Team notebook

IUT_ReverseHash

October 5, 2022

Contents

1 01 Templates

1.1 101 Nafis Template

```
/*
Check and remove this section while coding
1. Get rid of toolbars except compiler and
    main. Enable only logs and status.
2. Use C++17 in global compiler settings.
3. Turn on Wall, Wextra, Wshadow in warnings.
4. Make tab spout 4 spaces
5. Settings -> Compiler -> Linker Settings ->
    Other Linker Options:
    -Wl,--stack,268435456
```

```
6. Settings -> Environment -> General
    Settings -> Terminal to launch console
    programs -> select gnome
7. Do you have graph paper with you?
*/

/*
ID: nafis.f1
TASK:
LANG: C++
*/
#include<bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
typedef long long LL;
typedef pair<LL, LL> pLL;
#define ff first
#define ss second
#define show(x) cout << #x << ": " << x << ";
    "
#define INF 1000000000000015
#define MOD 1000000007
#define MAXN 2000006

void solve(int caseno)
{

}

int main()
{
        ios_base::sync_with_stdio(0);
        cin.tie(0);

        int T = 1, caseno = 0;

        cin >> T;

        while(T--)
        {
                solve(++caseno);
        }
}
```

2 02 Math

2.1 102 Derangements

```
// !n = (n-1) (!(n-1) + !(n-2))
LL D[MAXN];
LL getDerange(LL n)
{
    if(n == 1)
        return D[1] = 0;
    if(n == 2)
        return D[2] = 1;
    if(D[n] != 0)
        return D[n];

    return (n-1) * (getDerange(n-1) +
        getDerange(n-2));
}
```

3 03 Number Theory

3.1 101 Primality Test

```
// Easy BruteForce
bool isPrime(LL n)
{
    LL i;
    for(i = 2; i*i <= n; i++)
    {
        if(n%i == 0)
            return 0;
    }

    return 1;
}

// Fermat's Primality Test, vulnerable to
    Charmichael numbers
bool isPrime2(LL n)
{
    vector<LL> checkerPrimes = {2, 3, 5, 7};
    if(binary_search(checkerPrimes.begin(),
        checkerPrimes.end(), n) == 1)
        return 1;

    vector<LL> carmichael =
        {561,1105,1729,2465,2821,6601,8911,10585,1584
                                             29341,41041,46657
                                             115921,126217,162
                                             294409,314821,334
                                             488881,512461};

    if(binary_search(carmichael.begin(),
        carmichael.end(), n) == 1)
        return 0;

    for(auto cp : checkerPrimes)
    {
        if(bigmod(cp, n, n) != cp%n)
            return 0;
    }

    return 1;
}
```

3.2 102 Mulmod

```
LL mulmod(LL a, LL b, LL mod)
{
    if(b == 0)
        return 0;
    LL res = mulmod(a, b>>1, mod);
    res = (res<<1)%mod;
    if(b&1)
        return (res+a)%mod;
    else
        return res;
}

LL mulmod2(LL a, LL b, LL mod)
{
    LL ret = 0;

    while(b)
    {
        if(b&1)
            ret += a;

        a = a*2;
        b /= 2;
    }

    return ret;
}
```

3.3 103 Bigmod

```
LL bigmod(LL a, LL p, LL MOD)
{
    if(p == 0)
        return 1%MOD;
    if(p == 1)
        return a%MOD;

    LL res = bigmod(a, p/2, MOD);
    res = (res*res)%MOD;
    if(p&1)
        return (a*res)%MOD;
    return res;
}

LL bigmod2(LL a, LL p, LL MOD)
{
    LL res = 1%MOD;
    while(p)
    {
        if(p&1)
            res = (res*a)%MOD;
        a = (a*a)%MOD;
        p /= 2;
    }
    return res;
}
```

3.4 104 Invmod

```
LL inv[MAXN];

//only if mod is prime and gcd(a, mod) = 1
LL invmod(LL a, LL mod)
{
    return bigmod(a, mod-2, mod);
}

LL egcd(LL a, LL m, LL& x, LL& y)
{
    if(m == 0)
    {
        x = 1;
        y = 0;
        return a;
    }

    LL x1, y1;
    LL d = egcd(m, a%m, x1, y1);

    x = y1;
    y = x1 - y1*(a/m);

    return d;
}

//when gcd(a, mod) = 1
LL invod2(LL a, LL mod)
{
    LL x, y;
    egcd(a, mod, x, y);

    return (x%mod + mod) % mod;
}

//when N is prime
void allinvmod()
{
    LL i;
    inv[1] = 1;
    for(i = 2; i < N; i++)
        inv[i] = ((-N/i*inv[N%i]) % N + N) % N;
}
```

3.5 105 nCr

```
/*
We need actual value assuming answer fits in
    long long
   1. O(r): Multiply by (n-i) and divide by
       i, in each step. C(n, r) = C(n-1,
       r-1)*n/r

Prime mod M
   2. O(n): Precalculate factorial and
       inverse factorial array.
      O(1): Answer each query from these
          arrays
   3. O(M): Use Lucas Theorem

Non-prime mod M
   4. O(n*n): Use Pascal's Triangle
   5. O(M): Use Chinese Remainder Theorem
*/

LL fact[2000006];
LL inv[2000006];
LL dp[500][500];

LL getFact(LL n, LL mod);
LL bigmod(LL a, LL p, LL mod);
LL invmod(LL a, LL mod);

LL nCr1(LL n, LL r)
{
    if(n < r)
        return 0;

    r = min(r, n-r);

    if(r == 0)
        return 1;

    return n * nCr1(n-1, r-1) / r;
}

LL nCr2(LL n, LL r, LL mod)
{
    if(n < r)
```

```
        return 0;

    LL ret = getFact(n, mod);
    ret = (ret * invmod(getFact(r, mod), mod))
        % mod;
    ret = (ret * invmod(getFact(n-r, mod),
        mod)) % mod;
    return ret;
}

LL nCr3(LL n, LL r, LL mod)
{
    if(n < r)
        return 0;

    LL ret = 1;
    while(r)
    {
        ret = (ret * nCr2(n%mod, r%mod))%mod;
        n /= mod;
        r /= mod;
    }

    return ret;
}

LL nCr4(LL n, LL r, LL mod)
{
    if(n < r)
        return 0;

    if(dp[n][r] != 0)
        return dp[n][r];

    if(!r)
        return dp[n][r] = 1;

    return dp[n][r] = (nCr4(n-1, r-1, mod) +
        nCr4(n-1, r, mod)) % mod;
}

LL nCr5(LL n, LL r, LL mod)
{
    return -1;
}
```

3.6 106 Sieve of Eratosthenes

```
bool flag[MAXN];
vector<LL> primes;

void sieve()
{
    LL i, j;

    flag[2] = 1;
    for(i = 3; i < MAXN; i += 2)
        flag[i] = 1;

    for(i = 3; i*i < MAXN; i += 2)
    {
        if(flag[i])
        {
            for(j = i*i; j < MAXN; j += 2*i)
                flag[j] = 0;
        }
    }

    for(i = 2; i < MAXN; i++)
    {
        if(flag[i])
            primes.push_back(i);
    }
}
```

3.7 107 Linear Sieve

```
LL leastFactor[MAX];
bool isComposite[MAX];
vector<LL> primes;

void linSieve()
{
    LL i, j;
    for (i = 2; i < MAX; ++i)
    {
        if (leastFactor[i] == 0)
        {
            leastFactor[i] = i;
            primes.push_back(i);
        }
        for (j = 0; j < (LL)primes.size() &&
            primes[j] <= leastFactor[i] &&
            i*primes[j] < MAXN; ++j)
        {
            leastFactor[i * primes[j]] =
                primes[j];
        }
    }
}

void linSieve2()
{
    LL i, j;
    for(i = 2; i < N; i++)
    {
        if (!isComposite[i])
            primes.push_back(i);

        for(j = 0; j < (LL)primes.size() &&
            i*primes[j] < MAXN; j++)
        {
            isComposite[i * primes[j]] = 1;
            if(i % primes[j] == 0)
                break;
        }
    }
}
```

3.8 108 Number of Divisiors (sqrt)

```
bool flag[MAXN];
vector<LL> primes;

void sieve() {}

LL NOD(LL n)
{
    LL i, c, ret = 1;

    for(i = 0; primes[i]*primes[i] <= n; i++)
    {
        for(c = 0; n % primes[i] == 0; c++)
            n /= primes[i];
        ret *= (c+1);
    }

    if(n > 1)
        ret = ret << 1;

    return ret;
}
```

3.9 109 Sum of Divisors

```
bool flag[MAXN];
LL leastFactor[MAXN];
vector<LL> primes;
LL sod[MAXN];

void sieve() {}

LL linSOD(LL n)
{
    LL lf, c, p, ret = 1;

    while(n > 1)
    {
        lf = leastFactor[n];
        p = 1;

        for(c = 0; n%lf == 0; c++)
        {
            n /= lf;
            p *= lf;
        }

        ret *= (p*lf - 1)/(lf - 1);
    }

    return ret;
}

LL SOD(LL n)
{
    LL i, c, ret = 1;

    for(i = 0; primes[i]*primes[i] <= n; i++)
    {
        LL p = 1;
        for(c = 0; n % primes[i] == 0; c++)
        {
            n /= primes[i];
            p = p * primes[i];
        }
        ret *= (p * primes[i] - 1) /
             (primes[i] - 1);
    }

    if(n > 1)
        ret *= (n*n - 1) / (n - 1);
    return ret;
}

void allSOD()
{
    LL i, j;

    for(i = 1; i < MAXN; i++)
    {
        for(j = i; j < MAXN; j += i)
            sod[j] += i;
    }
}
```

3.10 110 Euler's Totient

```
bool flag[MAXN];
vector<int> primes;
ll phi[MAXN];

void sieve() {}

ll findPhi(ll n)
{
    if(phi[n] != 0)
        return phi[n];

    ll i, cnt, ret = n, temp = n;
    for(i = 0; primes[i] * primes[i] <= n; i++)
    {
        for(cnt = 0; n % primes[i] == 0; cnt++)
            n /= primes[i];

        if(cnt > 0)
            ret = ret / primes[i] * (primes[i]
                - 1);
    }

    if(n > 1)
        ret = ret / n * (n - 1);

    return phi[temp] = ret;
}

void sievephi()
{
    ll i, j;
    for(i = 1; i < MAXN; i++)
        phi[i] = i;

    for(i = 2; i < MAXN; i++)
    {
        if(phi[i] == i)
        {
            for(j = i; j < MAXN; j += i)
                phi[j] = phi[j] / i * (i - 1);
        }
    }
}

void segsievephi(ll a, ll b)
{
    ll i, j, cnt;

    for(i = a; i <= b; i++)
    {
        phi[i-a] = i;
        val[i-a] = i;
    }

    for(auto p : primes)
    {
        if(p * p > b)
            break;

        for(i = (a + p - 1) / p * p; i <= b; i
             += p)
        {
            for(cnt = 0; val[i - a] % p == 0;
                 cnt++)
                val[i - a] /= p;

            if(cnt)
                phi[i - a] = phi[i - a] / p *
                     (p - 1);
        }
    }

    for(i = a; i <= b; i++)
    {
        if(val[i - a] > 1)
            phi[i - a] = phi[i - a] / val[i -
                 a] * (val[i - a] - 1);
    }
}
```

3.11 111 Extended GCD

```
LL egcd(LL a, LL b, LL& x, LL& y)
{
    if (b == 0)
    {
        x = 1;
        y = 0;
        return a;
    }
    LL x1, y1;
    LL d = egcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

LL egcd2(LL a, LL b, LL& x, LL& y)
{
    x = 1, y = 0;
    LL x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1)
    {
        LL q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q *
             x1);
        tie(y, y1) = make_tuple(y1, y - q *
             y1);
        tie(a1, b1) = make_tuple(b1, a1 - q *
             b1);
    }
    return a1;
}

// Linear Diophantine Equation
// a*x + b*y = c
bool LDE(LL a, LL b, LL c, LL &x0, LL &y0, LL
     &d)
{
    d = egcd(abs(a), abs(b), x0, y0);
    if (c % d)
        return 0;

    x0 *= c / d;
    y0 *= c / d;
    x0 = (a < 0? -1 : 1) * x0;
    y0 = (b < 0? -1 : 1) * y0;
    return 1;
}

bool LDEall(LL a, LL b, LL c, LL t, LL &x, LL
     &y)
{
    LL d;
    if(LDE(a, b, c, x, y, d))
    {
        x = x + b*t;
        y = y - a*t;
        return 1;
    }

    return 0;
}
```

3.12 112 Matrix Exponentiation

```
typedef vector<vector<LL>> Mat;

Mat Mul(Mat A, Mat B)
{
    Mat ret(A.size(), vector<LL>(B[0].size()));
    LL i, j, k;

    for(i = 0; i < ret.size(); i++)
    {
        for(j = 0; j < ret[0].size(); j++)
        {
            for(k = 0; k < A[0].size(); k++)
                ret[i][j] = (ret[i][j] +
                     (A[i][k]*B[k][j])%MOD)%MOD;
        }
    }

    return ret;
}

Mat Pow(Mat A, LL p)
{
    Mat ret(A.size(), vector<LL>(A[0].size()));

    for(LL i = 0; i < ret.size(); i++)
        ret[i][i] = 1;

    while(p)
    {
        if(p&1)
            ret = Mul(ret, A);
        A = Mul(A, A);
        p >>= 1;
    }
    return ret;
}
```

3.13 113 Shanks' Baby Step, Giant Step

```
// Finds a^x = b (mod p)

LL bigmod(LL b, LL p, LL m) {}

LL babyStepGiantStep(LL a, LL b, LL p)
{
    LL i, j, c, sq = sqrt(p);
    map<LL, LL> babyTable;

    for(j = 0, c = 1; j <= sq; j++, c =
         (c*a)%p)
        babyTable[c] = j;

    LL giant = bigmod(a, sq*(p-2), p);

    for(i = 0, c = 1; i <= sq; i++, c =
         (c*giant)%p)
    {
        if(babyTable.find((c*b)%p) !=
             babyTable.end())
            return i*sq+babyTable[(c*b)%p];
    }

    return -1;
}
```

4 04 DP

5 05 Graph

5.1 101 Cycle Finding

```
LL n;
vector<LL> adj[MAXN];
LL color[MAXN], parent[MAXN];
LL cycle_start, cycle_end;

bool dfs(LL cur, LL p)
{
    parent[cur] = p;
    color[cur] = 1;
    for (LL nxt : adj[cur])
    {
        if (color[nxt] == 0)
        {
            if (dfs(nxt, cur))
                return true;
        }
        else if (color[nxt] == 1)
        {
            cycle_end = cur;
            cycle_start = nxt;
            return true;
        }
    }
    color[cur] = 2;
    return false;
}

void find_cycle()
{
    fill(color, color+n, 0);
    fill(parent, parent+n, -1);
    cycle_start = -1;

    LL i;

    for (i = 0; i < n; i++)
    {
        if (color[i] == 0 && dfs(i, -1))
            break;
    }

    if (cycle_start == -1)
        cout << "Acyclic" << "\n";
    else
    {
        vector<LL> cycle;
        cycle.push_back(cycle_start);
        for (LL v = cycle_end; v !=
             cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle found: ";
        for (LL v : cycle)
            cout << v << " ";
        cout << "\n";
    }
}
```

5.2 102 0-1 BFS

```
vector<pLL> adj[MAXN];
LL dist[MAXN];

void bfs01(LL cur)
{
    LL nxt, d;
    dequeu<pLL> dq;
    dq.push_back({0, cur});
    dist[cur] = 0;

    while(!dq.empty())
    {
        d = dq.front().ff;
        nxt = dq.front().ss;
        dq.pop_front();

        if(dist[nxt] < d)
            continue;

        for(auto [nxt, d2] : adj[cur])
        {
            if(dist[nxt] == -1 || d + d2 <
                 dist[nxt])
            {
                dist[nxt] = d + d2;

                if(d2)
                    dq.push_back({dist[nxt],
                         nxt});
                else
                    dq.push_front({dist[nxt],
                         nxt});
            }
        }
    }
}
```

5.3 103 Dijkstra

```
LL n;
vector <pLL> adj[MAXN];
LL dis[MAXN], par[MAXN];

void dijkstra(LL s)
{
    for(LL i = 1; i <= n; i++)
    {
        dis[i] = INF;
        par[i] = -1;
    }

    set<pLL> q;

    dis[s] = 0;
    q.insert({0, s});

    while(!q.empty())
    {
        pLL p = *q.begin();
        q.erase(q.begin());

        LL node = p.ss;
        if(p.ff > dis[node])
            continue;

        for (auto u : adj[node])
        {
            LL len = u.ff;
            LL to = u.ss;
            if (dis[node] + len < dis[to])
            {
                dis[to] = dis[node] + len;
                q.insert({dis[to], to});
                par[to] = node;
            }
        }
    }
}
```

5.4 104 Bellman Ford

```
struct edge
{
    int a, b, cost;
};

int n, m, v;
vector<edge> e;

void solve()
{
    vector<int> d (n, INF);
    vector<int> p (n, -1);
    d[v] = 0;
    bool any = 1;
    while(any)
    {
        any = 0;

        for (int j=0; j<m; j++)
        {
            if (d[e[j].a] < INF)
            {
                if (d[e[j].b] > d[e[j].a] +
                     e[j].cost)
                {
                    d[e[j].b] = d[e[j].a] +
                         e[j].cost;
                    p[e[j].b] = e[j].a;
                    any = 1;
                }
            }
        }
    }

    if (d[t] == INF)
        cout << "No path from " << v << " to "
             << t << ".";
    else
    {
        vector<int> path;
        for (int cur = t; cur != -1; cur =
             p[cur])
            path.push_back (cur);
        reverse (path.begin(), path.end());

        cout << "Path from " << v << " to " <<
             t << ": ";
        for (size_t i = 0; i < path.size();
             i++)
            cout << path[i] << ' ';
    }
}
```

5.5 105 Floyd warshall

```
LL n;
vector<pLL> adj[MAXN];
LL dis[MAXN][MAXN];

void floyd_warshall()
{
    LL i, j, k;

    memset(dist, -1, sizeof dist);
    for(i = 1; i <= n; i++)
    {
        dist[i][i] = 0;

        for(auto u : adj[i])
        {
            if(dist[i][u.ss] == -1)
                dist[i][u.ss] = dist[u.ss][i] =
                     c;
            else
                dist[i][u.ss] = dist[u.ss][i] =
                     min(dist[i][u.ss], u.ff);
        }
    }

    for(k = 1; k <= n; k++)
    {
        for(i = 1; i <= n; i++)
        {
            for(j = 1; j <= n; j++)
            {
                if(dist[i][k] != -1 &&
                    dist[k][j] != -1)
                {
                    if(dist[i][j] == -1)
                        dist[i][j] = dist[i][k]
                            + dist[k][j];
                    else
                        dist[i][j] =
                            min(dist[i][j],
                            dist[i][k] +
                            dist[k][j]);
                }
            }
        }
    }
}
```

5.6 106 Kruskal

```
LL leader[MAXN], n, m;
vector<pair<LL, pLL>>>edges;

LL Find(LL a)
{
    if(leader[a] == a)
        return a;
    leader[a] = Find(leader[a]);
    return leader[a];
}

void Union(LL a, LL b)
{
    a = Find(a);
    b = Find(b);

    leader[max(a, b)] = min(a, b);
}

LL kruskal()
{
    LL i, ret = 0;
    for(i = 1; i <= n; i++)
        leader[i] = i;

    sort(edges.begin(), edges.end());
    LL cnt = 0;
    for(auto [c, e] : edge)
    {
        [a, b] = e;

        if(Find(a) != Find(b))
        {
            Union(a, b);
            ret += c;
            cnt++;
            if(cnt == n-1)
                return ret;
        }
    }

    return -1;
}
```

5.7 107 Topsort with DFS

```
vector<int> adj[MAX];
stack<int> st;
int col[MAX];

bool dfs(int s)
{
    int ret = 1;
    col[s] = 1;
    for(auto u : adj[s])
    {
        if(col[u] == 0)
            ret = ret & dfs(u);
        else if(col[u] == 1)
            return 0;
    }
    col[s] = 2;
    st.push(s);
    return ret;
}

//Run it for all nodes
for(i = 1; i <= node; i++)
{
    if(col[i] == 0 && dfs(i) == 0)
    {
        cout << "impossible";
        return 0;
    }
}
```

5.8 108 TopSort with Indegree

```
int n;
vector<int> adj[MAXN];
vector<int> path;
int in[MAXN];

void topsort()
{
    queue<int> Q;
    int i;
    for(i = 1; i <= n; i++)
    {
        if(in[i] == 0)
            Q.push(i);
    }
    while(!Q.empty())
    {
        int node = Q.front();
        Q.pop();
        path.push_back(node);
        for(auto u : adj[node])
        {
            in[u]--;
            if(in[u] == 0)
                Q.push(u);
        }
    }
}

int main()
{
    int i, m;
    cin >> n >> m;
    for(i = 0; i < m; i++)
    {
        int u,v;
        cin >> u >> v;
        adj[u].push_back(v);
        in[v]++;
    }

    topsort();

    if(path.size() != n)
        cout << "impossible";
    else
    {
        for(i = 0; i < path.size(); i++)
            cout << path[i] << " ";
    }

}
```

5.9 109 Articulation Point

```
int n, m;
bool vis[MAXN];
int tin[MAXN], low[MAXN], timer;
vector<int> adj[MAXN];
set<int> AP;

void dfs(int v, int p = -1)
{
    vis[v] = 1;
    timer++;
    tin[v] = low[v] = timer;

    int child = 0;
    for(auto to : adj[v])
    {
        if(to == p)
            continue;
        if(!vis[to])
        {
            child++;
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if(low[to] >= tin[v] && p != -1)
                AP.insert(v);
        }
        else
            low[v] = min(low[v], tin[to]);
    }

    if(p == -1 && child > 1)
        AP.insert(v);
}

void findAP()
{
    AP.clear();
    timer = 0;
    int i;
    for(i = 1; i <= n; i++)
    {
        vis[i] = 0;
        tin[i] = -1;
        low[i] = -1;
    }

    for(i = 1; i <= n; i++)
    {
        if(!vis[i])
            dfs(i);
    }
}
```

5.10 110 Bridge

```
int n, m;
bool vis[MAXN];
int tin[MAXN], low[MAXN], timer;
vector<int> adj[MAXN];
vector<int> bridge[MAXN];

void dfs(int v, int p = -1)
{
    vis[v] = 1;
    timer++;
    tin[v] = low[v] = timer;

    int child = 0;
    for(auto to : adj[v])
    {
        if(to == p)
            continue;
        if(!vis[to])
        {
            child++;
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if(low[to] > tin[v])
            {
                bridge[v].push_back(to);
                bridge[to].push_back(v);
            }
        }
        else
            low[v] = min(low[v], tin[to]);
    }
}

void findBR()
{
    bridge.clear();
    timer = 0;
    int i;
    for(i = 1; i <= n; i++)
    {
        vis[i] = 0;
        tin[i] = -1;
        low[i] = -1;
    }

    for(i = 1; i <= n; i++)
    {
        if(!vis[i])
            dfs(i);
    }
}
```

5.11 111 Centroid Decomposition

```
int n;
vector <int> adj[MAXN];
int subtree_size[MAXN];

int get_subtree_size(int node, int par = -1)
{
        int ret = 1;

        for (auto next : adj[node])
    {
                if (next != par)
            ret += get_subtree_size(next, node);
        }
        return subtree_size[node] = ret;
}

int get_centroid(int node, int par = -1)
{
        for (auto next : adj[node])
    {
                if (next != par &&
                    subtree_size[next] * 2 > n)
                        return
                            get_centroid(next,
                            node);
        }
        return node;
}
```

5.12 112 Strongly Connected Components

```
vector <int> g[MAXN], gr[MAXN];
bool vis[MAXN];
vector<int> order, component;

void dfs1(int v)
{
    vis[v] = 1;
    for (auto u : g[v])
    {
        if (!vis[u])
            dfs1(u);
    }
    order.push_back (v);
}

void dfs2(int v)
{
    vis[v] = 1;
    component.push_back(v);
    for (auto u : gr[v])
    {
        if(!vis[u])
            dfs2(u);
    }
}

int main()
{
    int n, m, i, cnt = 0;
    cin >> n >> m;
    for (i = 0; i < m; i++)
    {
        int a, b;
        cin >> a >> b;
        g[a].push_back (b);
        gr[b].push_back (a);
    }

    memset(vis, 0, sizeof vis);
    for (i = 1; i <= n; i++)
    {
        if (!vis[i])
            dfs1(i);
    }

    memset(vis, 0, sizeof vis);
    for (i = 1; i <= n; i++)
    {
        int v = order[n-i];
        if (!vis[v])
        {
            dfs2 (v);
            cout << "Component No. " << ++cnt
                 << ": ";
            for(auto u : component)
                cout << u << " ";
            cout << endl;
            component.clear();
        }
    }
}
```

6 06 Tree

6.1 101 Least Common Ancestor

```
// Important Note: parent of 1 is 1
LL n;
vector<LL> adj[MAXN];
LL parent[MAXN], level[MAXN], anc[MAXN][21];

void dfs(LL cur, LL p, LL l)
{
    parent[cur] = p;
    level[cur] = l;

    for(auto nxt : adj[cur])
    {
        if(nxt != parent[cur])
            dfs(nxt, cur, l+1);
    }
}

void LCA_init()
{
    dfs(1, 1, 0);

    LL i, j;
    for(i = 1; i <= n; i++)
        anc[i][0] = parent[i];

    for(j = 1; j < 21; j++)
    {
        for(i = 1; i <= n; i++)
            anc[i][j] = anc[anc[i][j-1]][j-1];
    }
}
```

```
LL getLCA(LL u, LL v)
{
    if(level[u] < level[v])
        swap(u, v);

    LL i;
    for(i = 20; i >= 0; i--)
    {
        if(level[anc[u][i]] >= level[v])
            u = anc[u][i];
    }

    if(u == v)
        return u;

    for(i = 20; i >= 0; i--)
    {
        if(anc[u][i] != anc[v][i])
        {
            u = anc[u][i];
            v = anc[v][i];
        }
    }

    return parent[u];
}

// LCA_init()
// cout << getLCA(5, 8) << "\n";
```

6.2 102 Heavy Light Decomposition

```
int n, a[MAXN];
vector<int> adj[MAXN];
int parent[MAXN], level[MAXN],
    anc[MAXN][21];//for lca
int heavy[MAXN], subsize[MAXN];
int chainHead[MAXN], chainNo, basePos[MAXN],
    chainIdx[MAXN];//for hld
int base[MAXN], cnt, Tree[MAXN*4];//for
    segment Tree
```

```
int dfs(int node, int pr, int l)
{
    subsize[node] = 1;
    parent[node] = pr;
    level[node] = l;

    int mx = 0, x;

    for(auto u : adj[node])
    {
        if(u != parent[node])
        {
            x = dfs(u, node, l+1);
            subsize[node] += x;

            if(mx < x)
            {
                heavy[node] = u;
                mx = x;
            }
        }
    }

    return subsize[node];
}

void lca_init()
{
    int i, j;
    for(i = 1; i <= n; i++)
    {
        for(j = 0; j <= 20; j++)
            anc[i][j] = 1;
    }

    for(i = 1; i <= n; i++)
        anc[i][0] = parent[i];

    for(j = 1; (1<<j) <= n; j++)
    {
        for(i = 1; i <= n; i++)
            anc[i][j] = anc[anc[i][j-1]][j-1];
    }
}
```

```
int lca(int u, int v)
{
    if(level[u] < level[v])
        swap(u, v);

    int i;
    for(i = log2(n) + 1; i >= 0; i--)
    {
        if(level[anc[u][i]] >= level[v])
            u = anc[u][i];
    }

    if(u == v)
        return u;
    for(i = log2(n) + 1; i >= 0; i--)
    {
        if(anc[u][i] != anc[v][i])
        {
            u = anc[u][i];
            v = anc[v][i];
        }
    }
    return parent[u];
}

void hld_init(int u, int pr)
{
        if(chainHead[chainNo] == -1)
        chainHead[chainNo] = u;

        chainIdx[u] = chainNo;
        base[cnt++] = a[u];
        basePos[u] = cnt-1;

        if(heavy[u] > -1)
        hld_init(heavy[u], u);
        for(auto v : adj[u])
        {
                if(v != pr and v != heavy[u])
                {
                        chainNo++;
                        hld_init(v, u);
                }
        }
}
```

```
void build_tree(int node, int s, int e)
{
    if(s == e)
    {
        Tree[node] = base[s];
        return;
    }

    int mid = (s+e)/2, left = 2*node, right =
         2*node+1;

    build_tree(left, s, mid);
    build_tree(right, mid+1, e);

    Tree[node]= Tree[left] + Tree[right];
}

void update_tree(int node, int s, int e, int
     pos, int val)
{
    if(s > pos || e < pos)
        return;
    if(s == e)
    {
        base[s] = val;
        Tree[node] = val;
        return;
    }

    int mid = (s+e)/2, left = 2*node, right =
         2*node+1;

    update_tree(left, s, mid, pos, val);
    update_tree(right, mid+1, e, pos, val);

    Tree[node] = Tree[left] + Tree[right];
}

int query_tree(int node, int s, int e, int
     lo, int hi)
{
    if(hi < s || lo > e)
        return 0;
    if(lo <= s && hi >= e)
        return Tree[node];

    int mid = (s+e)/2, left = 2*node, right =
         2*node+1;
    int p1 = query_tree(left, s, mid, lo, hi);
    int p2 = query_tree(right, mid+1, e, lo,
         hi);

    return p1+p2;
}

int query_up(int u, int p)
{
        int uchain, pchain = chainIdx[p], ret
             = 0;

        while(1)
    {
                uchain = chainIdx[u];

                if(uchain == pchain)
                {
                        ret += query_tree(1, 1,
                             n, basePos[p],
                             basePos[u]);
                        break;
                }
                ret += query_tree(1, 1, n,
                     basePos[chainHead[uchain]],
                     basePos[u]);
                u = chainHead[uchain];
                u = parent[u];
        }
        return ret;
}

void update_hld(int p, int val)
{
    update_tree(1, 1, n, basePos[p], val);
}

int query_hld(int u, int v)
{
    int l = lca(u, v);
    return query_up(u, l) + query_up(v, l) -
         query_up(l, l);
}

void init()
{
    int i;
    for(i = 0; i <= n; i++)
    {
        heavy[i] = -1;
        chainHead[i] = -1;
    }
    dfs(1, 1, 0);
    lca_init();
    cnt = 1, chainNo = 1;
    hld_init(1, 1);
    build_tree(1, 1, n);
}

int main()
{
    int t, i, q, caseno = 0, u, v;
    scanf("%d", &t);

    while(t--)
    {
        scanf("%d", &n);
        for(i = 1; i <= n; i++)
        {
            scanf("%d", &a[i]);
            adj[i].clear();
        }

        for(i = 1; i < n; i++)
        {
            scanf("%d %d", &u, &v);
            adj[u+1].push_back(v+1);
            adj[v+1].push_back(u+1);
        }

        init();

        printf("Case %d:\n", ++caseno);
        scanf("%d", &q);
        while(q--)
        {
            int type, x, y;
            scanf("%d %d %d", &type, &x, &y);
            if(type)
                update_hld(x+1, y);
            else
                printf("%d\n", query_hld(x+1,
                     y+1));
        }
    }
}
```

6.3 103 Euler Tour on Tree

```
ll idx;
ll tour[MAXN];
vector<ll> adj[MAXN];

void createTourTree(ll node, ll p = -1)
{
    for(auto u : adj[node])
    {
        if(u != p)
            dfs(u, node);
    }

    tour[idx] = node;
    idx++;
}

idx = 0; // must before calling this function
```

7 07 Data Structure

7.1 101 Bitset

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC
     target("avx2,bmi,bmi2,lzcnt,popcnt")

bitset<MAXN> Add(bitset<MAXN> a, bitset<MAXN>
     b)
{
    bitset<MAXN> carry;
    while(b != bitset<MAXN>(0))
    {
        carry = a&b;
        a = a^b;
        b = carry << 1;
    }

    return a;
}

bitset<MAXN> Sub(bitset<MAXN> a, bitset<MAXN>
     b)
{
    b = ~b;
    b = Add(b, bitset<MAXN>(1));

    return Add(a, b);
}
```

7.2 102 Disjoint Set Union

```
LL leader[MAXN];

LL Find(LL x)
{
    if (x == leader[x])
        return x;
    return leader[x] = find_set(leader[x]);
}

void Union(LL x, LL y) {
    x = Find(x);
    y = Find(y);

    leader[max(x, y)] = min(x, y);
}
```

7.3 103 Ordered Set

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template<typename T>
using ordered_set = tree<T,
     null_type,less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;


ordered_set<LL> OS, OS2;

OS.insert(10);
OS.insert(20);
OS.insert(30);

//Find 2nd smallest element O(log(n))
cout << *(OS.find_by_order(1)) << "\n";

//Counting elements strictly less than 15
     O(log(n))
cout << OS.order_of_key(15) << "\n";

//Check existence of 30
cout << (OS.find(30) == OS.end()) << "\n";

//Delete 30
OS.erase(30);

//Swap two policy based data structure in O(1)
OS.swap(OS2);
```

7.4 104 Sparse Table RMQ

```
LL arr[MAXN], sparse[MAXN][20], Log[MAXN];

LL query(LL bg, LL ed)
{
    LL k = Log[ed-bg+1];
    return min(sparse[bg][k], sparse[ed - (1
         << k) + 1][k]);
}
```

```
void rmq_init()
{
    LL n, i, j;

    Log[1] = 0;
    for(i = 2; i < MAXN; i++)
        Log[i] = Log[i/2] + 1;

    cin >> n;
    for(i = 0; i <= n; i++)
        sparse[i][0] = arr[i];

    for(j = 1; j < 20; j++)
    {
        for(i = 0; i + (1LL << j) - 1 < n; i++)
            sparse[i][j] = min(sparse[i][j-1],
                sparse[i + (1LL <<
                (j-1))][j-1]);
    }
}
```

7.5 105 Segment tree

```
LL arr[MAXN];
LL Tree[4*MAXN];

void Build(LL node, LL bg, LL ed)
{
    if(bg == ed)
    {
        Tree[node] = arr[bg];
        return;
    }

    LL leftNode = 2*node, rightNode = 2*node +
        1;
    LL mid = (bg + ed)/2;

    Build(leftNode, bg, mid);
    Build(rightNode, mid+1, ed);

    Tree[node] = Tree[leftNode] +
        Tree[rightNode];
}

void Update(LL node, LL bg, LL ed, LL idx, LL
    val)
{
    if(bg == ed)
    {
        Tree[node] = val;
        arr[idx] = val;
        return;
    }

    LL leftNode = 2*node, rightNode = 2*node +
        1;
    LL mid = (bg + ed)/2;

    if(idx <= mid)
        Update(leftNode, bg, mid, idx, val);
    else
        Update(rightNode, mid+1, ed, idx, val);

    Tree[node] = Tree[leftNode] +
        Tree[rightNode];
}

LL Query(LL node, LL bg, LL ed, LL l, LL r)
{
    if(bg > r || ed < l)
        return 0;

    if(l <= bg && ed <= r)
        return Tree[node];

    LL leftNode = 2*node, rightNode = 2*node +
        1;
    LL mid = (bg + ed)/2;

    LL p1 = Query(leftNode, bg, mid, l, r);
    LL p2 = Query(rightNode, mid+1, ed, l, r);

    return p1 + p2;
}
```

7.6 106 Segment tree (Lazy)

```
// Node e dhukar sathe sathe lazy clear kora
    lagbe.
// Shei node amar desired range er vitore
    thakuk, othoba na thakuk.
// Naile WA khabi sure.
LL arr[MAXN];
LL Tree[4*MAXN], Lazy[4*MAXN];

void Build(LL node, LL bg, LL ed)
{
    if(bg == ed)
    {
        Lazy[node] = 0;
        Tree[node] = arr[bg];
        return;
    }

    LL leftNode = 2*node, rightNode = 2*node +
        1;
    LL mid = (bg + ed)/2;

    Build(leftNode, bg, mid);
    Build(rightNode, mid+1, ed);

    Tree[node] = Tree[leftNode] +
        Tree[rightNode];
    Lazy[node] = 0;
}

void updateRange(LL node, LL bg, LL ed, LL l,
    LL r, LL val)
{
    LL leftNode = 2*node, rightNode = 2*node +
        1;
    LL mid = (bg + ed)/2;

    if(Lazy[node] != 0)
    {
        Tree[node] += (ed - bg + 1) *
            Lazy[node];
        if(bg != ed)
        {
            Lazy[leftNode] += Lazy[node];
            Lazy[rightNode] += Lazy[node];
        }
        Lazy[node] = 0;
    }

    if(bg > r || ed < l)
        return;

    if(l <= bg && ed <= r)
    {
        Tree[node] += (ed - bg + 1) * val;
        if(bg != ed)
        {
            Lazy[leftNode] += val;
            Lazy[rightNode] += val;
        }
        return;
    }

    updateRange(leftNode, bg, mid, l, r, val);
    updateRange(rightNode, mid+1, ed, l, r,
        val);

    Tree[node] = Tree[leftNode] +
        Tree[rightNode];
}

LL queryRange(LL node, LL bg, LL ed, LL l, LL
    r)
{
    LL leftNode = 2*node, rightNode = 2*node +
        1;
    LL mid = (bg + ed)/2;

    if(Lazy[node] != 0)
    {
        Tree[node] += (ed - bg + 1) *
            Lazy[node];
        if(bg != ed)
        {
            Lazy[leftNode] += Lazy[node];
            Lazy[rightNode] += Lazy[node];
        }
        Lazy[node] = 0;
    }

    if(bg > r || ed < l)
        return 0;

    if(l <= bg && ed <= r)
        return Tree[node];

    LL p1 = queryRange(leftNode, bg, mid, l,
        r);
    LL p2 = queryRange(rightNode, mid + 1, ed,
        l, r);

    return (p1 + p2);
}
```

7.7 107 Trie

```
// Sometimes you need to use Add(0) at first

LL trie[MAXN][2], len[MAXN];
LL id;

void Add(LL x)
{
    LL r = 0;

    for(LL i = 34; i >= 0; i--)
    {
        LL bit = ((x & (1LL << i)) >> i);

        if(trie[r][bit] == 0)
            trie[r][bit] = ++id;

        r = trie[r][bit];
        len[r]++;
    }
}

void Erase(LL x)
{
    LL r = 0;

    for(LL i = 34; i >= 0; i--)
    {
        LL bit = ((x & (1LL << i)) >> i);

        r = trie[r][bit];
        len[r]--;
    }
}
```

7.8 108 Merge Sort Tree

```
vector<LL>Tree[4*MAXN];
LL arr[MAXN];

vector<LL> merge(vector<LL> v1, vector<LL> v2)
{
    LL i = 0, j = 0;
    vector<LL> ret;

    while(i < v1.size() || j < v2.size())
    {
        if(i == v1.size())
        {
            ret.push_back(v2[j]);
            j++;
        }
        else if(j == v2.size())
        {
            ret.push_back(v1[i]);
            i++;
        }
        else
        {
            if(v1[i] < v2[j])
            {
                ret.push_back(v1[i]);
                i++;
            }
            else
            {
                ret.push_back(v2[j]);
                j++;
            }
        }
    }

    return ret;
}

void Build(LL node, LL bg, LL ed)
{
    if(bg == ed)
    {
        Tree[node].push_back(arr[bg]);
        return;
    }

    LL leftNode = 2*node, rightNode = 2*node +
         1;
    LL mid = (bg+ed)/2;

    Build(leftNode, bg, mid);
    Build(rightNode, mid+1, ed);

    Tree[node] = merge(Tree[leftNode],
         Tree[rightNode]);
}

LL query(LL node, LL bg, LL ed, LL l, LL r,
     LL k)
{
    if(ed < l || bg > r)
        return 0;

    if(l <= bg && ed <= r)
        return upper_bound(Tree[node].begin(),
             Tree[node].end(), k) -
             Tree[node].begin();

    LL leftNode = 2*node, rightNode = 2*node +
         1;
    LL mid = (bg + ed)/2;

    return query(leftNode, bg, mid, l, r, k) +
         query(rightNode, mid+1, ed, l, r, k);
}
```

7.9 109 Square Root Decomposition

```
LL arr[MAXN], blocks[MAXN];
void Init()
{
    LL i, len = sqrt(n);
    for(i = 0; i < n; i++)
        blocks[i/len] += a[i];
}

LL Query(LL l, LL r)
{
    LL ret = 0;
    for(i = l; i <= r;)
    {
        if (i % len == 0 && i + len - 1 <= r)
        {
            sum += b[i / len];
            i += len;
        }
        else
        {
            sum += a[i];
            i++;
        }
    }

    return ret;
}

// ALTERNATE
int sq;
int arr[30004];
pair<pii, int> query[200005];
int freq[1000006], ans[200005];

bool cmp(pair<pii, int> a, pair<pii, int> b)
{
    if(a.ff.ff/sq != b.ff.ff/sq)
        return a.ff.ff < b.ff.ff;

    return a.ff.ss < b.ff.ss;
}

void solve(int caseno)
{
    int n, i, q, l, r, distinct;

    cin >> n;
    sq = sqrt(n);

    for(i = 1; i <= n; i++)
        cin >> arr[i];

    cin >> q;
    for(i = 0; i < q; i++)
    {
        cin >> query[i].ff.ff >>
             query[i].ff.ss;
        query[i].ss = i;
    }

    sort(query, query + q, cmp);

    distinct = 0;
    for(i = 0, l = 1, r = 0; i < q; i++)
    {
        while(r < query[i].ff.ss)
        {
            r++;
            if(freq[arr[r]] == 0)
                distinct ++;
            freq[arr[r]]++;
        }
        while(l > query[i].ff.ff)
        {
            l--;
            if(freq[arr[l]] == 0)
                distinct ++;
            freq[arr[l]]++;
        }

        while(l < query[i].ff.ff)
        {
            if(freq[arr[l]] == 1)
                distinct --;
            freq[arr[l]] --;

            l++;
        }
        while(r > query[i].ff.ss)
        {
            if(freq[arr[r]] == 1)
                distinct --;
            freq[arr[r]] --;

            r--;
        }

        ans[query[i].ss] = distinct;
    }

    for(i = 0; i < q; i++)
        cout << ans[i] << "\n";
}
```

7.10 110 Binary Indexed Tree

```
// Always 1 indexed
LL n;
LL a[MAXN], BIT[MAXN];

void Update(LL i, LL d)
{
    while(i <= n)
    {
        BIT[i] += d;
        i += i & (-i);
    }
}

LL Query(LL i)
{
    LL sum = 0;
    while(i > 0)
    {
        sum += BIT[i];
        i -= i & (-i);
    }

    return sum;
}
```

```
// Add 7 in position 0
Update(0, 7);
// Add 20 in position 8
Update(8, 20);

cout << "Sum of First 10 elements: " <<
    Query(10) << "\n";
cout << "Sum of elements in [2, 7]: " <<
    Query(7) - query(1) << "\n";
```

7.11 111 Binary Indexed Tree 2D

```
LL mx = 100, my = 100;
BIT[mx][my];

void update(LL x, LL y, LL val)
{
    LL y1;
    while (x <= mx)
    {
        y1 = y;
        while (y1 <= my)
        {
            BIT[x][y1] += val;
            y1 += (y1 & -y1);
        }
        x += (x & -x);
    }
}
LL query(LL x, LL y)
{
    LL y1, sum = 0;
    while(x)
    {
        y1 = y;
        while(y1)
        {
            sum += BIT[x][y1];
            y1 -= y1&(-y1);
        }
        x -= x&(-x);
    }
    return sum;
}
```

8 08 Geometry

8.1 101 0D Geo

```
#include<bits/stdc++.h>
using namespace std;
#define EPS 1e-9
#define PI 2*acos(0.0)

struct point
{
    double x, y;
    point() { x = y = 0.0; }
    point(double _x, double _y) : x(_x), y(_y)
        {}

    bool operator == (point other) const
    {
        return abs(x - other.x) < EPS && abs(y
            - other.y) < EPS;
    }

    bool operator < (point other) const
    {
        if(abs(x - other.x) > EPS)
            return x < other.x;
        return y < other.y;
    }

    double dist(point p1, point p2)
    {
        return sqrt((p1.x - p2.x)*(p1.x -
            p2.x) + (p1.y - p2.y)*(p1.y -
            p2.y));
    }

    //rotate point p by theta degrees CCW
        w.r.t origin (0, 0)
    point Rotate(point p, double theta)
    {
        double rad = theta * PI / 180;
        return point(p.x*cos(rad) -
            p.y*sin(rad),
                   p.x*sin(rad) +
                       p.y*cos(rad));
    }

};
```

8.2 102 2D Geo

```
#include<bits/stdc++.h>
using namespace std;
#define EPS 1e-9
#define PI 2*acos(0.0)

struct point
{
    double x, y;
    point() { x = y = 0.0; }
    point(double _x, double _y) : x(_x), y(_y)
        {}

    bool operator == (point other) const
    {
        return abs(x - other.x) < EPS && abs(y
            - other.y) < EPS;
    }

    bool operator < (point other) const
    {
        if(abs(x - other.x) > EPS)
            return x < other.x;
        return y < other.y;
    }

    double dist(point p1, point p2)
    {
        return sqrt((p1.x - p2.x)*(p1.x -
            p2.x) + (p1.y - p2.y)*(p1.y -
            p2.y));
    }

    //rotate p by theta degrees CCW w.r.t
        origin (0, 0)
    point Rotate(point p, double theta)
    {
        double rad = theta * PI / 180;
        return point(p.x*cos(rad) -
            p.y*sin(rad),
                   p.x*sin(rad) +
                       p.y*cos(rad));
    }
};

struct line
{
    //ax + by = c
    double a, b, c;

    //the answer is stored in third parameter
        (pass by reference)
    void pointsToLine(point p1, point p2, line
        &l)
    {
        if(abs(p1.x - p2.x) < EPS)
        {
            l.a = 1;
            l.b = 0;
            l.c = -p1.x;
        }
        else
        {
            double delx, dely;

            delx = p2.x - p1.x;
            dely = p2.y - p1.x;

            l.a = -dely / delx;
            l.b = 1; //we fix the value of b to
                1.0
            l.c = -(p1.x*dely - p1.y*delx) /
                delx;
        }
    }

    bool areParallel(line l1, line l2)
    {
        return (abs(l1.a-l2.a) < EPS) &&
            (abs(l1.b-l2.b) < EPS);
    }

    bool areSame(line l1, line l2)
    {
        return areParallel(l1, l2) &&
            (abs(l1.c - l2.c) < EPS);
    }
};
```

8.3 103 Closest pair

```
#include<bits/stdc++.h>
using namespace std;
long long ClosestPair(vector<pair<int, int>>
    pts)
{
    int n = pts.size();
    sort(pts.begin(), pts.end());
    set<pair<int, int>> s;

    long long best_dist = 1e18;
    int j = 0;
    for (int i = 0; i < n; ++i)
    {
        int d = ceil(sqrt(best_dist));
        while (pts[i].first - pts[j].first >=
            best_dist)
        {
            s.erase({pts[j].second,
                pts[j].first});
            j += 1;
        }

        auto it1 =
            s.lower_bound({pts[i].second - d,
            pts[i].first});
        auto it2 =
            s.upper_bound({pts[i].second + d,
            pts[i].first});

        for (auto it = it1; it != it2; ++it)
        {
            int dx = pts[i].first - it->second;
            int dy = pts[i].second - it->first;
            best_dist = min(best_dist, 1LL * dx
                * dx + 1LL * dy * dy);
        }
        s.insert({pts[i].second,
            pts[i].first});
    }
    return best_dist;
}

int main()
{
    vector<pair<int, int> > v;
    v.push_back({0, 2});
    v.push_back({0, 0});

    cout << ClosestPair(v);
}
```

8.4 104 Convex Hull

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define pii pair<ll, ll>
#define ff first
#define ss second

vector<pii> v;

bool cmp(pii a, pii b)
{
    return a.ff < b.ff || (a.ff == b.ff &&
        a.ss < b.ss);
}

bool clockWise(pii a, pii b, pii c)
{
    return
        a.ff*(b.ss-c.ss)+b.ff*(c.ss-a.ss)+c.ff*(a.ss-b.ss)
        <= 0;
    //being !clockWise and being anticlockWise
        aren’t same. look at "<="
}

bool anticlockWise(pii a, pii b, pii c)
{
    return
        a.ff*(b.ss-c.ss)+b.ff*(c.ss-a.ss)+c.ff*(a.ss-b.ss)
        >= 0;
    //being !clockWise and being anticlockWise
        aren’t same. look at ">="
}

void convex_hull()
{
    if(v.size() == 1)
        return;

    sort(v.begin(), v.end(), cmp);

    pii p1 = v[0], p2 = v.back();

    vector<pii> up, down;
    up.push_back(p1);
    down.push_back(p1);

    for (ll i = 1; i < (ll)v.size(); i++)
    {
        if (i == v.size() - 1 || clockWise(p1,
            v[i], p2))
        {
            while (up.size() >= 2 &&
                !clockWise(up[up.size()-2],
                up[up.size()-1], v[i]))
                up.pop_back();
            up.push_back(v[i]);
        }
        if (i == v.size() - 1 ||
            anticlockWise(p1, v[i], p2))
        {
            while(down.size() >= 2 &&
                !antiClockWise(down[down.size()-2],
                down[down.size()-1], v[i]))
                down.pop_back();
            down.push_back(v[i]);
        }
    }

    v.clear();
    for (ll i = 0; i < (ll)down.size(); i++)
        v.push_back(down[i]);
    for (ll i = up.size() - 2; i > 0; i--)
        v.push_back(up[i]);
}
```

8.5 105 Line Intersection

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

struct point
{
    ll x, y;
};

bool intersect(point p1, point p2, point p3,
    point p4)
{
    ll a1, b1, c1;
    a1 = p1.y - p2.y;
    b1 = p2.x - p1.x;
    c1 = p2.x*p1.y - p1.x*p2.y;

    ll a2, b2, c2;
    a2 = p3.y - p4.y;
    b2 = p4.x - p3.x;
    c2 = p4.x*p3.y - p3.x*p4.y;

    ll det = a1*b2 - b1*a2;

    if(!det)
        return 0;

    ll px = (b2*c1 - b1*c2);
    ll py = (a1*c2 - a2*c1);

    if(px < min(p1.x*det, p2.x*det) || px >
        max(p1.x*det, p2.x*det) || py <
```

```
            min(p1.y*det, p2.y*det) || py >
            max(p1.y*det, p2.y*det))
        return 0;
    if(px < min(p3.x*det, p4.x*det) || px >
            max(p3.x*det, p4.x*det) || py <
            min(p3.y*det, p4.y*det) || py >
            max(p3.y*det, p4.y*det))
        return 0;

    return 1;
}

int main()
{
    point p1{10, 0}, p2{0, 20}, p3{5, 5},
        p4{10009, 10009};
    cout << intersect(p1, p2, p3, p4);
}
```

9 09 Flow or Matching

9.1 101 Edmonds Karp

```
// Complexity O(VE^2)
LL n;
LL cap[109][109], parent[109];
bool vis[109];
vector<LL> adj[109];

LL bfs(LL s, LL t)
{
    memset(vis, 0, sizeof vis);

    LL cur, flow, new_flow;

    queue<pLL> q;
    q.push({INF, s});
    vis[s] = 1;

    while(!q.empty())
    {
        flow = q.front().ff;
        cur = q.front().ss;
        q.pop();

        for(auto nxt : adj[cur])
        {
            if(vis[nxt] == 0 && cap[cur][nxt])
            {
                parent[nxt] = cur;
                new_flow = min(flow,
                    cap[cur][nxt]);

                if(nxt == t)
                    return new_flow;

                q.push({new_flow, nxt});
                vis[nxt] = 1;
            }
        }
    }

    return 0;
}

LL maxflow(LL s, LL t)
{
    LL prev, cur;
    LL flow = 0, new_flow;

    while(new_flow = bfs(s, t))
    {
        flow += new_flow;

        for(cur = t; cur != s; cur = prev)
        {
            prev = parent[cur];
            cap[prev][cur] -= new_flow;
            cap[cur][prev] += new_flow;
        }
    }

    return flow;
}

/*
memset(cap, 0, sizeof cap);
cap[a][b] += w;
cap[b][a] += w;
adj[a].pb(b);
adj[b].pb(a);

cout << maxflow(s, t);
*/
```

10 10 String

10.1 101 Hashing

```
LL base = 31;

LL n, p[MAXN];
LL preHash[MAXN];
LL sufHash[MAXN];

void initHash()
{
    LL i;

    p[0] = 1;
    for(i = 1; i < MAXN; i++)
        p[i] = (p[i-1] * base) % MOD;
}

void createHash (string s)
{
    LL i;

    n = s.size();

    for(i = 1; i <= n; i++)
    {
        preHash[i] = (preHash[i-1] * base) %
            MOD;
        preHash[i] = (preHash[i] + s[i-1] -
            'a' + 1) % MOD;
    }
    for(i = n; i > 0; i--)
    {
        sufHash[i] = (sufHash[i+1] * base) %
            MOD;
        sufHash[i] = (sufHash[i] + s[i-1] -
            ’a’ + 1) % MOD;
    }
}
```

10.2 102 KMP

```
// Longest Prefix which is also a Suffix
LL LPS[MAXN], n;

void KMP(string s)
{
    LL i, j, n = s.size();

    LPS[0]=0;

    for (i = 1; i < n; i++)
    {
        j = LPS[i-1];

        while (j > 0 && s[i] != s[j])
            j = LPS[j-1];

        if (s[i] == s[j])
            j++;

        LPS[i] = j;
    }
}
```

10.3 103 Z algorithm

```
vector<LL> z_function(string s)
{
    LL i, l, r, n = s.size();
    vector<LL> Z(n);
    Z[0] = 0;

    for(i = 1, l = 0, r = 0; i < n; i++)
    {
        if(i <= r) //This condition is false
            when i=1
            Z[i] = min(r-i+1, Z[i-l]);
        while(i+Z[i] < n && s[Z[i]] ==
            s[i+Z[i]])
            Z[i]++; //if Z[1] has previous
                value, it will cause problem
                here

        if(i+Z[i] - 1 > r)
        {
            l = i;
            r = i+Z[i]-1;
        }
    }

    return Z;
}
```

10.4 104 Manacher’s Algo

```
LL pal[300009], n;
string s = "#";

void manacher()
{
    LL i, l, r, k;

    for(i = 0, l = 0, r = -1; i < n; i++)
    {
        if(i > r)
            k = 1;
        else
            k = min(pal[l + r - i], r - i + 1);

        while(i-k >= 0 && i+k < n && s[i-k] ==
            s[i+k])
            k++;

        pal[i] = k;
        k--;

        if(i+k > r)
        {
            l = i-k;
            r = i+k;
        }
    }
}

int main()
{
    LL i, ans = 0;
    string stemp;

    cin >> n >> stemp;

    for(i = 0; i < stemp.size(); i++)
    {
        s.push_back(stemp[i]);
        s.push_back(’#’);
    }

    n = s.size();

    manacher();

    for(i = 0; i < n; i++)
        cout << pal[i] << " ";
}
```

10.5 105 Suffix Array

```
vector<pair<pii, int>> bucket[MAXN];
void radix_sort(vector<pair<pii, int>> &v)
{
    int i, j, n = v.size();

    for(i = 0; i < MAXN; i++)
        bucket[i].clear();

    for(i = 0; i < n; i++)
        bucket[v[i].ff.ss + 1].push_back(v[i]);
    v.clear();
    for(i = 0; i < MAXN; i++)
    {
        for(auto u : bucket[i])
            v.push_back(u);
        bucket[i].clear();
    }

    for(i = 0; i < n; i++)
        bucket[v[i].ff.ff].push_back(v[i]);

    v.clear();
    for(i = 0; i < MAXN; i++)
    {
        for(auto u : bucket[i])
            v.push_back(u);
    }
}

vector<int> get_SA(string s)
{
    int i, len, cnt, n = s.size();
    vector<int> prev(n), sa(n);
    vector<pair<pii, int>> curr;

    for(i = 0; i < n; i++)
        prev[i] = s[i];

    for(len = 2; len <= 2*n; len *= 2)
    {
        curr.clear();
        for(i = 0; i < n; i++)
        {
            if(i+len/2 >= n)
                curr.push_back({{prev[i], -1}, i});
            else
                curr.push_back({{prev[i], prev[i+len/2]}, i});
        }

        radix_sort(curr);
        // sort(curr.begin(), curr.end());

        for(i = cnt = 0; i < n; i++)
        {
            if(i == 0 || curr[i].ff == curr[i-1].ff)
                prev[curr[i].ss] = cnt;
            else
                prev[curr[i].ss] = ++cnt;
        }
    }

    for(i = 0; i < n; i++)
        sa[prev[i]] = i;

    return sa;
}

vector<int> get_LCP(string s, vector<int>&sa)
{
    int i, j, k, n = s.size();

    vector<int> lcp(n-1, 0), rank(n, 0);

    for(i = 0; i < n; i++)
        rank[sa[i]] = i;

    for(i = 0, k = 0; i < n; i++)
    {
        if(rank[i] == n-1)
        {
            k = 0;
            continue;
        }

        j = sa[rank[i] + 1];
        while(i+k < n && j+k < n && s[i+k] == s[j+k])
            k++;

        lcp[rank[i]] = k;

        k = max(k-1, 0);
    }

    return lcp;
}
```

11 11 Miscellaneous

11.1 101 Boyer-Moore Majority Voting

```
//Finds the element that is present for more than N/2 times (if there is any) in O(n) time, O(1) space

ll arr[maxN];

int findMajority()
{
    ll major, cnt = 0, n = arr.size();

    for(i = 0; i < n; i++)
    {
        if(cnt == 0)
        {
            major = arr[i];
            cnt = 0;
        }

        if(arr[i] == major)
            cnt++;
        else
            cnt--;
    }

    cnt = 0;
    for(i = 0; i < n; i++)
    {
        if(arr[i] == major)
            cnt++;
    }

    if(cnt <= n/2)
        return -1;
    else
        return major;
}
```