**Topic**

1. Finding Centroids
2. Is Prime
3. NcR
4. Topological Sorting
5. Finding Bridges (DFS)
6. Finding Articulation Points (DFS)
7. Dijkstra
8. Hashing
9. Circle in Directed Graph

**Finding Centroids:**

vector<ll> centroid;

vector<ll> v[100005];

ll sz[100005];

void dfs(ll u, ll p ,  ll n)

{

    sz[u]=1;

    bool is=true;

    for(auto it: v[u])

    {

        if(it!=p)

        {

            dfs(it,u,n);

            sz[u]+=sz[it];

            if(sz[it]>n/2) is=false;

        }

    }

    if(n-sz[u]>n/2) is=false;

    if(is) centroid.pb(u);

}

**Is Prime:**

bool isprime(ll n)

{

    if(n<=1)

        return false;

    if(n<=3)

        return true;

    if(n%2==0||n%3==0)

        return false;

    for(ll i=5; i\*i<=n; i=i+6)

        if(n%i==0||n%(i+2)==0)

            return false;

    return true;

}

**NcR:**

ll mod=998244353;

ll fact[300006];

ll fast\_mod\_exp(ll a, ll b, ll m) {

  ll res = 1;

  while (b > 0) {

    if (b & 1)

      res = (res \* a) % m;

    a = (a \* a) % m;

    b = b >> 1;

  }

  return res;

}

ll mod\_inverse(ll a, ll m) {

  // m is prime

  return fast\_mod\_exp(a, m - 2, m);

}

// ncr = ( n\*(n-1)\*(n-2)\*.....(n-r+1) ) \* mod\_inverse(fact[r],p);

ll ncr(ll n, ll r, ll p) {

  if (r == 0)

    return 1;

  // ll ans = 1;

  // for (ll i = 0; i <= r - 1; i++) {

  //  ans = (ans \* (n - i)) % p;

  // }

  return ((fact[n] \* mod\_inverse(fact[n - r], p)) % p \* mod\_inverse(fact[r], p)) % p;

}

void solve()

{

  ll n,k;

  cin>>n>>k;

    fact[0] = fact[1] = 1;

  for (ll i = 1; i <= 300002; i++) {

    fact[i] = (fact[i - 1] \* i ) % mod;

  }

//Your Code

}

**NCR 2:**

 const int32\_t M=1e9+7;

 const int32\_t MM=998244353;



 const int N=5000;



 #define NCR

 #define PRIME M

 int pw(int a,int p=M-2,int MOD=M){

 int result = 1;

 while (p > 0) {

 if (p & 1)

 result = a \* result % MOD;

 a = a \* a % MOD;

 p >>= 1;

 }

 return result;

 }

 int fact[N],invfact[N];

 void init(){

 int p=PRIME;

 fact[0]=1;

 int i;

 for(i=1;i<N;i++){

 fact[i]=i\*fact[i-1]%p;

 }

 i--;

 invfact[i]=pw(fact[i],p-2,p);

 for(i--;i>=0;i--){

 invfact[i]=invfact[i+1]\*(i+1)%p;

 }

 }

 int ncr(int n,int r){

 if(r > n || n < 0 || r < 0)return 0;

 return fact[n]\*invfact[r]%PRIME\*invfact[n-r]%PRIME;

 }

**Topological Sorting:**

vector <ll> res,v[1000];

 ll n,k,m;

ll in[100]={0};

void toposort()

{

  queue<ll> q;

  loop(i,1,n+1)

  {

    if(in[i]==0)

    q.push(i);

  }

  while(!q.empty())

  {

    ll cur=q.front();

    q.pop();

    res.pb(cur);

    for(auto node : v[cur])

    {

      in[node]--;

      if(in[node]==0)

      q.push(node);

    }

  }

}

**Finding Bridges (DFS):**

int n; // number of nodes

vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;

vector<int> tin, low;

int timer;

void dfs(int v, int p = -1) {

    visited[v] = true;

    tin[v] = low[v] = timer++;

    for (int to : adj[v]) {

        if (to == p) continue;

        if (visited[to]) {

            low[v] = min(low[v], tin[to]);

        } else {

            dfs(to, v);

            low[v] = min(low[v], low[to]);

            if (low[to] > tin[v])

                IS\_BRIDGE(v, to);

        }

    }

}

void find\_bridges() {

    timer = 0;

    visited.assign(n, false);

    tin.assign(n, -1);

    low.assign(n, -1);

    for (int i = 0; i < n; ++i) {

        if (!visited[i])

            dfs(i);

    }

}

# Finding Articulation Points (DFS):

int n; // number of nodes

vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;

vector<int> tin, low;

int timer;

void dfs(int v, int p = -1) {

    visited[v] = true;

    tin[v] = low[v] = timer++;

    int children=0;

    for (int to : adj[v]) {

        if (to == p) continue;

        if (visited[to]) {

            low[v] = min(low[v], tin[to]);

        } else {

            dfs(to, v);

            low[v] = min(low[v], low[to]);

            if (low[to] >= tin[v] && p!=-1)

                IS\_CUTPOINT(v);

            ++children;

        }

    }

    if(p == -1 && children > 1)

        IS\_CUTPOINT(v);

}

void find\_cutpoints() {

    timer = 0;

    visited.assign(n, false);

    tin.assign(n, -1);

    low.assign(n, -1);

    for (int i = 0; i < n; ++i) {

        if (!visited[i])

            dfs (i);

    }

}

**Dijkstra:**

ll dis[10005];

ll vis[10005]={0};

void dijkstra(vector<pr> \*v, ll st, ll d)

{

     priority\_queue<pr,vector<pr>,greater<pr>> q;

    q.push({d,st});

    dis[st]=d;

    while(!q.empty())

    {

        ll cur=q.top().ss;

        ll cur\_d=q.top().ff;

        q.pop();

        if(vis[cur]==1) continue;

        vis[cur]=1;

        for(auto edge : v[cur])

        {

            if(cur\_d+edge.ss <dis[edge.ff])

            {

                dis[edge.ff]=cur\_d+edge.ss;

                q.push({dis[edge.ff],edge.ff});

            }

        }

    }

}

**Hashing:**

#include<bits/stdc++.h>

using namespace std;

/\* Call init() before doing anything for god's sake.

Chance query return type to int or pair depending on number of hashes

Vector is really slow.

String -> abcde, base 10, a is 1

hash array will be-> [1,12,123,1234,12345]

hash value will be-> 12345

query (2,3)-> 34

update will only work for hash value. Array will not be modified and will be incosistent.

update (2,'e)-> new hash value is 12545

\*/

#define N 1

#define MAX 100000

long long base[N],mod[N],power[N][MAX+10];

int totalHash;

void init()

{

    totalHash=2;

    base[0]=3407;

    base[1]=4721;

    mod[0]=1000003999;

    mod[1]=1000000861;

    for(int i=0;i<totalHash;i++)

    {

        power[i][0]=1;

        for(int j=1;j<=MAX;j++)

        {

            power[i][j]=((power[i][j-1]\*base[i])%mod[i]);

        }

    }

}

struct HashData{

    long long ara[N][MAX+10],Hash[N];

    char str[MAX+10];

    int len;

    void init(char \*s)

    {

        strcpy(str,s);

        for(int i=0;i<totalHash;i++)

        {

            ara[i][0]=str[0];

            for(int j=1;j<len;j++)

            {

                ara[i][j]=(ara[i][j-1]\*base[i])%mod[i];

                ara[i][j]+=str[j];

                if(ara[i][j]>=mod[i])

                ara[i][j]-=mod[i];

            }

            Hash[i]=ara[i][len-1];

        }

    }

    inline pair<int,int> query (int st,int ed)

    {

        int ret[2];

        for(int i=0;i<totalHash;i++)

        {

            long long nw=ara[i][ed];

            if(st>0)

            {

                nw-=(ara[i][st-1]\*power[i][ed-st+1])%mod[i];

                if(nw<0)

                nw+=mod[i];

            }

            ret[i]=nw;

        }

        return {ret[0],ret[1]};

    }

    inline void append(char c)

    {

        len++;

        for(int i=0;i<totalHash;i++)

        {

            if(len>1)

            ara[i][len-1]=(ara[i][len-2]\*base[i])%mod[i];

            else

            ara[i][len-1]=0;

            ara[i][len-1]+=(c);

            if(ara[i][len-1]>=mod[i])

            ara[i][len-1]-=mod[i];

            Hash[i]=ara[i][len-1];

        }

        str[len-1]=c;

        str[len]=0;

    }

    inline bool isEqual (const HashData &b)

    {

        for(int i=0;i<totalHash;i++)

        {

            if(Hash[i]!=b.Hash[i])

            return false;

        }

        return true;

    }

    inline void update (int idx,char c)

    {

         for(int i=0;i<totalHash;i++)

        {

            Hash[i]-=(power[i][len-idx-1]\*str[idx])%mod[i];

            if(Hash[i]<0)

            Hash[i]+=mod[i];

            Hash[i]+=(power[i][len-idx-1]\*c)%mod[i];

            if(Hash[i]>=mod[i])

            Hash[i]-=mod[i];

        }

        str[idx]=c;

    }

};

**Circle in Directed Graph**

ll n,m,vis[N]={0},dis[N];

vector<ll> a[N],res;

stack<ll> rec;

bool is[N];

bool dfs(ll st)

{

  vis[st]=1;

  is[st]=true;

  rec.push(st);

  for(auto it: a[st])

  {

    if(vis[it]==0)

    {

      if(dfs(it))

      return true;

    }

    else

    {

      if(is[it])

      {

       // dbg1(it);

        rec.push(it);

        return true;

      }

    }

  }

  rec.pop();

  is[st]=false;

  return false;

}