
  }
};

// BEGIN CUT
// The following code solves SPOJ problem #4110: Fast Maximum Flow (FASTFLOW)

int main()
{
  int N, E;
  scanf("%d%d", &N, &E);
  Dinic dinic(N);
  for(int i = 0; i < E; i++)
  {
    int u, v;
    LL cap;
    scanf("%d%d%lld", &u, &v, &cap);
    dinic.AddEdge(u - 1, v - 1, cap);
    dinic.AddEdge(v - 1, u - 1, cap);
  }
  printf("%lld\n", dinic.MaxFlow(0, N - 1));
  return 0;
}

// END CUT
```

/////////

Max Matching:

```
vector<int> v[1001];
bool vis[1001];
int previous[1001];
bool match(int i){
        if(i == -1) return 1;
        if(vis[i]) return 0;
        vis[i]=1;
        for(auto x : v[i]){
                if(match(previous[x]))
                {
                        previous[x]=i;
                        return 1;
                }
        }
        return 0;
}
signed main() {
        IOS;
        int n;
        cin>>n;
        for(int i=0;i<n;i++){
                previous[i]=-1;
                int k;
                cin>>k;
                for(int j=0;j<k;j++){
                        int y;
                        cin>>y;
                        v[i].pb(y);
                }
        }
        int matchings = 0 ;
        for(int i=0;i<n;i++){
                memset(vis,0,sizeof(vis));
                if(match(i)) matchings++;
        }
        cout<<n-matchings;
```

Priority Queue

```
class ComparisonClass{
public:
bool operator() (pair<int,int> a, pair<int,int> b) {
        return a.s<b.s;
    }
};
signed main() {
  IOS;
  priority_queue<pii,vector<pii>,ComparisonClass> q;
  q.push({10,1});
  q.push({20,5});
  q.push({30,3});
```

TRIE XOR

```
int  curxor = 0 ;
struct Trienode{
        struct Trienode* bits[2];
        int sum;
};
struct Trienode* newnode(int val){
        struct Trienode* temp = new Trienode;
        temp->bits[0] = NULL;
        temp->bits[1] = NULL;
        temp->sum = val;
        return temp;
}
void insert(struct Trienode* root,int num){
        struct Trienode* pCrawl = root;
        for(int i = 33 ; i >=0 ;i--){
                if((1LL<<i) & num){
                        if(pCrawl->bits[1]==NULL){
                                pCrawl->bits[1] = newnode(0);
                        }
                        pCrawl = pCrawl->bits[1];
                }
                else{
                        if(pCrawl->bits[0]==NULL){
                                pCrawl->bits[0] = newnode(0);
                        }
                        pCrawl = pCrawl->bits[0];
                }
                pCrawl->sum++;
        }

}
int lol(struct Trienode* root){
       if(root){
             return root->sum;
       }
       return 0;
}
int get(struct Trienode* root,int prefix,int k){
       struct Trienode* pCrawl = root;
       int ans = 0;
       for(int i = 33 ; i>=0 ; i--){
             if(!pCrawl)break;

             int PB = (prefix >> i) & 1LL;
    int bit = (k >> i) & 1LL;
    if (PB == bit) {
      if (PB == 1) {
        ans += lol(pCrawl->bits[1]);
      }
      pCrawl = pCrawl->bits[0];
    }
    else {
      if (PB == 0) {
        ans += lol(pCrawl->bits[0]);
      }
      pCrawl = pCrawl->bits[1];
    }
       }
       return ans;
}
```