CODE SHEET FOR TEAM CONTEST

BUET_TOO_HARD_TO_SOLVE

Members

Latif Siddiq Sunny | Mohammad Anas | Masud Rana

Number Theory

1.Sieve

```
#define M 1000000
bool marked[M];

bool isPrime(int n) {
  if (n < 2) return false;
  if (n == 2) return true;
  if (n % 2 == 0) return false;
  return marked[n] == false;
}

void sieve(int n) {
  for (int i = 3; i * i <= n; i += 2) {
    if (marked[i] == false) { // i is a prime
      for (int j = i * i; j <= n; j += i + i) {
        marked[j] = true;
      }
    }
  }
}
```

2.Memory efficient sieve

```
#define M 100000000
int marked[M/64 + 2];

#define on(x) (marked[x/64] & (1<<((x%64)/2)))
#define mark(x)  marked[x/64] |= (1<<((x%64)/2))

void sieve(int n) {
  for (int i = 3; i * i < n; i += 2) {
    if (!on(i)) {
      for (int j = i * i; j <= n; j += i + i) {
        mark(j);
      }
    }
  }
}

bool isPrime(int num) {
  return num > 1 && (num == 2 || ((num & 1) && !on(num)));
```

```
}
```

3.Divisor count

```
vector<int> primes; // we'll preload primes once at the beginning
int countDivisor(int n) {
  int divisor = 1;
  for (int i = 0; i*i<=n; i++) {
    if (n % primes[i] == 0) {
      int cnt = 1;
      while (n % primes[i] == 0) {
        n /= primes[i];
        cnt++;
      }
      divisor *= cnt;
    }
  }
  return divisor;
}
```

4.GCD

```
long long int gcd(long long int a,long long int b) {
  return b == 0 ? a : gcd(b, a % b); // return __gcd(a,b);
}
```

5.LCM

```
long long int gcd(long long int a,long long int b) {
  return (a / gcd(a, b)) * b;
}
```

6. Euler totient function

```
int phi (int n) {
  int ret = n;
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      while (n % i == 0) {
        n /= i;
      }
      ret -= ret / i;
    }
  }
```

```
  // this case will happen if n is a prime number
  // in that case we won't find any prime that divides n
  // that's less or equal to sqrt(n)
  if (n > 1) ret -= ret / n;
  return ret;
}
```

7.Optimized eular function

```
#define M 1000005
int phi[M];

void calculatePhi() {
  for (int i = 1; i < M; i++) {
    phi[i] = i;
  }
  for (int p = 2; p < M; p++) {
    if (phi[p] == p) { // p is a prime
      for (int k = p; k < M; k += p) {
        phi[k] -= phi[k] / p;
      }
    }
  }
}
```

8. Extended Euclid

```
typedef pair<int, int> pii;
#define x first
#define y second

pii extendedEuclid(int a, int b) { // returns x, y | ax + by = gcd(a,b)
  if(b == 0) return pii(1, 0);
  else {
    pii d = extendedEuclid(b, a % b);
    return pii(d.y, d.x - d.y * (a / b));
  }
}
```

9. Modular Inverse

```
int modularInverse(int a, int n) {
  pii ret = extendedEuclid(a, n);
  return ((ret.x % n) + n) % n;
}
```

10. Big Mod

```
long long bigmod ( long long a, long long p, long long m ) //Sunny
{
    long long res = 1;
    long long x = a;
    while ( p )
    {
        if ( p & 1 ) //p is odd
        {
            res = ( res * x ) % m;
        }
        x = ( x * x ) % m;
        p = p >> 1;
    }
    return res;
}
/// for int , by Anas
template <typename T>
T mod(T a, T b, T c)
{
    if(b==0) return 1;
    if(b%2==0)
    {
        T x=mod(a,b/2,c);
        return (x%c*x%c)%c; /// to avoid overflow as may x >= 10^10
    }
    else return (a%c * mod(a,b-1,c))%c;
}

/// for upto 10^18 by Anas
template <typename T>
T mmul(T a, T b, T m) {
    a %= m;
    T result = 0;
    while (b) {
        if (b % 2) result = (result + a) % m;
        a = (a + a) % m;
        b /= 2;
    }
    return result;
}

template <typename T>
T mpow(T a, T b, T m) {
    a %= m;
    T result = 1;
    while (b) {
        if (b % 2) result = mmul(result, a, m);
```

```
        a = mmul(a, a, m);
        b /= 2;
    }
    return result;
}
```

11. Sum of divisors

```
#define MOD 1e9+7
int sod(long long int n)
{

    long long sum=1;
    long long nowsum=0;
    long long rt=sqrtl(n);
    for (int i = 0; primes[i]<=rt; i++)
    {
        if (n % primes[i] == 0)
        {   //cout<<primes[i]<<endl;
            int cnt = 0;
            while (n % primes[i] == 0)
            {
                n /= primes[i];
                cnt++;
            }
            rt=sqrtl(n);
           //cout<<cnt<<endl;
            long long x=bigmod(primes[i],(cnt)+1,MOD);
            long long y=modularInverse(primes[i]-1,MOD);
            nowsum=(((x-1+MOD)%MOD)*(y%MOD))%MOD;
            sum=((sum%MOD)*(nowsum%MOD))%MOD;

        }

    }
    if(n>1)
    {
        long long x=bigmod(n,1);
        long long y=modularInverse(n-1);
        nowsum=(((x-1+MOD)%MOD)*(y%MOD))%MOD;
        sum=((sum%MOD)*(nowsum%MOD))%MOD;
    }
    return sum;
}
```

Graph Theory

1.BFS DFS

```
void bfs(int source)
        {
        color[source]=1;
        dist[source]=0;
        queue<int>q;
        q.push(source);
        while(!q.empty())
        {
                int u=q.front();
                q.pop();
                for(int i=0; i<g[u].size(); i++)
                {
                int v=g[u][i];
                if(color[v]==0)
                {
                      dist[v]=dist[u]+1;
                        color[v]=1;
                        q.push(v);
                        parent[v]=u;
                }
                }
                color[u]=2;
        }

}
```

2. Dijkstra

```
#include <bits/stdc++.h>
using namespace std;
#define pii pair<int,int>
#define pb push_back
#define INF 0x7fffffff
#define sz 100005
priority_queue<pii,vector<pii>,greater<pii> >q;
vector<pii>G[sz];
int dist[sz];
bool vis[sz];
int parent[sz];
int n,m;
```

```
void printpath(int src)
{
        if(parent[src]!=-1)
        printpath(parent[src]);
        cout<<src<<" ";
}
void dijktras(int src)
{
        for(int i=0; i<=n; i++)
        {
        dist[i]=INF;
        vis[i]=false;
        parent[i]=-1;
        }
        dist[src]=0;
        q.push(pii(0,src));
        while(!q.empty())
        {
        int s=q.top().second;
        q.pop();
        for(int i=0; i<G[s].size(); i++)
        {
                int v=G[s][i].second;
                int w=G[s][i].first;
                if(!vis[v] && dist[s]+w<dist[v])
                {
                dist[v]=dist[s]+w;
                q.push(pii(dist[v],v));
                parent[v]=s;
                }
        }
        vis[s]=true;
        }
}
```

3. Articulation Point

```
#define mx 100005

vector<int>g[mx];
int visited[mx];
bool articulation[mx];
int low[mx],d[mx],parent[mx];
int t;

void articulating(int s,bool root)
{
        t++;
        low[s]=d[s]=t;
```

```
        visited[s]=1;
        int child=0;

        for(int i=0; i<g[s].size(); i++)
        {
        int v=g[s][i];

        if(visited[v])
        {
               low[s]=min(low[s],d[v]);
        }
        if(!visited[v])
        {
               parent[s]=v;
               articulating(v,false);

               if(d[s]<=low[v]&&!root)
               {
               articulation[s]=true;
               }
               low[s]=min(low[s],low[v]);
               child++;
        }
        if(child>1&&root)
        {
               articulation[s]=true;
        }

        }

}
void setting()
{
        for(int i=0; i<mx; i++)
        {
        g[i].clear();
        visited[i]=articulation[i]=low[i]=d[i]=parent[i]=0;
        t=0;
        }
}
```

4. Krushkal

```
#define mx 500005

int id[mx],n,m;
pair <long long, pair<int, int> > p[mx];
```

```
void initialize()
{
        for(int i = 0; i < mx; ++i)
        id[i] = i;
}

int root(int x)
{
        while(id[x] != x)
        {
        id[x] = id[id[x]];
        x = id[x];
        }
        return x;
}

void union1(int x, int y)
{
        int p = root(x);
        int q = root(y);
        id[p] = id[q];
}

long long kruskal(int k)
{
        int x, y;
        long long cost, minimumCost = 0;
        initialize();
        for(int i = 0; i < k; ++i)
        {
        x = p[i].second.first;
        y = p[i].second.second;
        cost = p[i].first;
        if(root(x) != root(y))
        {

             minimumCost += cost;
                union1(x, y);

        }
        }
        return minimumCost;
}
```

5. Prims

```
#define mx 50005
typedef pair<long long int,int> PII;
```

```
vector<PII>g[mx];
bool marked[mx];
long long prim(int x)
{
        priority_queue<PII, vector<PII>, greater<PII> > Q;
        int y;
        long long minimumCost = 0;
        PII p;
        Q.push(make_pair(0, x));
        while(!Q.empty())
   {

         p = Q.top();
        Q.pop();
        x = p.second;
        // Checking for cycle
        if(marked[x] == true)
                continue;
        minimumCost += p.first;
        marked[x] = true;
        for(int i = 0; i < g[x].size(); ++i)
        {
                y = g[x][i].second;
                if(marked[y] == false)
                Q.push(g[x][i]);
        }
        }
        return minimumCost;
}
```

6. Floyd

```
#define mx 505

long long matrix[mx][mx];

int main()
{
        memset(matrix,INT_MAX,sizeof matrix);

        int n;
        scanf("%d",&n);
        for(int i=1; i<=n; i++)
        {
        for(int j=1; j<=n; j++)
        {
                scanf("%d",&matrix[i][j]);
        }
```

```
        }
        for(int k=1; k<=n; k++)
        {
        for(int i=1; i<=n; i++)
        {
                for(int j=1; j<=n; j++)
                {
                if ((matrix[i][k] + matrix[k][j] )< matrix[i][j])
                {
                        matrix[i][j] = matrix[i][k] + matrix[k][j];

                 }

                }
           }
        }


}
```

7.Bellman ford

```
int inf=INT_MAX;
vector< pair<int,int> >edge;
int w[205][205];
int b[205];
int dist[205];

int n,m;

void bellman(int s)
{
        dist[s]=0;
        for(int i=1; i<=n-1; i++)
        {
        for(int j=0; j<edge.size(); j++)
        {
                int x=edge[j].first;
                int y=edge[j].second;
                int z=w[x][y];
                //cout<<z<<endl;
                if( (dist[x] + z) < dist[y])
                {
                dist[y] = dist[x] + z;
                }
        }
        }
        for(int j=0; j<m; j++)
```

```
        {
        int x=edge[j].first;
        int y=edge[j].second;
        int z=w[x][y];
        if((dist[x] + z) < dist[y] )
        {

           printf("Negative cycle found\n");
        }
        }


}
```

8. Max flow

```
#define mx 105
int g[mx][mx],rgraph[mx][mx];
int n,source,sink,m;
int vis[mx],par[mx];
bool bfs()
{
    memset(vis,0,sizeof vis);
    //memset(par,0,sizeof par);
    queue< int> Q;
    vis[source]=1;
    Q.push(source);
    par[source]=source;
    bool isPath= false;


    while(!Q.empty())
    {
        int u = Q.front();
        Q.pop();
        if(u==sink)
        {
            isPath=true;
            break;
        }
        for(int i=1; i<=n; i++)
        {
            if(vis[i]==0 && rgraph[u][i]>0)
            {
       par[i]=u;
              vis[i]=1;
 Q.push(i);
```

```
            }
}


    }

    return isPath;
}
int Ford_Fulkerson()
{
    int max_flow=0;
    while(bfs())
    {
        int v=sink;
        int path_flow = INT_MAX;
        for (; v != source; v = par[v])
        {
            int  u = par[v];
            path_flow = min(path_flow, rgraph[u][v]);
        }
        v = sink;
        for (; v != source; v = par[v])
        {
            int u = par[v];
            rgraph[u][v] -= path_flow;
            rgraph[v][u] += path_flow;
        }
        max_flow += path_flow;
    }
    return max_flow;

}
```

9. Heavy Light Decomposition

```
///http://lightoj.com/volume_showproblem.php?problem=1348
#include<bits/stdc++.h>

using namespace std;

#define ll long long int
#define db(x) cout<<#x<<" -> "<<x<<endl
#define maxn 30005
#define xx first
#define yy second
#define mp make_pair
```

```
typedef pair< int , int > pii;

int sub[maxn],d[maxn],par[maxn];
bool vis[maxn];
int P[maxn][17];
vector< int > g[maxn];
int st[maxn*6],qt[maxn*6];
int chainHead[maxn];
int chainInd[maxn],posInBase[maxn],baseArray[maxn];
int Cost[maxn],otherEnd[maxn];
int n,chainNo=0,ptr=0;
int root ;

void clr(){
    memset(vis,false,sizeof(vis));
    memset(chainHead,-1,sizeof(chainHead));
    chainNo = 0;
    ptr  = 0;
    root = 0;
    for(int i=0; i<=n+5; i++){
        g[i].clear();
    }
}

void dfs(int src, int parent, int dep){
    par[src] =  parent;
    vis[src] =  true;
    d[src]   =  dep;
    sub[src] =  1;
    for(int i=0; i<g[src].size(); i++){
        int temp = g[src][i];
        if( vis[temp] )  continue;
//        otherEnd[index[src][i]] = temp;
        dfs(temp,src,dep+1);
        sub[src]+=sub[temp];
    }
}

int lca_query(int p, int q){
    if( d[p] < d[q] ) swap(p,q);
    int log = 1;
    while(true){
        int next = log+1;
        if((1<<next)>d[p]) break;
        log++;
    }
    for(int i=log; i>=0; i--){
        if((d[p]-(1<<i))>=d[q]){
            p = P[p][i];
        }
```

```
    }
    if(p==q) return p;
    for(int i=log; i>=0; i--){
        if(P[p][i]!=-1 && P[p][i]!=P[q][i]){
            p = P[p][i] ; q = P[q][i];
        }
    }
    return par[p];
}

void lca_init(){
    memset(P,-1,sizeof(P));
    for(int i=0; i<n; i++){
        P[i][0] = par[i];
    }
    for(int j=1; (1<<j)<n; j++){
        for(int i=0; i<n; i++){
            if( P[i][j-1]== -1 ) continue;
            P[i][j] = P[P[i][j-1]][j-1];
        }
    }
}

void HLD(int curNode, int cost , int prev){
    if(chainHead[chainNo]==-1){
        chainHead[chainNo] = curNode;
    }
    chainInd[curNode] = chainNo;
    posInBase[curNode] = ptr;
    baseArray[ptr++] = cost;
    int sc = -1;
    int ncost;
    ///Loop to find special child
    for(int i=0; i<g[curNode].size(); i++){
        int temp = g[curNode][i];
        if(temp == prev) continue;
        if(sc==-1 || sub[sc]<sub[temp]){
            sc = temp;
            ncost = Cost[g[curNode][i]];
        }
    }
    if(sc!=-1){
        ///Expand the chain;
        HLD(sc,ncost,curNode);
    }
    for(int i=0; i<g[curNode].size(); i++){
        int temp = g[curNode][i];
        if(temp == prev) continue;
        if(sc!=temp){
            ///New chain at each normal node;
```

```
            chainNo++;
            HLD(temp,Cost[temp],curNode);
        }
    }
}

///make segment tree it uses the baseArray for building segment tree
void make_tree(int cur, int s, int e){
    if(s==e-1){
        st[cur] = baseArray[s];
        return ;
    }
    int c1 = (cur<<1) , c2 = c1|1 , m = (s+e)>>1;
    /// in [s,e) range so (s,m) & (m,e) in make_tree
    make_tree(c1,s,m);
    make_tree(c2,m,e);
    st[cur] = st[c1]+st[c2];
}

///point update . Update a single element of the segment tree
void update_tree(int cur, int s, int e, int x, int val){
   // printf("%d %d %d %d %d\n",cur,s,e,x,val);
    if(s>x || e<=x) return ;
    if(s==x && s==e-1){
        st[cur] = val;
//        printf("Cur: %d , val: %d\n",cur,val);
        return;
    }
    int c1 = (cur<<1),c2 = c1|1,m = (s+e)>>1;
    update_tree(c1,s,m,x,val);
    update_tree(c2,m,e,x,val);
    st[cur] = st[c1]+st[c2];
}

///query in the range [s,e)
void query_tree(int cur, int s, int e, int S, int E){
    if(s >= E || e <= S) {
            qt[cur] = 0;
            return;
       }
       if(s >= S && e <= E) {
              qt[cur] = st[cur];
              return;
       }
       int c1 = (cur<<1),c2=c1|1,m = (s+e)>>1;
       query_tree(c1, s, m, S, E);
       query_tree(c2, m, e, S, E);
       qt[cur] = qt[c1]+qt[c2];
}
// * query_up:
```

```
// * It takes two nodes u and v, condition is that v is an ancestor of u
// * We query the chain in which u is present till chain head, then move to next chain up
// * We do that way till u and v are in the same chain, we query for that part of chain and
break

int query_up(int u, int v){
    if(u==v){ return 0; }
    int uchain,vchain=chainInd[v],ans = 0;
//      uchain & vchain are the chain numbers of u &v respectively!
    while(true){
        uchain = chainInd[u];
//        db(ans);
        if(uchain==vchain){
            if(u==v) break;
//      Both u and v are in the same chain, so we need to query from u to v, update answer
and break.
//      We break because we came from u up till v, we are done
           query_tree(1,0,ptr,posInBase[v]+1,posInBase[u]+1);
           ans+=qt[1];
           break;
        }
        query_tree(1,0,ptr,posInBase[chainHead[uchain]],posInBase[u]+1);
//      Above is call to segment tree query function. We do from chainHead of u till u. That
is the whole chain from
        ans+=qt[1];
        u = chainHead[uchain];
        u = par[u];
    }
    return ans;
}

int query(int u, int v){
    int lca = lca_query(u,v);
    int x = query_up(u,lca);
    int y = query_up(v,lca);
    return (x+y)+Cost[lca];
}

/*
 * change:
 * We just need to find its position in segment tree and update it
 */

void change(int i, int val){
//    int u = otherEnd[i];
    update_tree(1,0,ptr,posInBase[i],val);
}

int main(){
    ios_base::sync_with_stdio(false);
```

```
    cin.tie(0);
    int t;
    scanf("%d",&t);
    int tc  = 0;
    memset(chainHead,-1,sizeof(chainHead));
    while(t--){
        scanf("%d",&n);
        for(int i=0; i<n; i++){
            scanf("%d",&Cost[i]);
        }
        for(int i=1; i<=n-1; i++){
            int u,v;
            scanf("%d %d",&u,&v);
            g[u].push_back(v);
            g[v].push_back(u);
//            index[u].push_back(i-1);
//            index[v].push_back(i-1);
        }
        d[0] = 0;
        dfs(0,-1,0);
        lca_init();
        HLD(0,Cost[0],-1);
        make_tree(1,0,ptr);
        printf("Case %d:\n",++tc);
        int q;
        scanf("%d",&q);
        while(q--){
            int a,b,c;
            scanf("%d %d %d",&a,&b,&c);
            if(a==0){
               int ans = query(b,c);
               printf("%d\n",ans);
            }
            else{
                Cost[b] = c;
                change(b,c);
            }
        }
        clr();
    }
    return 0;
}

/**
2
4
10 20 30 40
0 1
1 2
1 3
```

```
3
0 2 3
1 1 100
0 2 3
4
10 20 30 40
0 1
1 2
1 3
3
0 2 3
1 1 100
0 2 3
*/
```

Geometry

1.Template

```
#include <bits/stdc++.h>
using namespace std;
#define PI          acos(-1.0)
#define EPS         1e-9
#define Z(x)        fabs(x)<EPS
#define E(x,y)      Z(x-y)
#define RAD(x)      (x)*PI/180
#define PRE(x)      ((x)==0?n-1:x-1)
#define NEX(x)      ((x)==n-1?0:x+1)
int dcmp(double x) { return x > EPS ? 1: x < -EPS ? -1 : 0; }
struct point {
    double  x,y;
    point(double x=0, double y=0) : x(x), y(y) {}
    point(const point &p) : x(p.x), y(p.y)    {}
    bool operator <  (const point &p) const { return (x<p.x)||(E(x,p.x) && y < p.y );}
    bool operator == (const point &p) const { return E(x,p.x)&&E(y,p.y);    }
    point operator + (const point &p) const { return point(x+p.x , y + p.y);}
    point operator - (const point &p) const { return point(x-p.x , y - p.y);}
    point operator * (double c) const { return point(x*c,y*c);}
    point operator / (double c) const { return point(x/c,y/c);}
    void input(){scanf("%lf %lf",&x,&y);}
};

double dot(point p,point q){ return p.x * q.x + p.y * q.y;}
double cross(point p,point q) {return p.x*q.y - p.y*q.x;}
```

```
double sqdist(point p,point q) {return dot(p-q , p-q);}
double dist(point p,point q)  { return sqrt(sqdist(p,q));}
double mag(point p) {return sqrt(p.x*p.x+p.y*p.y);}
double angle(point a,point b) { return acos(dot(a,b)/mag(a)/mag(b));}

// rotate a point cross or CW around the origin
point rotccw90(point p){return point(-p.y,p.x);}
point rotcw90(point p)   {return point(p.y,-p.x);}
point rotccw(point p,double ang) {return point(p.x*cos(ang) - p.y*sin(ang) , p.x*sin(ang) +
p.y * cos(ang));}

// project point c onto line through a and b
point PPL (point a,point b,point c){return a + (b-a)* dot(c-a,b-a) / dot(b-a,b-a);}

// project point c onto line segment through a and b
point PPS(point a, point b, point c) {
    double r = dot(b-a,b-a);
    if (fabs(r) < EPS) return a;
    r = dot(c-a, b-a)/r;
    if (r < 0) return a;
    if (r > 1) return b;
    return a + (b-a)*r;
}
// determine if lines from a to b and c to d are parallel or collinear
bool IsLP(point a, point b, point c, point d) {
    return fabs(cross(b-a, c-d)) < EPS;
}

bool IsLC(point a, point b, point c, point d) {
    return IsLP(a, b, c, d)
    && fabs(cross(a-b, a-c)) < EPS
    && fabs(cross(c-d, c-a)) < EPS;
}
// Checks if p lies on the segment connection ab
bool onSegment(point a, point b, point p)
{
    return Z(cross(a-p,b-p)) && dot(a-p,b-p) < 0;
}
// determine if line segment from a to b intersects with
// line segment from c to d
bool SGIN(point a, point b, point c, point d) {
    if (IsLC(a, b, c, d)) {
        if (sqdist(a, c) < EPS || sqdist(a, d) < EPS ||
            sqdist(b, c) < EPS || sqdist(b, d) < EPS) return true;
        if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
            return false;
        return true;
    }
    if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
    if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
```

```
    return true;
}


point LLIN(point a, point b, point c, point d) {
    b=b-a; d=c-d; c=c-a;
    return a + b*cross(c, d)/cross(b, d);
}

// compute center of circle given three points
point CCC(point a, point b, point c) {
    b=(a+b)/2;
    c=(a+c)/2;
    return LLIN(b, b+rotcw90(a-b), c, c+rotcw90(a-c));
}

//point in polygon(1-Strictly Interior,0-Strictly Exterior
bool PIPoly(const vector<point> &p, point q) {
    bool c = 0;
    for (int i = 0; i < p.size(); i++){
        int j = (i+1)%p.size();
        if (((p[i].y <= q.y && q.y < p[j].y) ||
             (p[j].y <= q.y && q.y < p[i].y)) &&
            q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y))
            c = !c;
    }
    return c;
}

// determine if point is on the boundary of a polygon
bool POPoly(const vector<point> &p, point q) {
    for (int i = 0; i < p.size(); i++)
        if (sqdist(PPS(p[i], p[(i+1)%p.size()], q), q) < EPS)
            return true;
    return false;
}

// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<point> CLIN(point a, point b, point c, double r) {
    vector<point> ret;
    b = b-a;
    a = a-c;
    double A = dot(b, b);
    double B = dot(a, b);
    double C = dot(a, a) - r*r;
    double D = B*B - A*C;
    if (D < -EPS) return ret;
    ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
    if (D > EPS)
```

```
        ret.push_back(c+a+b*(-B-sqrt(D))/A);
    return ret;
}

// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<point> CCIN(point a, point b, double r, double R) {
    vector<point> ret;
    double d = sqrt(sqdist(a, b));
    if (d > r+R || d+min(r, R) < max(r, R)) return ret;
    double x = (d*d-R*R+r*r)/(2*d);
    double y = sqrt(r*r-x*x);
    point v = (b-a)/d;
    ret.push_back(a+v*x + rotccw90(v)*y);
    if (y > 0)
        ret.push_back(a+v*x - rotccw90(v)*y);
    return ret;
}
// p = a + t * (b-a). find t.
double getT(point a, point dir, point p)
{
      if(dcmp(dir.x) == 0) return (p.y - a.y) / dir.y;
      return (p.x - a.x)/ dir.x;
}
// Gives area of a circle sector passing through a,b
double SectorArea(point r, point a, point b, double R)
{
      double ang = angle(a-r,b-r);
      return R*R*ang/2;
}
// Common area of a circle and and a segment(a,b)
double TRICA(point r, point a, point b, double R)
{
      double ra = dist(r, a) , rb = dist(r, b);
      if(ra < R + EPS && rb < R + EPS) return cross(a - r, b - r) / 2;
      if(dcmp(cross(a-r,b-r)) == 0) return 0;
      double rtos = dist(r, PPS(a,b,r));
      if(rtos > R - EPS) return SectorArea(r, a, b, R);
      vector< point > ins = CLIN(a,b,r,R);
      if(ins.size() < 2) return SectorArea(r, a, b, R);

      point ta = ins[0], tb = ins[1];
      double t1 = getT(a, b - a, ta) , t2 = getT(a, b-a, tb);
      if(t1 > t2) swap(ta, tb);

      if(ra < R + EPS) return cross(a - r, tb - r) / 2 + SectorArea(r, tb , b,R);
      if(rb < R + EPS) return cross(ta - r, b - r) / 2 + SectorArea(r , a, ta ,R);
      return cross(ta - r, tb - r) / 2 + SectorArea(r, a, ta, R) + SectorArea(r, tb, b,R);
}
// Simple polygon intersection area with a circle.
```

```
double SPICA(vector< point > &p, point r, double R )
{
      double ret = 0;
      int n = p.size();
      for(int i= 0; i < n; i ++) {
            int turn = dcmp(cross(p[i] - r, p[NEX(i)] -  r));
            if(turn > 0 ) ret += TRICA(r, p[i] , p[NEX(i)],R);
            else ret -= TRICA(r,p[NEX(i)], p[i],R);

      }
      return fabs(ret);

}
// This code computes the area or centroid of a (possibly nonconvex)
// polygon, assuming that the coordinates are listed in a clockwise or
// counterclockwise fashion.  Note that the centroid is often known as
// the "center of gravity" or "center of mass".

double ComputeSignedArea(const vector<point> &p) {
    double area = 0;
    for(int i = 0; i < p.size(); i++) {
        int j = (i+1) % p.size();
        area += p[i].x*p[j].y - p[j].x*p[i].y;
    }
    return area / 2.0;
}

double CAR(const vector<point> &p) {
    return fabs(ComputeSignedArea(p));
}
point ComputeCentroid(const vector<point> &p) {
  point c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
}

// centroid
point CCN(const vector<point> &p) {
    point c(0,0);
    double scale = 6.0 * ComputeSignedArea(p);
    for (int i = 0; i < p.size(); i++){
        int j = (i+1) % p.size();
        c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
    }
    return c / scale;
}
```

```
double myAngle(point a,point b)
{
    double ang = angle(a,b);
    if(dcmp(cross(a,b)) >= 0) return ang;
    return 2*PI - ang;
}
// tests whether or not a given polygon (in CW or cross order) is simple
bool IsSimple(const vector<point> &p) {
    for (int i = 0; i < p.size(); i++) {
        for (int k = i+1; k < p.size(); k++) {
            int j = (i+1) % p.size();
            int l = (k+1) % p.size();
            if (i == l || j == k) continue;
            if (SGIN(p[i], p[j], p[k], p[l]))
                return false;
        }
    }
    return true;
}

bool cw(const point &a, const point &b, const point &c) {
    return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x) < 0;
}
vector<point> convexHull(vector<point>& p) {
    int n = (int)p.size();
    if (n <= 1)
        return p;
    sort(p.begin(), p.end());
    int cnt = 0;
    vector<point> q(n * 2);
    for (int i = 0; i < n; q[cnt++] = p[i++])
        for (; cnt >= 2 && !cw(q[cnt - 2], q[cnt - 1], p[i]); --cnt)
            ;
    for (int i = n - 2, t = cnt; i >= 0; q[cnt++] = p[i--])
        for (; cnt > t && !cw(q[cnt - 2], q[cnt - 1], p[i]); --cnt)
            ;
    q.resize(cnt - 1 - (q[0] == q[1]));
    return q;
}
```

Data Structure

1.Mo’s Algorithm

```
#include <cstdio>
#include <algorithm>
```

```
using namespace std;

#define N 311111
#define A 1111111
#define BLOCK 555 // ~sqrt(N)

int cnt[A], a[N], ans[N], answer = 0;

struct node {
	int L, R, i;
}q[N];

bool cmp(node x, node y) {
	if(x.L/BLOCK != y.L/BLOCK) {
		// different blocks, so sort by block.
		return x.L/BLOCK < y.L/BLOCK;
	}
	// same block, so sort by R value
	return x.R < y.R;
}

void add(int position) {
	cnt[a[position]]++;
	if(cnt[a[position]] == 1) {
		answer++;
	}
}

void remove(int position) {
	cnt[a[position]]--;
	if(cnt[a[position]] == 0) {
		answer--;
	}
}

int main() {
	int n;
	scanf("%d", &n);
	for(int i=0; i<n; i++)
		scanf("%d", &a[i]);

	int m;
	scanf("%d", &m);
	for(int i=0; i<m; i++) {
		scanf("%d%d", &q[i].L, &q[i].R);
		q[i].L--; q[i].R--;
		q[i].i = i;
	}

	sort(q, q + m, cmp);
```

```
        int currentL = 0, currentR = 0;
        for(int i=0; i<m; i++) {
                int L = q[i].L, R = q[i].R;
                while(currentL < L) {
                        remove(currentL);
                        currentL++;
                }
                while(currentL > L) {
                        add(currentL-1);
                        currentL--;
                }
                while(currentR <= R) {
                        add(currentR);
                        currentR++;
                }
                while(currentR > R+1) {
                        remove(currentR-1);
                        currentR--;
                }
                ans[q[i].i] = answer;
        }

        for(int i=0; i<m; i++)
                printf("%d\n", ans[i]);
}
```

2. Disjoint Set union

```
#define mx 1500005
int Arr[mx];
int size[mx];
void initialize(  int N)
{
    for(int i = 0; i<N; i++)
    {
        Arr[ i ] = i ;
        size[ i ] = 1;
    }
}
int root (int i)
{
    while(Arr[ i ] != i)
    {
        Arr[ i ] = Arr[ Arr[ i ] ] ;
        i = Arr[ i ];
    }
```

```
    return i;
}

bool find(int A,int B)
{
    if( root(A)==root(B) )       //if A and B have same root,means they are connected.
        return true;
    else
        return false;
}
void weighted_union(int size[ ],int A,int B)
{
    int root_A = root(A);
    int root_B = root(B);
    if(size[root_A] < size[root_B ])
    {
        Arr[ root_A ] = Arr[root_B];
        size[root_B] += size[root_A];
    }
    else
    {
        Arr[ root_B ] = Arr[root_A];
        size[root_A] += size[root_B];
    }
}
```

3. BIT

```
int tree[SIZE];
int n;

void update(int x,int val)
{
    if(x == 0)
    {
        return ;
    }

    while(x <= n)
    {
        tree[x] +=  val;
        x += x & -x;
    }

}

int query(int i)
{
```

```
    int  ans;
    ans = 0;

    while(i > 0)
    {
        ans = ans + tree[i];
        i = i - (i & (-i));
    }

    return ans;
}
```

4. LCA

```
///http://lightoj.com/volume_showproblem.php?problem=1101
/// query on a graph with the help of LCA
///Problem in short : Given a dense graph. In each query type (a,b) .  Answer the max weight
edge //in the path from a to b.
///solution:
#include<bits/stdc++.h>

using namespace std;

#define ll long long int
#define db(x) cout<<#x<<" -> "<<x<<endl
#define mp make_pair
#define maxn 50005
#define xx first
#define yy second

vector < pair< int , pair< int , int > > > v;
int n,m;
int parent[maxn];
vector< pair<int, int > > g[maxn];
int dis[maxn],lvl[maxn],par[maxn];
bool vis[maxn];
int sparse_parent[maxn][20],sparse_dist[maxn][20];

int findset(int x){
    if(x!=parent[x]){
        parent[x] = findset(parent[x]);
    }
    return parent[x];
}

void kruskal(){
    for(int i=0; i<=n; i++){
        parent[i] = i;
```

```
    }
    sort(v.begin(),v.end());
    for(int i=0; i<v.size(); i++){
        int pu = findset(v[i].second.first);
        int pv = findset(v[i].second.second);
        if(pu!=pv){
            g[v[i].yy.xx].push_back(mp(v[i].yy.yy,v[i].xx));
            g[v[i].yy.yy].push_back(mp(v[i].yy.xx,v[i].xx));
            parent[pu] = pv;
        }
    }
}

void dfs(int src, int parent , int dep){
    vis[src] = true;
    par[src] = parent;
    lvl[src] = dep;
    for(int i=0; i<g[src].size(); i++){
        int x = g[src][i].xx;
        int y = g[src][i].yy;
        if( vis[x] ) continue;
        dis[x] = min(dis[src]+y,dis[x]);
        dfs(x,src,dep+1);
    }
}

void lca_init(){
    memset(sparse_parent,-1,sizeof(sparse_parent));
//  memset(sparse_dist,0,sizeof(sparse_dist));
    for(int i=0; i<n; i++ ){  ///sparse_parent[i][j] : node i er 2^j th parent;
sparse_dist[i][j] : node i e 2^j th parent porjonto max weight
        sparse_parent[i][0] = par[i];
        if(par[i]==-1) { sparse_dist[i][0] = dis[i] ; continue; }
        sparse_dist[i][0] = (dis[i] - dis[par[i]]);
//      printf("sparse_dist[%d][%d]: %d\n",i,0,sparse_dist[i][0]);
    }
    for(int j=1; (1<<j)<n; j++){
        for(int i=0; i<n; i++){
            if(sparse_parent[i][j-1]==-1) continue;
            sparse_parent[i][j] = sparse_parent[sparse_parent[i][j-1]][j-1] ;
            sparse_dist[i][j] =
max(sparse_dist[i][j-1],sparse_dist[sparse_parent[i][j-1]][j-1]);
//      printf("sparse_dist[%d][%d]: %d\n",i,j,sparse_dist[i][j]);
        }
    }
}

int lca_query(int p, int q){
    if(lvl[p]<lvl[q]) swap(p,q);
    int log = 1;
```

```
    while(true){
        int next = log+1;
        if((1<<next)>lvl[p]) break;
        log++;
    }
    int ret = -1000000;
    for(int i=log; i>=0; i--){  ///same level e niye ashtechi
        if((lvl[p]-(1<<i))>=lvl[q]){
            ret = max(ret,sparse_dist[p][i]);
            p = sparse_parent[p][i];
        }
    }
    if(p==q) return ret;
    for(int i=log; i>=0; i--){
        if(sparse_parent[p][i]!=-1 && sparse_parent[p][i]!=sparse_parent[q][i]){
            ret = max(ret,max(sparse_dist[p][i],sparse_dist[q][i]));
            p = sparse_parent[p][i];
            q = sparse_parent[q][i];
        }
    }
    int x ,y;
    if(par[p]==-1) x = dis[p];
    else x = dis[p]-dis[par[p]];
    if(par[q]==-1) y = dis[q];
    else y = dis[q]-dis[par[q]];
    ret = max(ret,max(x,y));
    return ret;
}



int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    int t;
//    cout<<log2(maxn)<<endl;
    cin>>t;
    int tc  = 0 ;
    while(t--){
        cin>>n>>m;
        for(int i=1; i<=m; i++){
            int a,b,c;
            cin>>a>>b>>c;
            a--;
            b--;
            v.push_back(mp(c,mp(a,b)));
        }
        kruskal();
        memset(dis,1000000,sizeof(dis));
        memset(vis,false,sizeof(vis));
```

```
        dis[0] = 0;
        dfs(0,-1,0);
//        for(int i=0; i<n; i++){
//          cout<<i<<" : "<<dis[i]<<endl;
//        }
        lca_init();
        printf("Case %d:\n",++tc);
        int q;
        cin>>q;
        while(q--){
            int a,b;
            cin>>a>>b;
            a--;
            b--;
            printf("%d\n",lca_query(a,b));
        }
        for(int i=0; i<=n; i++){
            g[i].clear();
        }
        v.clear();
    }
    return 0;
}

/**
2
4 5
1 2 10
1 3 20
1 4 100
2 4 30
3 4 10
2
1 4
4 1
2 1
1 2 100
1
1 2

*/



///Main LCA code Part


#define maxn 300111
#define logN 20
```

```
vector <int> adj[maxn];
int f[maxn][logN], depth[maxn], n;
void dfs(int u) {
    for (int i = 1; i < logN; i++)
        f[u][i] = f[f[u][i - 1]][i - 1];

    for (int i = 0; i < (int) adj[u].size(); i++) {
        int v = adj[u][i];

        if (!depth[v]) {
            f[v][0] = u;
            depth[v] = depth[u] + 1;
            dfs(v);
        }
    }
}
int lca (int u, int v) {
    if (depth[u] < depth[v]) swap(u, v);

    for (int i = logN - 1; i >= 0; i--)
        if (depth[f[u][i]] >= depth[v]) {
            u = f[u][i];
        }

    if (u == v) return u;

    for (int i = logN - 1; i >= 0; i--)
        if (f[u][i] != f[v][i]) {
            u = f[u][i];
            v = f[v][i];
        }

    return f[u][0];
}

int dist (int u, int v) {
    int x = lca(u, v);
    int res = depth[u] + depth[v] - 2 * depth[x];
    return res;
}
```

5.Segment tree with lazy

```
long long tree[4*100005],lazy[400005];

void updateRange(int node, int start, int end, int l, int r,long long int val)
{
    if(lazy[node] != -1)
```

```
    {

        tree[node] = (end - start + 1) * lazy[node];
        if(start != end)
        {
            lazy[node*2] = lazy[node];
            lazy[node*2+1] = lazy[node];
        }
        lazy[node] = -1;
    }
    if(start > end or start > r or end < l)
        return;
    if(start >= l and end <= r)
    {
        tree[node] = (end - start + 1) * val;
        if(start != end)
        {
            lazy[node*2] = val;
            lazy[node*2+1] = val;
        }
        return;
    }
    int mid = (start + end) / 2;
    updateRange(node*2, start, mid, l, r, val);
    updateRange(node*2 + 1, mid + 1, end, l, r, val);
    tree[node] = tree[node*2] + tree[node*2+1];
}

long long  queryRange(int node, int start, int end, int l, int r)
{
    if(start > end or start > r or end < l)
        return 0;
    if(lazy[node] != -1)
    {

        tree[node] = (end - start + 1) * lazy[node];
        if(start != end)
        {
            lazy[node*2] = lazy[node];
            lazy[node*2+1] = lazy[node];
        }
        lazy[node] = -1;
    }
    if(start >= l and end <= r)
        return tree[node];
    int mid = (start + end) / 2;
    long long  int p1 = queryRange(node*2, start, mid, l, r);
    long long  int p2 = queryRange(node*2 + 1, mid + 1, end, l, r);
    return (p1 + p2);
}
```

```
void update(int node, int l, int r,int i,int x,int val)
{
    if(l>x or r<x)return;
    if(l==x and r==x)
    {
        tree[i][node]=val;
        return;
    }
    else
    {
        int mid = (l +r) / 2;

        update(2*node, l, mid,i,x,val);

        update(2*node+1, mid+1, r,i,x,val);

        tree[i][node] = tree[i][2*node]+ tree[i][2*node+1];
    }
}
int query(int node, int start, int end, int l, int r,int i)
{
    if(r < start or end < l)
    {

        return INT_MIN;
    }
    if(l <= start and end <= r)
    {
        return tree[i][node];
    }
    int mid = (start + end) / 2;
    int p1 = query(2*node, start, mid, l, r,i);
    int p2 = query(2*node+1, mid+1, end, l, r,i);
    return p1+p2;
}
```

6.Trie

```
struct trienode
{
    int isleaf;
    struct trienode *child[26]];
    trienode()
    {
        for(int i=0; i<26; i++)
            this->child[i]=NULL;
        this->isleaf=0;
```

```
    }

};
bool Insert(struct trienode *root, string key)
{
    int level;
    int length = key.size();
    int index;

    struct trienode *temp = root;
    int ok=0;
    for (level = 0; level < length; level++)
    {
        index = key[level]-'A';
        if (!temp->child[index])
            temp->child[index] = new trienode();
        if(temp->isleaf)
        {
            ok=1;
            break;
        }
        temp = temp->child[index];
    }
    temp->isleaf = true;
    return ok;
}
//You have to delete and re-assign root for every test case
void Delete(struct trienode *cur)
{
    for(int i=0; i<26; i++)
    {
        if(cur->child[i])
            Delete(cur->child[i]);
    }
    delete (cur);
}
```

7.Aho-Corasick Algorithm

```
////////////////////////////////////////////////////////////////////////////////////
// Aho-Corasick's algorithm, as explained in  http://dx.doi.org/10.1145/360825.360855  //
////////////////////////////////////////////////////////////////////////////////////

const int MAXS = 6 * 50 + 10; // Max number of states in the matching machine.
                              // Should be equal to the sum of the length of all keywords.
```

```
const int MAXC = 26; // Number of characters in the alphabet.

int out[MAXS]; // Output for each state, as a bitwise mask.
               // Bit i in this mask is on if the keyword with index i appears when the
               // machine enters this state.

// Used internally in the algorithm.
int f[MAXS]; // Failure function
int g[MAXS][MAXC]; // Goto function, or -1 if fail.

// Builds the string matching machine.
//
// words - Vector of keywords. The index of each keyword is important:
//         "out[state] & (1 << i)" is > 0 if we just found word[i] in the text.
// lowestChar - The lowest char in the alphabet. Defaults to 'a'.
// highestChar - The highest char in the alphabet. Defaults to 'z'.
//               "highestChar - lowestChar" must be <= MAXC, otherwise we will
//               access the g matrix outside its bounds and things will go wrong.
//
// Returns the number of states that the new machine has.
// States are numbered 0 up to the return value - 1, inclusive.
int buildMatchingMachine(const vector<string> &words, char lowestChar = 'a', char
highestChar = 'z') {
    memset(out, 0, sizeof out);
    memset(f, -1, sizeof f);
    memset(g, -1, sizeof g);

    int states = 1; // Initially, we just have the 0 state

    for (int i = 0; i < words.size(); ++i) {
        const string &keyword = words[i];
        int currentState = 0;
        for (int j = 0; j < keyword.size(); ++j) {
            int c = keyword[j] - lowestChar;
            if (g[currentState][c] == -1) { // Allocate a new node
                g[currentState][c] = states++;
            }
            currentState = g[currentState][c];
        }
        out[currentState] |= (1 << i); // There's a match of keywords[i] at node
currentState.
    }

    // State 0 should have an outgoing edge for all characters.
    for (int c = 0; c < MAXC; ++c) {
        if (g[0][c] == -1) {
            g[0][c] = 0;
        }
    }
```

```
    // Now, let's build the failure function
    queue<int> q;
    for (int c = 0; c <= highestChar - lowestChar; ++c) {  // Iterate over every possible
input
        // All nodes s of depth 1 have f[s] = 0
        if (g[0][c] != -1 and g[0][c] != 0) {
            f[g[0][c]] = 0;
            q.push(g[0][c]);
        }
    }
    while (q.size()) {
        int state = q.front();
        q.pop();
        for (int c = 0; c <= highestChar - lowestChar; ++c) {
            if (g[state][c] != -1) {
                int failure = f[state];
                while (g[failure][c] == -1) {
                    failure = f[failure];
                }
                failure = g[failure][c];
                f[g[state][c]] = failure;
                out[g[state][c]] |= out[failure]; // Merge out values
                q.push(g[state][c]);
            }
        }
    }

    return states;
}

// Finds the next state the machine will transition to.
//
// currentState - The current state of the machine. Must be between
//                0 and the number of states - 1, inclusive.
// nextInput - The next character that enters into the machine. Should be between lowestChar
//             and highestChar, inclusive.
// lowestChar - Should be the same lowestChar that was passed to "buildMatchingMachine".

// Returns the next state the machine will transition to. This is an integer between
// 0 and the number of states - 1, inclusive.
int findNextState(int currentState, char nextInput, char lowestChar = 'a') {
    int answer = currentState;
    int c = nextInput - lowestChar;
    while (g[answer][c] == -1) answer = f[answer];
    return g[answer][c];
}


// How to use this algorithm:
//
```

```
// 1. Modify the MAXS and MAXC constants as appropriate.
// 2. Call buildMatchingMachine with the set of keywords to search for.
// 3. Start at state 0. Call findNextState to incrementally transition between states.
// 4. Check the out function to see if a keyword has been matched.
//
// Example:
//
// Assume keywords is a vector that contains {"he", "she", "hers", "his"} and text is a
string
// that contains "ahishers".
//
// Consider this program:
//
// buildMatchingMachine(v, 'a', 'z');
// int currentState = 0;
// for (int i = 0; i < text.size(); ++i) {
//    currentState = findNextState(currentState, text[i], 'a');
//    if (out[currentState] == 0) continue; // Nothing new, let's move on to the next
character.
//    for (int j = 0; j < keywords.size(); ++j) {
//        if (out[currentState] & (1 << j)) { // Matched keywords[j]
//            cout << "Keyword " << keywords[j] << " appears from "
//                 << i - keywords[j].size() + 1 << " to " << i << endl;
//        }
//    }
// }
//
// The output of this program is:
//
// Keyword his appears from 1 to 3
// Keyword he appears from 4 to 5
// Keyword she appears from 3 to 5
// Keyword hers appears from 4 to 7

////////////////////////////////////////////////////////////////////////////////
//                          End of Aho-Corasick's algorithm.                  //
////////////////////////////////////////////////////////////////////////////////


int main(){
    vector<string> keywords;
    keywords.push_back("he");
    keywords.push_back("she");
    keywords.push_back("hers");
    keywords.push_back("his");
    string text = "ahishers";

    buildMatchingMachine(keywords, 'a', 'z');
    int currentState = 0;
    for (int i = 0; i < text.size(); ++i) {
```

```
        currentState = findNextState(currentState, text[i], 'a');
        if (out[currentState] == 0) continue; // Nothing new, let's move on to the next
 character.
        for (int j = 0; j < keywords.size(); ++j) {
            if (out[currentState] & (1 << j)) { // Matched keywords[j]
                cout << "Keyword " << keywords[j] << " appears from "
                     << i - keywords[j].size() + 1 << " to " << i << endl;
            }
        }
    }

    return 0;

 }
```

8.KMP

```
 #include<bits/stdc++.h>
 using namespace std;
 void computeLPSArray(string pat, int M, int* lps);

 // Prints occurrences of txt[] in pat[]
 int KMPSearch(string pat, string txt)
 {  int co=0;
        int M = pat.size();
        int N = txt.size();

        // create lps[] that will hold the longest prefix suffix
        // values for pattern
        int lps[M];

        // Preprocess the pattern (calculate lps[] array)
        computeLPSArray(pat, M, lps);

        int i = 0; // index for txt[]
        int j = 0; // index for pat[]
        while (i < N) {
               if (pat[j] == txt[i]) {
                      j++;
                      i++;
               }

               if (j == M) {
                   co++;
               //     printf("Found pattern at index %d ", i - j);
                      j = lps[j - 1];
               }

               // mismatch after j matches
```

```
                else if (i < N && pat[j] != txt[i]) {
                        // Do not match lps[0..lps[j-1]] characters,
                        // they will match anyway
                        if (j != 0)
                                j = lps[j - 1];
                        else
                                i = i + 1;
                }
        }
        return co;
}

// Fills lps[] for given patttern pat[0..M-1]
void computeLPSArray(string pat, int M, int* lps)
{
        // length of the previous longest prefix suffix
        int len = 0;

        lps[0] = 0; // lps[0] is always 0

        // the loop calculates lps[i] for i = 1 to M-1
        int i = 1;
        while (i < M) {
                if (pat[i] == pat[len]) {
                        len++;
                        lps[i] = len;
                        i++;
                }
                else // (pat[i] != pat[len])
                {
                        // This is tricky. Consider the example.
                        // AAACAAAA and i = 7. The idea is similar
                        // to search step.
                        if (len != 0) {
                                len = lps[len - 1];

                                // Also, note that we do not increment
                                // i here
                        }
                        else // if (len == 0)
                        {
                                lps[i] = 0;
                                i++;
                        }
                }
        }
}

int main()
{
```

```
    int test;
    cin>>test;
    for(int cs=1; cs<=test; cs++)
    {

        string a,b;
        cin>>a>>b;
        cout<<"Case "<<cs<<": "<<KMPSearch(b,a)<<endl;
    }
}
```

String

1.String Hashing

```
long long compute_hash(string const& s) {
    const int p = 31;
    const int m = 1e9 + 9;
    long long hash_value = 0;
    long long p_pow = 1;
    for (char c : s) {
        hash_value = (hash_value + (c - 'a' + 1) * p_pow) % m;
        p_pow = (p_pow * p) % m;
    }
    return hash_value;
}
vector<vector<int>> group_identical_strings(vector<string> const& s) {
    int n = s.size();
    vector<pair<long long, int>> hashes(n);
    for (int i = 0; i < n; i++)
        hashes[i] = {compute_hash(s[i]), i};

    sort(hashes.begin(), hashes.end());

    vector<vector<int>> groups;
    for (int i = 0; i < n; i++) {
        if (i == 0 || hashes[i].first != hashes[i-1].first)
            groups.emplace_back();
        groups.back().push_back(hashes[i].second);
    }
    return groups;
}
int count_unique_substrings(string const& s) {
    int n = s.size();

    const int p = 31;
    const int m = 1e9 + 9;
```

```
    vector<long long> p_pow(n);
    p_pow[0] = 1;
    for (int i = 1; i < n; i++)
        p_pow[i] = (p_pow[i-1] * p) % m;

    vector<long long> h(n + 1, 0);
    for (int i = 0; i < n; i++)
        h[i+1] = (h[i] + (s[i] - 'a' + 1) * p_pow[i]) % m;

    int cnt = 0;
    for (int l = 1; l <= n; l++) {
        set<long long> hs;
        for (int i = 0; i <= n - l; i++) {
            long long cur_h = (h[i + l] + m - h[i]) % m;
            cur_h = (cur_h * p_pow[n-i-1]) % m;
            hs.insert(cur_h);
        }
        cnt += hs.size();
    }
    return cnt;
}
```

2.Rabin-Karp Algorithm

```
vector<int> rabin_karp(string const& s, string const& t) {
    const int p = 31;
    const int m = 1e9 + 9;
    int S = s.size(), T = t.size();

    vector<long long> p_pow(max(S, T));
    p_pow[0] = 1;
    for (int i = 1; i < (int)p_pow.size(); i++)
        p_pow[i] = (p_pow[i-1] * p) % m;

    vector<long long> h(T + 1, 0);
    for (int i = 0; i < T; i++)
        h[i+1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
    long long h_s = 0;
    for (int i = 0; i < S; i++)
        h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;

    vector<int> occurences;
    for (int i = 0; i + S - 1 < T; i++) {
        long long cur_h = (h[i+S] + m - h[i]) % m;
        if (cur_h == h_s * p_pow[i] % m)
            occurences.push_back(i);
    }
    return occurences;
}
```

3.Prefix function

```
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;
}
```

4. Z function

```
vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}
```

5. Finding all sub-palindromes in O(N)

```
vector<int> d1(n);
for (int i = 0, l = 0, r = -1; i < n; i++) {
    int k = (i > r) ? 1 : min(d1[l + r - i], r - i);
    while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
        k++;
    }
    d1[i] = k--;
    if (i + k > r) {
        l = i - k;
        r = i + k;
    }
}
vector<int> d2(n);
for (int i = 0, l = 0, r = -1; i < n; i++) {
```

```
    int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
    while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
        k++;
    }
    d2[i] = k--;
    if (i + k > r) {
        l = i - k - 1;
        r = i + k ;
    }
}
```

Game Theory

1.NIM

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int test;
    scanf("%d",&test);
    for(int cs=1;cs<=test;cs++){
        int n;
        scanf("%d",&n);
        int b[n];
        for(int i=0;i<n;i++) scanf("%d",&b[i]);


        int ans=b[0];
        for(int i=1;i<n;i++) ans^=b[i];
        printf("Case %d: ",cs);
        if(ans!=0){
            printf("Alice\n");
        }
        else printf("Bob\n");



    }
}
```

2.Misere Nim

```
#include<bits/stdc++.h>
```

```
using namespace std;
int main()
{
    int test;
    scanf("%d",&test);
    for(int cs=1; cs<=test; cs++)
    {
        int n,m;
        scanf("%d",&n);

        int now=0;
        int co=0;
        for(int j=0; j<n; j++)
        {
            int x;
            scanf("%d",&x);
            if(x==1)
                co++;
            now^=x;
        }
        if(co==n)
        {
            if(!now)
            {
                printf("Case %d: Alice\n",cs);
            }
            else
                printf("Case %d: Bob\n",cs);
            continue;
        }


        if(now)
        {
            printf("Case %d: Alice\n",cs);
        }
        else
            printf("Case %d: Bob\n",cs);




    }
}
```

3.Grundy number

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

int g[10005];
int exist[10005];
int getgrundy(int n)
{
    memset(exist,0,sizeof(exist));
    for(int i=1; i<=n/2; i++)
    {
        if(i==n-i)
            continue;
        exist[g[i]^g[n-i]]=1;
    }
    int grundy=0;
    while(exist[grundy])
        grundy++;
    return grundy;

}
int main()
{

    for(int i=3; i<=10000; i++)
        g[i]=getgrundy(i);
    int test;
    scanf("%d",&test);
    for(int cs=1; cs<=test; cs++)
    {
        int n;
        scanf("%d",&n);
        int ans=0;
        for(int i=0;i<n;i++){
            int x;
            scanf("%d",&x);
            ans^=g[x];
        }
        if(ans){
            printf("Case %d: Alice\n",cs);
        }
        else            printf("Case %d: Bob\n",cs);




    }
}
```

Extra

1.nCr%p

```
// Returns nCr % p.  In this Lucas Theorem based program,
// this function is only called for n < p and r < p.
int nCrModpDP(int n, int r, int p)
{
    // The array C is going to store last row of
    // pascal triangle at the end. And last entry
    // of last row is nCr
    int C[r+1];
    memset(C, 0, sizeof(C));

    C[0] = 1; // Top row of Pascal Triangle

    // One by constructs remaining rows of Pascal
    // Triangle from top to bottom
    for (int i = 1; i <= n; i++)
    {
        // Fill entries of current row using previous
        // row values
        for (int j = min(i, r); j > 0; j--)

            // nCj = (n-1)Cj + (n-1)C(j-1);
            C[j] = (C[j] + C[j-1])%p;
    }
    return C[r];
}

// Lucas Theorem based function that returns nCr % p
// This function works like decimal to binary conversion
// recursive function.  First we compute last digits of
// n and r in base p, then recur for remaining digits
int nCrModpLucas(int n, int r, int p)
{
   // Base case
   if (r==0)
      return 1;

   // Compute last digits of n and r in base p
   int ni = n%p, ri = r%p;

   // Compute result for last digits computed above, and
   // for remaining digits.  Multiply the two results and
   // compute the result of multiplication in modulo p.
```

```
    return (nCrModpLucas(n/p, r/p, p) * // Last digits of n and r
            nCrModpDP(ni, ri, p)) % p;  // Remaining digits
}
```

2.Matrix Explanation

```
typedef unsigned long long ll;
ll mod=1000000007;
class Matrix
{

public :
    ll a[2][2];


    Matrix ()
    {
        a[0][0]=0;
        a[0][1]=0;
        a[1][0]=0;
        a[1][1]=0;
    }
    Matrix operator*(Matrix x)
    {
        Matrix temp;

        for(int i = 0; i < 2; i ++)
        {
            for(int j = 0; j < 2; j ++)
            {
                for(int k = 0; k < 2; k ++)
                {

temp.a[i][j]=((temp.a[i][j]%mod)+(((a[i][k])%mod*(x.a[k][j])%mod)%mod)%mod)%mod;

                }
            }
        }
        return temp;
    }
    void show()
    {
        for(int i=0; i<2; i++)
        {
            for(int j=0; j<2; j++) cout<<a[i][j]<<" ";
            cout<<endl;
        }
    }
```

```
};
Matrix binpow(Matrix n,ll m)
{


    if(m==1) return n;
    Matrix  temp=binpow(n,m/2);
   // temp.show();
    temp=temp*temp;
    if(m%2==0) return temp;
    else return ((n)*(temp));
}
```