

```

#define Fast ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define Freed freopen ("Oin.txt","r",stdin);
#define Fout freopen ("Oout.txt","w",stdout);
#define ll long long
#define pb push_back
#define mp make_pair
#define pi acos(-1.0)
#define Mod 1000000007
#define limit 1000008
using namespace std;
///Iterative
ll bigmod(ll b,ll p)
{
    ll ans=1;
    while(p)
    {
        if(p&1)
            ans = (ans*b)%Mod;
        b = (b*b)%Mod;
        p = p/2;
    }
    return ans;
}
2.*// Totient O(n*logn)
seive();
for(int i=2; i<limit; i++) coprime[i]=i;
for(int i=2; i<limit; i++)
{
    if(!vis[i])
        for(int j=i; j<limit; j+=i)
            coprime[j] = (coprime[j]*(i-1))/i;
}
3.*// Hashing
void cumforwardhashing(string s,ll base,ll mod,ll A[])
{
    ll i,n=s.size();
    A[0] = s[0]-'a'+1;
    for(i=1; i<n; i++)
    {
        A[i] = ((A[i-1]*base)+s[i]-'a'+1)%mod;
    }
}
void cumbackwardhashing(string s,ll base,ll mod,ll A[])
{
    ll i,n=s.size();
    A[n-1] = s[n-1]-'a'+1;
    for(i=n-2; i>=0; i--)
    {
        A[i] = ((A[i+1]*base)+s[i]-'a'+1)%mod;
    }
}
4.*//structure maping
struct pos
{
    int cx,cy,nx,ny;
};
bool operator< (pos a,pos b )
{
    //there are lots of fact for this return
    if(a.cx!=b.cx) return a.cx<b.cx;
    else if(a.cy!=b.cy) return a.cy<b.cy;
    else if(a.nx!=b.nx) return a.nx<b.nx;
    else return a.ny<b.ny;
}

```

### **5.\*//DSU u,v ///DSU O(n)**

```

int pr[limit];
int Find(int u)
{
    if(pr[u]==u) return u;
    return pr[u] = Find(pr[u]);
}
void dsu(int u,int v)
{
    pr[Find(v)] = Find(u);
}
6.*//DSU on tree.
ll cnt[maxn];
bool big[maxn];
void add(ll v, ll p, ll x)
{
    cnt[ col[v] ] += x;
    for(auto u: g[v])
    {
        if(u != p && !big[u])
            add(u, v, x)
    }
}
void del(ll v, ll p, ll x)
{
    cnt[ col[v] ] -= x;
    for(auto u: g[v])
    {
        if(u != p && !big[u])
            add(u, v, x)
    }
}
void dfs(ll v, ll p, bool keep)
{
    ll mx = -1, bigChild = -1;
    for(auto u : g[v])
    {
        if(u != p && sz[u] > mx)
        {
            mx = sz[u];
            bigChild = u;
        }
    }
    for(auto u : g[v])
    {
        if(u != p && u != bigChild)
            dfs(u, v, 0); // run a dfs on small childs and clear them from cnt
    }
    if(bigChild != -1)
    {
        dfs(bigChild, v, 1);
        big[bigChild] = 1; // bigChild marked as big and not cleared from
        cnt
    }
    add(v, p, 1); //added to the ans subtree of v
}
if(bigChild != -1)
    big[bigChild] = 0;
if(keep == 0)
    del(v, p, 1); //delete subtree of v
}

```

7. ///**max flow Ford–Fulkerson, Time: atmost  $n*t \leq 300$ , memory:  $O(n^2)$**

```

vector<ll> g[limit];
bool vis[limit];
ll cost[limit][limit];
ll path(ll u,ll w)
{
    if(u==destination)
    {
        flow += w;
        return w;
    }
    vis[u]=1;
    for(ll v: g[u])
    {
        if(vis[v] || cost[u][v]==0) continue;
        ll x = path(v,min(w,cost[u][v]));
        if(x)
        {
            cost[v][u] += x;
            cost[u][v] -= x;
            return x;
        }
    }
    return 0;
}
8./// Maximum Spanning Tree. Kruskal's Algorithm  $O(E \log E)$  with DSU
struct E
{
    ll u,v,w;
};

bool operator<(E a,E b)
{
    return a.w>b.w;
}

ll p[limit];
ll Find(ll u)      // this part is for disjoint set union , initially p[x] = x;
{
    if(p[u]==u) return u;
    return p[u] = Find(p[u]);
}

void MST()
{
    ll node,edge,i,j;
    E z;          // z is structure type variable
    vector <E> vec;
    cin >> node >> edge;
    ll u,v,w;
    for(i=0; i<edge; i++)
    {
        cin >> u >> v >> w;
        z.u = u;
        z.v = v;
        z.w = w;
        vec.pb(z);
    }
    sort(vec.begin(),vec.end());
}

for(i=0; i<limit; i++)           // set DSU, parent of i is i
    p[i] = i;

vector<pair<ll, ll> > g[limit];

ll ans = 0;
for(i=0; i<edge; i++)
{
    z = vec[i];
    u = z.u;
    v = z.v;
    w = z.w;
    if(Find(u)!=Find(v))
    {
        ans += w;      // ans is the maximum cost
        p[u] = p[v];  // parent of u is v
        g[u].pb(mp(v,w));
        g[v].pb(mp(u,w)); // g Maximum spanning tree only
    }
}
cout << ans << endl;
}

9.///Merge sort  $O(n \log n)$ 
void Mergesort(int l,int r)
{
    if(l==r) return ;
    int i,j,mid=(l+r)/2;
    Mergesort(l,mid);
    Mergesort(1+mid,r);
    v.clear();
    for(i=l,j=mid+1; ; )
    {
        if(i>mid && j>r) break;
        if(i>mid)
        {
            v.pb(D[j]);
            j++;
        }
        else if(j>r)
        {
            v.pb(D[i]);
            i++;
        }
        else if(D[i]<=D[j])
        {
            v.pb(D[i]);
            i++;
        }
        else
        {
            v.pb(D[j]);
            j++;
        }
    }
    for(i=l,j=0; i<=r; i++,j++)
    {
        D[i] = v[j];
    }
}

```

## 10. //Number of Divisor for 1 to n. Complexity less than O(n\*logn)

```
ll nod[limit];
void NOD(ll n)
{
    ll i,j;
    for(i=1; i<=n; i++)
    {
        for(j=i; j<=n; j+=i)
            nod[j]++;
    }
}
///Sum of Divisor for 1 to n. Complexity less than O(n*logn)
ll sod[limit];
void SOD(ll n)
{
    ll i,j;
    for(i=1; i<=n; i++)
    {
        for(j=i; j<=n; j+=i)
            sod[j] += i;
    }
}
```

## 11. //nPr O(r)

```
ll nPr(ll n,ll r)
{
    ll ans=1;
    for(ll i=n,j=0; j<r; i--,j++)
        ans = (ans*i)%Mod;
    return ans;
}
///nCr O(n*r)
long long int ncr[2000][2000];
long long int nCr(long long int n, long long int r)
{
    if(n==r) return 1;
    if(r==1) return n;
    if(ncr[n][r]) return ncr[n][r];
    return ncr[n][r] = (nCr(n-1,r-1)+nCr(n-1,r))%Mod;
}
///nCr O(1) when Mod is prime
ll fact[limit],factorialNumInverse[limit],naturalNumInverse[limit];
// Function to precompute inverse of numbers
void InverseofNumber()
{
    naturalNumInverse[0] = naturalNumInverse[1] = 1;
    for (int i = 2; i < limit; i++)
        naturalNumInverse[i] = naturalNumInverse[Mod % i] * (Mod -
Mod / i) % Mod;
}
// Function to precompute inverse of factorials
void InverseofFactorial()
{
    factorialNumInverse[0] = factorialNumInverse[1] = 1;
    for (int i = 2; i < limit; i++)
        factorialNumInverse[i] = (naturalNumInverse[i] * *
factorialNumInverse[i - 1]) % Mod;
}
void factorial()
{
    fact[0] = 1;
    for (int i = 1; i < limit; i++) {
        fact[i] = (fact[i - 1] * i) % Mod;
    }
}
```

```
}
```

## 12. //PBDS or Order set/multiset/map any operation O(logn)

```
///insert,erase,size,order_of_key,find_by_order
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using namespace std;
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update>
new_data_type;
```

```
int main()
{
    int t,time,i;
    new_data_type os;
    os.insert(5);
    // Deleting 2 from the set if it exists
    if (o_set.find(2) != o_set.end())
        o_set.erase(o_set.find(2));
    // Finding the second smallest element in the set
    cout << *(o_set.find_by_order(1))
        << endl;
    // elements strictly less than k=4
    cout << o_set.order_of_key(4)
        << endl;
    return 0;
}
```

## 13. //prime factorization

```
cin >> n ;
vector <int> Div;
for(i=0; prime[i]*prime[i]<=n; i++)
{
    while(n%prime[i]==0)
    {
        Div.pb(prime[i]);
        n/=prime[i];
    }
}
if(n>1)
    Div.pb(n);
for(i=0; i<Div.size(); i++) cout << Div[i] << " ";
```

```

14. //seive
bool vis[limit];
vector<int>prime;
void seive()
{
    vis[0]=vis[1]=1;
    for(int i=4; i<limit; i+=2) vis[i] = 1;
    for(int i=3; i*i<limit; i+=2)
    {
        if(vis[i]) continue;
        for(int j=i*i; j<limit; j+=2*i)
        {
            vis[j] = 1;
        }
    }
    prime.pb(2);
    for(int j=3; j<limit; j+=2)
        if(vis[j]==0)
            prime.pb(j);
}

15. //segree() O(nlogn)
ll A[limit],T[4*limit];
ll segtree(ll node,ll l,ll r)
{
    if(l==r) return T[node]=A[l];
    ll mid=(l+r)/2;
    T[2*node] = segtree(2*node,l,mid);
    T[2*node+1] = segtree(2*node+1,mid+1,r);
    return T[node] = T[2*node]^T[2*node+1];
}
///segupdate O(logn)
ll segupdate(ll node,ll l,ll r,ll pos,ll key)
{
    if(l==r && l==pos) return T[node] = T[node]^key;
    ll mid=(l+r)/2;
    if(pos<=mid)
        T[2*node] = segupdate(2*node,l,mid,pos,key);
    else
        T[2*node+1] = segupdate(2*node+1,mid+1,r,pos,key);
    return T[node] = T[2*node]^T[2*node+1];
}
///segquery O(logn)
ll segquery(ll node,ll l,ll r,ll from,ll to)
{
    if(l>=from && r<=to) return T[node];
    if(l>to || r<from) return 0;
    ll mid=(l+r)/2;
    return
segquery(2*node,l,mid,from,to)^segquery(2*node+1,mid+1,r,from,t
o);
}

void solve(ll t)
{
    ll i,j,n,q,m,k;
    cin >> n >> q;
    for(i=1; i<=n; i++)
        cin >> A[i];
    segtree(1,1,n);
    for(i=0; i<q; i++)
    {
        ll type, pos, key, from, to;

```

```

        cin >> type;
        if(type==1)
        {
            cin >> pos >> key;
            segupdate(1,1,n,pos,key);
        }
        else
        {
            cin >> from >> to;
            cout << segquery(1,1,n,from,to) << endl;
        }
    }
    return ;
}

16.// SSC
vector<ll>g[limit],tg[limit];
bool vis[limit];
ll st[limit],ft[limit];
ll tme;

void dfs(ll u)
{
    st[u] = tme++;
    vis[u] = 1;
    for(ll v:g[u])
    {
        if(vis[v]==0)
        {
            dfs(v);
        }
    }
    ft[u] = tme++;
}

void dfs2(ll u)
{
    //cout << u << "->";
    vis[u] = 1;
    for(ll v:tg[u])
    {
        if(vis[v]==0)
        {
            dfs2(v);
        }
    }
}

/// Strongly Connected Component(SCC) O(Node+Edge)
void SCC(ll t)
{
    ll i,j;
    ll n,e,u,v;
    cin >> n >> e ;

    //clear
    for(i=0; i<n+5; i++)
    {
        g[i].clear();
        tg[i].clear();
        vis[i] = 0;
        st[i] = ft[i] = 0;
    }
}

```

```

tme = 1;

for(i=0; i<e; i++)
{
    cin >> u >> v;
    g[u].pb(v);
    tg[v].pb(u);
}
for(i=1; i<=n; i++)
{
    if(vis[i]==0)
    {
        dfs(i);
    }
}
vector<pair<ll, ll> > seq;

for(i=1; i<=n; i++)
{
    seq.pb(mp(ft[i],i));
}
sort(seq.begin(),seq.end());

ll ct=0;
memset(vis,0,sizeof(vis));
for(i=n-1; i>=0; i--)
{
    u = seq[i].second;
    if(vis[u]==0)
    {
        dfs2(u);
        ct++;
        cout << endl;
    }
}
cout << "Total number of SCC: " << ct << endl;
return ;
}

```

### 17. //Query range , Heavy Light Tricks

```

const ll block = 350 ;
ll d[limit],cum[limit],PS[350][limit];

void solve(ll t)
{
    ll i,j,n,m,k,q;
    string s;
    cin >> n >> q;
    //block = sqrt(n);

    cum[0] = 0;
    for(i=1; i<=n; i++)
    {
        cin >> d[i];
        cum[i] = cum[i-1]+d[i];
    }

    memset(PS, 0, sizeof(PS));
    for(k=1; k<block; k++)
    {
        for(ll pos = n-k; pos<=n+1; pos++)
            PS[k][pos] = 0;
        for(ll pos=n-k+1; pos>0; pos--)

```

```

            PS[k][pos] = (cum[pos+k-1]-cum[pos-1]) - PS[k][pos+k];
            //cout << k << " " << pos << " " << PS[k][pos] << endl;
        }
    }
    for(i=0; i<q; i++)
    {
        ll l,r;
        cin >> l >> r >> k;
        ll ans = 0;
        if(k>=block)
        {
            ll sign = 1;
            for(j=l; j<r; j+=k)
            {
                ans += (cum[j+k-1]-cum[j-1])*sign;
                sign = sign*-1;
            }
        }
        else
        {
            ans = PS[k][l]; //cout << ans << " akhane\n";
            if(((r-l+1)/k)&1)
                ans += PS[k][r+1];
            else
                ans -= PS[k][r+1];
        }
        cout << ans << endl;
    }
    return ;
}


```

### 18. //MO's Algorithm

```

ll cnt[limit];
ll frequ[limit];
ll mxfrequ;

void add(ll x)
{
    cnt[frequ[x]]--;
    frequ[x]++;
    if(mxfrequ < frequ[x]) mxfrequ = frequ[x];
    cnt[frequ[x]]++;
}

void sub(ll x)
{
    cnt[frequ[x]]--;
    if(mxfrequ==frequ[x] && cnt[frequ[x]]==0) mxfrequ--;
    frequ[x]--;
    cnt[frequ[x]]++;
}

/*
    split all the query in sqrt(n) block according to left side
    and in each block all data are sorted according to right side
*/
///MO's algorithm O(n*sqrt(n))
ll block;
struct st
{
    ll l,r,idx;
}
```

```

bool operator<(const st& a) const
{
    if(l/block != a.l/block) return l/block < a.l/block;
    return r < a.r;
}
};

//MO's

void MO(II t)
{
    II n,q,i;
    cin >> n >> q ;
    II d[n+5]; d[0] = 0;
    vector<II> out(q+5);

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

    //MO's
    block = sqrt(n);
    st query[q+5];
    for(i=0; i<q; i++)
    {
        cin >> query[i].l >> query[i].r;
        query[i].idx = i;
    }
    sort(query,query+q);
    //MO's

    II left = 1,right = 0;
    for(i=0; i<q; i++)
    {
        //cout <<i<<" "<<left<<" "<<right<<" "<<query[i].l<<" "
        "<<query[i].r<<endl;
        while(right < query[i].r)
        {
            right++;
            add(d[right]);
        }
        while( right > query[i].r)
        {
            sub(d[right]);
            right--;
        }
        while(left < query[i].l)
        {
            sub(d[left]);
            left++;
        }
        while(left > query[i].l)
        {
            left--;
            add(d[left]);
        }
        out[query[i].idx] = 1 + max(0, mxfrequ - (query[i].r - query[i].l + 2
        - mxfreq));
        //cout      <<left<<"      "<<right<<"      "<<mxfrequ<<" 
        "<<out[query[i].idx]<<endl;
    }
    for(i=0; i<q; i++)
        cout << out[i] << endl;
    return ;
}

```

**19.///Sparse\_Table ST O(n\*logn)**

```

II sparse_table[limit][25];
void ST(II t)
{
    II i,j,n,m,k,q;

    cin >> n >> q;
    II d[n+5];
    for(i=1; i<=n; i++)
        cin >> d[i];

    II sparse_table[n+5][25];
    for(i=1; i<=n; i++)
    {
        sparse_table[i][0] = d[i];      //min/max for length 1
    }
    II len;
    for(j=1, len=1; j<25; j++, len=len*2)
    {
        for(i=1; i<=n; i++)
        {
            if((i+len*2-1)>n) break;      //out of bound
            sparse_table[i][j] = min(sparse_table[i][j-1],
            sparse_table[i+len][j-1]);
        }
    }

    for(i=0; i<q; i++)
    {
        II l,r;
        cin >> l >> r;
        II len = (r-l);
        j = log2(len);
        len = 1 << j;
        cout << min(sparse_table[l][j], sparse_table[r-len+1][j]) << endl;
        //minimum between l+len and r-len
    }
    return ;
}

20.// Lis(largest independent set)
#include <bits/stdc++.h>
using namespace std;

// A utility function to find max of two
int max(int x, int y) { return (x > y) ? x : y; }

/* A binary tree node has data, pointer
to left child and a pointer to
right child */
class node
{
public:
    int data;
    int liss;
    node *left, *right;
};

// A memoization function returns size
// of the largest independent set in
// a given binary tree
int LISS(node *root)
{
    if (root == NULL)
        return 0;

```

```

if (root->liiss)
    return root->liiss;

if (root->left == NULL && root->right == NULL)
    return (root->liiss = 1);

// Calculate size excluding the current node
int liiss_excl = LISS(root->left) +
LISS(root->right);

// Calculate size including the current node
int liiss_incl = 1;
if (root->left)
    liiss_incl += LISS(root->left->left) +
LISS(root->left->right);
if (root->right)
    liiss_incl += LISS(root->right->left) +
LISS(root->right->right);

// Maximum of two sizes is LISS, store it for future uses.
root->liiss = max(liiss_incl, liiss_excl);

return root->liiss;
}

// A utility function to create a node
node* newNode(int data)
{
    node* temp = new node();
    temp->data = data;
    temp->left = temp->right = NULL;
    temp->liiss = 0;
    return temp;
}

// Driver code
int main()
{
    // Let us construct the tree
    // given in the above diagram
    node *root      = newNode(20);
    root->left       = newNode(8);
    root->left->left     = newNode(4);
    root->left->right    = newNode(12);
    root->left->right->left = newNode(10);
    root->left->right->right = newNode(14);
    root->right       = newNode(22);
    root->right->right   = newNode(25);

    cout << "Size of the Largest Independent Set is " << LISS(root);

    return 0;
}

```

## 21 // Least common ancestor

```

#include <iostream>

using namespace std;

// A Binary Tree Node
struct Node
{
    struct Node *left, *right;
    int key;
};

// Utility function to create a new tree Node
Node* newNode(int key)
{
    Node *temp = new Node();
    temp->key = key;
    temp->left = temp->right = NULL;
    return temp;
}

// This function returns pointer to LCA of two given values n1 and n2.
// This function assumes that n1 and n2 are present in Binary Tree
struct Node *findLCA(struct Node* root, int n1, int n2)
{
    // Base case
    if (root == NULL) return NULL;

    // If either n1 or n2 matches with root's key, report
    // the presence by returning root (Note that if a key is
    // ancestor of other, then the ancestor key becomes LCA
    if (root->key == n1 || root->key == n2)
        return root;

    // Look for keys in left and right subtrees
    Node *left_lca  = findLCA(root->left, n1, n2);
    Node *right_lca = findLCA(root->right, n1, n2);

    // If both of the above calls return Non-NULL, then one key
    // is present in once subtree and other is present in other,
    // So this node is the LCA
    if (left_lca && right_lca) return root;

    // Otherwise check if left subtree or right subtree is LCA
    return (left_lca != NULL)? left_lca:
right_lca;
}

// Driver program to test above functions
int main()
{
    // Let us create binary tree given in the above example
    Node * root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    cout << "LCA(4, 5) = " << findLCA(root, 4, 5)->key;
    cout << "nLCA(4, 6) = " << findLCA(root, 4, 6)->key;
    cout << "nLCA(3, 4) = " << findLCA(root, 3, 4)->key;
    cout << "nLCA(2, 4) = " << findLCA(root, 2, 4)->key;
    return 0;
}

```

```

23.// Convex hull set1
#include <bits/stdc++.h>
using namespace std;

struct Point
{
    int x, y;
};

// To find orientation of ordered triplet (p,
q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0; // colinear
    return (val > 0)? 1: 2; // clock or
counterclock wise
}

// Prints convex hull of a set of n points.
void convexHull(Point points[], int n)
{
    // There must be at least 3 points
    if (n < 3) return;

    // Initialize Result
    vector<Point> hull;

    // Find the leftmost point
    int l = 0;
    for (int i = 1; i < n; i++)
        if (points[i].x < points[l].x)
            l = i;

    // Start from leftmost point, keep moving
    // counter-clockwise
    // until reach the start point
    again. This loop runs O(h)
    // times where h is number of points in
    result or output.
    int p = l, q;
    do
    {
        // Add current point to result
        hull.push_back(points[p]);

        // Search for a point 'q' such that
        orientation(p, q,
                    // x) is counter-clockwise for all
        points 'x'. The idea
        // is to keep track of last visited
        most counter-clock-
        // wise point in q. If any point 'i'
        is more counter-clock-
        // wise than q, then update q.
        q = (p+1)%n;
        for (int i = 0; i < n; i++)
        {
            // If i is more counter-clockwise
            // than current q, then
            // update q
            if (orientation(points[p],
                           points[i], points[q]) == 2)
                q = i;
        }
        // Now q is the most counter-clockwise
        // with respect to p
        // Set p as q for next iteration, so
        // that q is added to
        // result 'hull'
        p = q;

    } while (p != l); // While we don't come
    to first point

    // Print Result
    for (int i = 0; i < hull.size(); i++)
        cout << "(" << hull[i].x << ", "
              << hull[i].y << ")\n";
}

// Driver program to test above functions
int main()
{
    Point points[] = {{0, 3}, {2, 2}, {1, 1},
                      {2, 1},
                      {3, 0}, {0, 0}, {3,
3}};

    int n = sizeof(points)/sizeof(points[0]);
    convexHull(points, n);
    return 0;
}

23./Convex hull set2
#include <iostream>
#include <stack>
#include <stdlib.h>
using namespace std;

struct Point
{
    int x, y;
};

// A global point needed for sorting points
// with reference
// to the first point Used in compare
function of qsort()
Point p0;

// A utility function to find next to top in
// a stack
Point nextToTop(stack<Point> &S)
{
    Point p = S.top();
    S.pop();
    Point res = S.top();
    S.push(p);
    return res;
}

// A utility function to swap two points
void swap(Point &p1, Point &p2)
{
    Point temp = p1;
    p1 = p2;
    p2 = temp;
}

// A utility function to return square of
// distance
// between p1 and p2
int distSq(Point p1, Point p2)
{
    return (p1.x - p2.x)*(p1.x - p2.x) +
           (p1.y - p2.y)*(p1.y - p2.y);
}

```

```

// To find orientation of ordered triplet (p,
q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0; // colinear
    return (val > 0)? 1: 2; // clock or
                           counterclock wise
}

// A function used by library function
qsort() to sort an array of
// points with respect to the first point
int compare(const void *vp1, const void *vp2)
{
    Point *p1 = (Point *)vp1;
    Point *p2 = (Point *)vp2;

    // Find orientation
    int o = orientation(p0, *p1, *p2);
    if (o == 0)
        return (distSq(p0, *p2) >= distSq(p0,
*p1))? -1 : 1;

    return (o == 2)? -1: 1;
}

// Prints convex hull of a set of n points.
void convexHull(Point points[], int n)
{
    // Find the bottommost point
    int ymin = points[0].y, min = 0;
    for (int i = 1; i < n; i++)
    {
        int y = points[i].y;

        // Pick the bottom-most or chose the
        left
        // most point in case of tie
        if ((y < ymin) || (ymin == y &&
            points[i].x < points[min].x))
            ymin = points[i].y, min = i;
    }

    // Place the bottom-most point at first
    position
    swap(points[0], points[min]);

    // Sort n-1 points with respect to the
    first point.
    // A point p1 comes before p2 in sorted
    output if p2
        // has larger polar angle (in
    counterclockwise
        // direction) than p1
    p0 = points[0];
    qsort(&points[1], n-1, sizeof(Point),
compare);

    // If two or more points make same angle
    with p0,
        // Remove all but the one that is farthest
    from p0
}

```

// Remember that, in above sorting, our  
criteria was

// to keep the farthest point at the end  
when more than

// one points have same angle.

int m = 1; // Initialize size of modified  
array

for (int i=1; i<n; i++)

{

// Keep removing i while angle of i  
and i+1 is same

// with respect to p0

while (i < n-1 && orientation(p0,
points[i],
points[i+1]) == 0)
 i++;
}

points[m] = points[i];
m++; // Update size of modified array
}

// If modified array of points has less  
than 3 points,  
// convex hull is not possible
if (m < 3) return;

// Create an empty stack and push first  
three points
// to it.
stack<Point> S;
S.push(points[0]);
S.push(points[1]);
S.push(points[2]);

// Process remaining n-3 points
for (int i = 3; i < m; i++)
{
 // Keep removing top while the angle
 formed by
 // points next-to-top, top, and
 points[i] makes
 // a non-left turn
 while (S.size()>1 &&
orientation(nextToTop(S), S.top(), points[i])
!= 2)
 S.pop();
 S.push(points[i]);
}

// Now stack has the output points, print
contents of stack
while (!S.empty())
{
 Point p = S.top();
 cout << "(" << p.x << ", " << p.y
<< ")" << endl;
 S.pop();
}

// Driver program to test above functions
int main()
{
 Point points[] = {{0, 3}, {1, 1}, {2, 2},
{4, 4},
{0, 0}, {1, 2}, {3, 1},
{3, 3}};
 int n = sizeof(points)/sizeof(points[0]);
 convexHull(points, n);
 return 0;
}

\* एकी बालों मिलाकर बास्तविक दृश्यावार =  $P_h$  होता है।

\* चूंगा न अस्तक दैनि दृश्यावार  $P_h = \pi r^2$

\* बहुभुज के केन्द्र परिभ्रमण (१)

$$\theta = (n-2) * 180$$

$$A = \frac{n}{2} \times l \times h = \frac{n}{2} \times a \times h$$

\* समद्वय वृत्त का क्षेत्रफल (२)

$$A = \frac{1}{2} \times a \times h$$

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

$$A = \frac{1}{2} ab \sin C$$

$$A = \frac{abc}{4R}$$

$$A = Sp$$

\* वृत्ताभास का क्षेत्रफल (३)

$$A = \frac{\pi r^2 \theta}{360}$$

\* समाकृत शब्द (AP) : अधिक वर्तुल सार्वजनिक व्यवहार आहात।

\* प्राथमिक - शब्द (GP) : अधिक वर्तुल सार्वजनिक व्यवहार आहात।

\* Special (AP/GP) : Power ने अधिक वर्तुल सार्वजनिक व्यवहार आहात। AP/GP असकते।

$A_1 = 6, 9, 12, 15, 18, \dots, 126$

$T_n = 2 + 5 + 8 + 11 + 14 + \dots + n$

\* अखंक अद्वय =  $a$

\* सार्वजनिक अद्वय ( $d$ ) = अधिक अद्वय - अप्रथम अद्वय।

\*  $n$  वर्तुल अद्वय ( $T_n$ ) = अप्रथम अद्वय +  $(n-1) \times \text{सर्वांगी अद्वय}$

\*  $n$  वर्तुल अद्वय ( $T_n$ ),  $n = \frac{n(\text{अप्रथम अद्वय} + \text{अखंक अद्वय})}{2} + 1$

\* अखंक अद्वय संग्रही (Sum) ( $S_n$ ) =  $\frac{n(\text{अप्रथम अद्वय} \times (\text{अखंक अद्वय} + (n-1) \times \text{सर्वांगी अद्वय}))}{2}$

①  $d = T_2 - T_1$

②  $T_n = a + (n-1) \times d$

③  $n = \frac{T_n - a}{d} + 1$

④  $S_n = \frac{n}{2} [2a + (n-1)d]$

⑤  $1 + 2 + 3 + 4 + \dots + n = \frac{n(n+1)}{2}$

⑥  $2 + 4 + 6 + 8 + \dots + 2n = n(n+1)$

⑦  $1 + 3 + 5 + 7 + \dots + (2n-1) = n^2$

[अखंक अद्वय]

$S_n$  अद्वय अद्वय  
अप्रथम  $T_1$  अद्वय अद्वय  
अद्वय अद्वय  $T_2$  अद्वय  
अद्वय अद्वय  $T_3$  अद्वय  
अद्वय अद्वय  $T_4$  अद्वय  
अद्वय अद्वय  $T_5$  अद्वय  
अद्वय अद्वय  $T_6$  अद्वय  
अद्वय अद्वय  $T_7$  अद्वय  
अद्वय अद्वय  $T_8$  अद्वय  
अद्वय अद्वय  $T_9$  अद्वय

Priority Queue : normally it's contain non increasing order.  $\rightarrow$  as Max heap

\* Priority queue <int> PQ;

PQ.push(10);  $\rightarrow$  10

PQ.push(20);  $\rightarrow$  20, 10

PQ.push(15);  $\rightarrow$  20, 15, 10

PQ.top();  $\rightarrow$  return 20

PQ.pop();  $\rightarrow$  15, 10

PQ.size();  $\rightarrow$  return 2

\* Priority queue as non decreasing order  $\rightarrow$  as min heap

Priority queue <int, vector<int>, greater<int>> PQ;

PQ.push(10);  $\rightarrow$  10

PQ.push(20);  $\rightarrow$  10, 20

PQ.push(15);  $\rightarrow$  10, 15, 20

PQ.top();  $\rightarrow$  return 10

PQ.pop();  $\rightarrow$  15, 20

PQ.size();  $\rightarrow$  return 2

Push, Pop operation  $O(\log n)$   
Top, Size, empty operation  $O(1)$

FACT

$\sum_{i=1}^n i^k = \frac{n(n+1)(2n+1)}{6}$

$\sum_{i=1}^n i^3 = \frac{n(n+1)^2}{2}$

प्राथमिक शब्द (AP) : अधिक वर्तुल सार्वजनिक व्यवहार आहात।

$A_1 = 1, 2, 4, 8, 16, 32, \dots$

$A_2 = 3 - 9 + \frac{16}{3} - \dots$

$A_3 = 1 + 3 + 9 + 27 + \dots$

\* अखंक अद्वय =  $a$

\* सार्वजनिक अद्वय  $r = \frac{\text{अप्रथम अद्वय}}{\text{अखंक अद्वय}}$

\*  $n$  वर्तुल अद्वय,  $T_n = \text{अप्रथम अद्वय} \times (राशी अद्वय)^{n-1}$

\*  $n$  वर्तुल अद्वय,  $n =$

\* अखंक अद्वय गुणक,  $S_n = \frac{\text{अप्रथम अद्वय} \times (राशी अद्वय)^n - 1}{r - 1}$

\* अखंक अद्वय गुणक,  $S_n = \frac{a \times (1 - r^n) \times (1 - r^{n-1})}{(1 - r)^2}$

①  $r = \frac{T_2}{T_1} = \dots = \frac{T_n}{T_{n-1}}$

②  $T_n = a r^{n-1}$

③  $n = \frac{T_n}{a r^{n-1}}$

④  $S_n = \frac{a (r^n - 1)}{r - 1}$

⑤  $S_n = a \frac{(1 - r^n)}{1 - r}$

সুচনা শব্দ: প্রতি e হিসেবে মিলিষি করা হবে।

$$\begin{aligned} \textcircled{1} \quad e^x &= 1 + \frac{1}{1!} + \frac{x}{2!} + \frac{x^2}{3!} + \frac{x^3}{4!} + \dots + \frac{x^n}{n!} \\ \textcircled{2} \quad e^x &= 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots + \frac{x^n}{n!} \\ \textcircled{3} \quad e^x &= 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \dots + \frac{1}{n!} \\ \textcircled{4} \quad e^{-1} &= 1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \frac{1}{4!} - \dots + \frac{(-1)^n}{n!} \\ \textcircled{5} \quad \frac{1}{2}(e+e^{-1}) &= 1 + \frac{1}{2!} + \frac{1}{4!} + \frac{1}{6!} + \dots + \frac{1}{(2n)!} \\ \textcircled{6} \quad \frac{1}{2}(e-e^{-1}) &= \frac{1}{1!} + \frac{1}{3!} + \frac{1}{5!} + \frac{1}{7!} + \dots + \frac{1}{(2n-1)!} \\ \textcircled{7} \quad x < e < 3 \\ \textcircled{8} \quad \frac{1}{2}(e^x - e^{-x}) &= \frac{x}{1!} + \frac{x^3}{3!} + \frac{x^5}{5!} + \dots + \frac{x^{2n-1}}{(2n-1)!} \\ \textcircled{9} \quad \frac{1}{2}(e^x + \frac{1}{e^x}) &= 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \dots + \frac{x^{2n}}{(2n)!} \end{aligned}$$

\* প্রশ্ন মিলিষির  $T_n$  এর যার প্রয়োজন হলে? আঃ ৭০  $T_n$

ও  $n = 1, 2, 3, 4, \dots$  বিস্তৃত হিসেবে প্রয়োজন করলেই যেটি মিলিষির প্রয়োজন হিসেবে হবে।

$$\log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \frac{x^6}{6} + \dots - \infty$$

$$\log(1-x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \frac{x^4}{4} - \frac{x^5}{5} - \frac{x^6}{6} - \dots - \infty$$

$$\left( \frac{1}{b} \right)^{\%m} = (a \% m) \times (b^{-1}) \% m$$

$$\therefore b^{-1} = (b^{m-2}) \% m$$

$$= ((b \% m)^{m-2}) \% m$$

3.1.6 মডুলার ইনভার্স (Modular Inverse)

আগেই বলেছি অনেক হিসাবের উভয় অনেক বড় আসে সুতরাং প্রায় সময়ই আমাদের সেই বড় উভয়ের না চেয়ে কোনো একটি সংখ্যা দিয়ে mod করে উভয়ের চাওয়া হয়। এখন সেই হিসাবে যদি যোগ, বিয়োগ, গুণ থাকে তাহলে কোনো সমস্যা হয় না, কিন্তু যদি ভাগ থাকে তাহলে বেশ সমস্যা হয়ে যায়, কারণ  $\frac{a}{b} \bmod M$  এবং  $\frac{a \bmod M}{b \bmod M}$  একই কথা নয়। উদাহরণ তো খুব সহজ  $\frac{12}{3} \bmod 3 = \frac{12 \bmod 3}{3 \bmod 3} = \frac{0}{0} \equiv 0$  দিয়ে ভাগ? ভয়কর কথা! তুমি যদি  $b$  দিয়ে ভাগ করলে চাও তাহলে  $b^{-1} \bmod M$  রের করতে হবে এবং এটি দিয়ে গুণ করলেই  $b$  দিয়ে ভাগের কাজ হয়ে যাবে। আমরা কিছুক্ষণ আগেই শিখে এসেছি যে  $M$  যদি মৌলিক সংখ্যা হয় তাহলে  $b^{M-1} \equiv 1 \pmod M$  বা  $b^{-1} \equiv b^{M-2} \pmod M$  আর যদি মৌলিক সংখ্যা না হয় তাহলে  $b^{\phi(M)} \equiv 1 \pmod M$  বা  $b^{\phi(M)-1} \equiv b^{-1} \pmod M$ . তবে এই দুক্ষেত্রেই  $M$  এবং  $b$  কে সহমৌলিক হতে হবে। আর তোমরা তো জানোই কীভাবে  $b^{M-2} \pmod M$  বা  $b^{\phi(M)-1} \pmod M$  রের করতে পারব? BigMod!

যদি  $M$  আর  $b$  সহমৌলিক না হয়? তাহলে এর কোনো নিদিষ্ট ইনভার্স নেই। কেন? তার আগে একটা জিনিস, সেটা হলো  $b^{-1} \pmod M$  আসলে কি? এটা এমন একটি সংখ্যা যাকে  $b$  দিয়ে গুণ করলে কাটাকুটি শিয়ে 1 থাকবে। অর্থাৎ,  $b \times b^{-1} \pmod M \equiv 1$ . যেমন  $3^{-1} \equiv 3^5 \pmod 7$  বা  $243 \pmod 7$  বা 5 আর  $3 \times 5 = 15 \equiv 1 \pmod 7$ . কিন্তু আমরা যদি জানতে চাই  $4^{-1} = ?$  মড 6. ধরা যাবে এটি  $x$  তাহলে  $4x \equiv 1 \pmod 6$  হতে হবে। কিন্তু আমরা যদি একে একে  $x$  এর মান 0 হতে 5 পর্যন্ত চেষ্টা করি। কোনোটার জন্য কি  $4x \pmod 6$  কী 1 হয়? হয় না। এর মানে সমাধান নেই। এটা প্রমাণ করাও বেশ সহজ। মনে কর  $b^{-1} \equiv x \pmod M$  আর  $b$  ও  $M$  যেহেতু সহমৌলিক নয় তার মানে তাদের একটি সাধারণ গুণনীয়ক আছে, ধরা যাক  $p$ . আমরা লিখতে পারি  $bx \equiv 1 \pmod M$  অর্থাৎ  $bx \equiv 1 \pmod p$  কে  $M$  দিয়ে ভাগ করলে ভাগশেষ 1. ধরা যাবে ভাগফল হলো  $Q$ . তাহলে আমরা লিখতে পারি  $bx - QM = 1$ . কিন্তু দেখ বায় দিকের দুটি টার্ম  $bx$  ও  $QM$  কিন্তু  $p$  দ্বারা ভাগ যায়। কিন্তু ডানদিকের 1 কিন্তু ভাগ যায় না। অর্থাৎ এরকম  $x$  আসলে কখনই থাকা সম্ভব না।

### 3.1.7 Extended GCD

যদি  $a$  ও  $b$  এর গ.স.গ.  $g$  হয় তাহলে এমন  $x$  এবং  $y$  যে পাওয়া সম্ভব যেন  $ax + by = g$  হয়। টেবিল 3.2 এ আমরা  $a = 10$  এবং  $b = 6$  এর জন্য  $x$  ও  $y$  রের করে দেখালাম। অনেকেই এই টেবিল দেখে ভরকে যায়। আসলে তায় পাওয়ার কিছু নেই। এই টেবিলের প্রথম কলাম হলো সাধারণ ইউক্লিডের গ.স.গ. বের করার পদ্ধতি হতে পাওয়া। এখন গ.স.গ. বের করার সময় অবশ্যই ছেট সংখ্যাকে কোনো একটি সংখ্যা দিয়ে গুণ করে বড়ি হতে বাদ দিয়েই ভাগশেষ বের করা হয়। এই গুণ ও বিয়োগের কাজটা আমাদের পরের দুই কলামেও করতে হবে। এরকম করলে

<sup>1</sup> খেয়াল কর 6 চেষ্টা করা আর 0 চেষ্টা করা একই কথা, একই ভাবে 7 আর 1 একই কথা। তাই 0 হতে 5 চেষ্টা করা আর সব চেষ্টা করা একই কথা।

0, 1, 1, 2, 3, 5, 8, 13, 21

$F_0, F_1, F_2, F_3, F_4, F_5, F_6, F_7, F_8$

$$\begin{aligned} \left[ \begin{array}{c} F_2 \\ F_1 \end{array} \right] &= \left[ \begin{array}{cc} 1 & 1 \\ 1 & 0 \end{array} \right] \left[ \begin{array}{c} F_1 \\ F_0 \end{array} \right] = \left[ \begin{array}{cc} 1 & 1 \\ 1 & 0 \end{array} \right] \times \left[ \begin{array}{c} 1 \\ 1 \end{array} \right] = \left[ \begin{array}{c} 1+1 \\ 1+0 \end{array} \right] = \left[ \begin{array}{c} 2 \\ 1 \end{array} \right] \\ \left[ \begin{array}{c} F_3 \\ F_2 \end{array} \right] &= \left[ \begin{array}{cc} 1 & 1 \\ 1 & 0 \end{array} \right] \left[ \begin{array}{c} F_2 \\ F_1 \end{array} \right] = \left[ \begin{array}{cc} 1 & 1 \\ 1 & 0 \end{array} \right] \times \left[ \begin{array}{c} 2 \\ 1 \end{array} \right] = \left[ \begin{array}{c} 2+1 \\ 1+0 \end{array} \right] = \left[ \begin{array}{c} 3 \\ 1 \end{array} \right] \\ &\vdots \end{aligned}$$

একইভাবে,

$$\begin{aligned} \left[ \begin{array}{c} F_4 \\ F_3 \end{array} \right] &= \left[ \begin{array}{cc} 1 & 1 \\ 1 & 0 \end{array} \right]^3 \left[ \begin{array}{c} F_1 \\ F_0 \end{array} \right] = \left[ \begin{array}{cc} 1 & 1 \\ 1 & 0 \end{array} \right] \times \left[ \begin{array}{c} 2 \\ 1 \end{array} \right] = \left[ \begin{array}{c} 1+2+1 \\ 1+0 \end{array} \right] = \left[ \begin{array}{c} 3 \\ 1 \end{array} \right] \\ \therefore F_4 &= 3 \\ F_3 &= 2 \end{aligned}$$

$$\left[ \begin{array}{c} F_n \\ F_{n-1} \end{array} \right] = \left[ \begin{array}{cc} 1 & 1 \\ 1 & 0 \end{array} \right]^{n-1} \left[ \begin{array}{c} F_1 \\ F_0 \end{array} \right]$$

তোমরা যদি ভেবে থাক মাট্রিক্স এর ঘাত তুলতে গেলে তো খবর হয়ে যাবে, তাহলে ভুল ভাব। কিছুক্ষণ আগেই কিন্তু আমরা BigMod শিখে এসেছি, ওখানে ছিল একটি সংখ্যা, এখানে আছে মাট্রিক্স। সংখ্যা গুণ করার পরিবর্তে তুমি শুধু মাট্রিক্স গুণ করবে তাহলেই হবে। আমি তোমাদের পরামর্শ দিব তোমার নিজেরাই এই জিনিসটা কোড কর। প্রথম দিকে যদিও কোড করতে একটু সমস্যা হবে কিন্তু দুই একবার নিজে থেকে কোড করলে দেখবে অনেক সহজ হয়ে যাবে। স্টারলিং সংখ্যাও এই পদ্ধতিতে বের করা যায়, তবে সেক্ষেত্রে  $k \times k$  আকারের মাট্রিক্স লাগে। সুতরাং বুতেই পারে  $k$  ছেট আর  $n$  অনেক বড় হলে এই পদ্ধতিতে তোমরা স্টারলিং সংখ্যার মান বের করতে পারবে। চেষ্টা করে দেখতে পার সমাধান করতে পার কি না।

### 3.2.6 ইনক্লুশন এক্সক্লুশন নীতি (Inclusion Exclusion Principle)

গণনা বা counting এর ক্ষেত্রে অত্যন্ত গুরুত্বপূর্ণ একটি পদ্ধতি হলো ইনক্লুশন এক্সক্লুশন নীতি (Inclusion-Exclusion Principle), মনে কর তোমাকে বলা হলো, 1 হতে 100 পর্যন্ত কতগুলো গাউন্ডের প্রাপ্তির মূল্য নির্ণয় করার পদ্ধতি। এখানে আমরা ধৰণের পরিবর্তন করে নিতে কোণ 3.1 এ দেখানো হলো। এখানে আমরা ধরে নিয়েছি যে সমাধান না থাকে তাহলে কী করতে হবে তা নিজেরা পরিবর্তন করে নিতে পরে আশা করি। আর আমি এখানে অদলবদল করার কাজে swap ফাংশন ব্যবহার করেছি। এটা তোমরা stl এর algorithm হেডার ফাইলে পাবে।

#### কোড 3.11: gauss.cpp

```

1 for (i = 0; i < n; i++)
2 {
3     id = i;
4     for (j = i + 1; j < n; j++)
5         if (fabs(mat[j][i]) > fabs(mat[id][i]))
6             id = j;
7
8     if (id != i)
9     {
10        for (j = i; j <= n; j++)
11            swap(mat[i][j], mat[id][j]);
12    }
13
14
15    for (j = 0; j < n; j++)
16        if(j != i)
17        {
18            factor = mat[j][i] / mat[i][i];
19
20            for(k = i; k <= n; k++)
21                mat[j][k] -= factor * mat[i][k];
22        }
23    }
24 }
```

factor =  $\frac{2}{3} = \frac{1}{3}$

factor =  $\frac{1}{3} = \frac{1}{3}$

factor =  $\frac{1}{3} = \frac{1}{3}$

factor =  $\frac{1}{3} = \frac{1}{3}$

## ৫.১১.৮ Lazy without Propagation

মনে কর তোমাদেরকে বলা হলো যে  $n$  টি বাল্ক পর আছে এবং শুরুতে তারা সবাই বন্ধ এখন একটি অপারেশনে: হতে  $j$  পর্যন্ত সব বাল্ক toggle করতে বলা হতে পারে।<sup>১</sup> আবার তোমারে কখনো কখনো জিজ্ঞাসা করা হতে পারে যে  $q$  তম বাল্কটি চালু আছে, নাকি বন্ধ? তুমি এই সমস্যায় সমাধান সেগমেন্টটি ব্যবহার করে  $O(\log n)$  এ করতে পারবে। এক্ষেত্রে ধারনাটি হলো যখন তুমি<sup>২</sup> হতে  $j$  পর্যন্ত বাল্ককে toggle করবে তখন কিন্তু তুমি এই সীমার মধ্যে প্রতিটি বাল্ককে আপডেট করতে পারবে না, তোমাকে গুচ্ছ ধরে আপডেট করতে হবে। ধর তোমার কাছে  $n = 8$  টি বাল্ক আছে আর তোমাকে 1 হতে 4 পর্যন্ত বাল্ক আপডেট করতে বলা হলো। তুমি যা করবে তাহলে সেগমেন্টটি এর শুধু  $[1, 4]$  এর সেগমেন্টে লিখে রাখবে যে এই সীমার প্রতিটি বাল্ক toggle কর হয়েছে। চিত্ৰ ৫.১.২ এর সঙ্গে তুলনা করতে পার। যদি তোমাকে বলে 1 থেকে 3 পর্যন্ত toggle করতে হবে, তাহলে তুমি  $[1, 2]$  এবং  $[3, 3]$  এই সীমা দুটি আপডেট করবে। আপডেট এর সময় শুধু তুমি লিখে রাখবে যে এই সীমাটি করবার toggle হয়েছে। লাভ কী? ধর তোমাকে জিজ্ঞাস করুন 3 এর অবস্থা কী? তুমি যা করবে, কুট থেকে  $[3, 3]$  পর্যন্ত যাবে এবং শুধু দেখবে এটি যেই যেই সীমা দিয়ে যায় সেসব সীমা করবার করে toggle হয়েছে। এ থেকেই তুমি তোমার উভয় পেয়ে যাবে। তাহলে কুয়েরি যে মাত্র  $O(\log n)$  এ হচ্ছে তা তো খুব সহজেই বোৱা যায়, কিন্তু আপডেট? আমরা কিন্তু ইতোমধ্যেই সাবসেকশন ৫.১.৩ তে এরকম কিছু প্রমাণ করে এসেছি সুতরাং আমাদের আপডেটও  $O(\log n)$ । কোড ৫.৯ এ কুয়েরি ও আপডেটের কোড দেওয়া হলো। আমাদের এই সমাধানে আমরা যে একদম নিচ পর্যন্ত না গিয়ে উপরেই কিছু লিখে রেখে শেষ করে ফেনেছি আপডেটের কাজ, একেই lazy বলা হয়। আমরা এখানে lazy কে কিন্তু ডেজেনে নিচে নামাইনি, সে জন্য একে without propagation বলে। ডেজেনে নিচে নামানোর মানে কী তা পরবর্তী সাবসেকশনেই পরিকার হয়ে যাবে।

### কোড ৫.৯: lazyWithoutPropagation.cpp

```

1 void update(int at, int L, int R, int l, int r)
2 {
3     if(r < L || R < l) return;
4     if(l <= L && R <= r) {toggle[at] ^= 1; return;}
5
6     int mid = (L + R)/2;
7     update(at * 2, L, mid, l, r);
8     update(at * 2 + 1, mid + 1, R, l, r);
9 }
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
587
588
589
589
590
591
592
593
594
595
596
597
597
598
599
599
600
601
602
603
603
604
605
606
606
607
607
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378

```

```

28     update(at == 2, L, mid, 1, r);
29     update(at == 2 + 1, mid + 1, R, 1, r);
30
31     on[at] = on[at == 2] + on[at == 2 + 1];
32 }
33
34 int query(int at, int L, int R, int l, int r)
35 {
36     if(r < L || R < l) return;
37     if(l <= L && R <= r) return on[at];
38
39     if(toggle[at]) Propagate(at, L, R);
40
41     int mid = (L + R)/2;
42     int x = query(at == 2, L, mid, 1, r);
43     int y = query(at == 2 + 1, mid + 1, 1, r);
44
45     return x + y;
46 }

```

## ৫.১১.৬ একটি উদাহরণ

নিচেরের lazy এর ধরণা এই অধ্যায়ের সবচেয়ে কঠিন একটি ব্যাপার। একে বুঝিয়ে ওছিয়ে  
বলাও একটু কঠিন। সেজন্য আরও একটি উদাহরণ দিয়ে এটা বোানোর চেষ্টা করি। আমাদের  
সমস্যা হলো, একটি সংখ্যার অ্যারে আছে। কুয়েরি খুব সহজ,  $i$  হতে  $j$  ইনডেক্সের মাঝের সর্বনিম্ন  
পরিমাণ বাড়াতে হবে। কীভাবে করা যায়? প্রথমত আমাদের সেগমেন্ট ট্রি তে প্রতিটি নোডে একটি  
ভ্যারিয়েবল রাখতে হবে যাতে থাকবে ওই সীমার সর্বনিম্ন। তাহলে আমরা যখন কুয়েরি করব তখন  
সীমাগুলোতে যাব, সেখানে একটি lazy এর মতো থাকবে যা বলবে এই সীমার প্রতিটি সংখ্যাকে  
আরও কর্তৃ যোগ করা হয়েছে। আগের মতো থাকবে যা বলবে এই সীমার প্রতিটি সংখ্যাকে  
যোগ করা হয়েছে। এরপরে যখন আমরা এই A নোড হতে নামব তখন এই S কে তার চাইডের  
অতিরিক্ত পরিমাণ যোগ করা হলো। এর ফলে আমাদের দুদিকের সর্বনিম্ন মানও কিন্তু S করে বেড়ে  
যাবে। এবার আমরা যেই আপডেট করতে এসেছি সেটা নিচের দিকে যাবে। আপডেট করা শেষে

যখন করে আপন তখন আমার শোভের সর্বান্য মাশকে আপডেট করতে হবে। স্বাভাবিকভাবেই  
আপডেট করব অর্থাৎ, আমার সর্বনিম্ন মান হবে দুদিকের সর্বনিম্ন মানের মধ্যে যে কম। কারণ আমি  
যেই আপডেট করতে নিচে নেমেছি সেটা হ্যাতে কোনো একদিকে অনেক কম সংখ্যা তৈরি করে  
দিয়ে এসেছে। এভাবে lazy with propagation এর মাধ্যমে এই সমস্যা সমাধান করা যায়।

## ৫.১২ বাইনারি ইনডেক্সড ট্রি (Binary Indexed Tree) - BIT

সংক্ষেপে একে BIT বলা হয়। এটি সেগমেন্ট ট্রি এর মতোই একটি টেটো স্ট্রাকচার তবে এটি  
একটু জটিল, কিন্তু জমার ব্যাপার হলো এর কোড খুবই ছোট। তুমি পুরো টেটো স্ট্রাকচার না বুঝেও  
ব্যবহার করতে পারবে। সত্যি কথা বলতে আমি নিজেও এই টেটো স্ট্রাকচার খুব ভালো মতো বুঝি  
না। কিন্তু এটি ব্যবহার করতে আমার বেশি কষ্ট ও হয় না। এটি ঠিক যে, BIT দিয়ে তুমি যা যা করতে  
পারবে সেগমেন্ট ট্রি দিয়েও প্রায় সবই তুমি করতে পারবে, তবে সেগমেন্ট ট্রি দিয়ে এমন কিছু করা  
যায় যা আসলে তুমি BIT দিয়ে করতে পারে না। কিন্তু BIT এর সুবিধা হলো, এটি অনেক ছোট  
কোড, এর জন্য  $n$  আকারের মেমোরী লাগে এবং এটি অনেক স্ফূর্ত।<sup>১</sup>

কুব সংক্ষেপে বলতে হলে বলা যায়, BIT এ তোমার দুই ধরনের অপারেশন করতে পার।

১. কোনো একটি স্থান  $idx$  কে  $v$  পরিমাণ বৃক্ষি  $update(idx, v)$

২. কুব হতে  $idx$  পর্যন্ত যোগফল বের করা  $read(idx)$

আরও বেশ কিছু অপারেশন করা যায় যা আসলে বহুল ব্যবহৃত না। তোমরা topcoder এর  
টিউটোরিয়ালে দেখতে পার।

কোড ৫.১১ এ  $read$  এবং  $update$  দেখানো হল। এখানে  $MaxVal$  হলো  $n$  এর মান। অর্থাৎ  
তোমার অ্যারে যত বড় আর কী! আর BIT এ তোমাকে  $1 - indexing$  ব্যবহার করতে হবে।

### কোড ৫.১১: bit.cpp

```

1 int read(int idx)
2 {
3     int sum = 0;
4
5     while (idx > 0)
6     {
7         sum += tree[idx];
8     }
9 }

```

<sup>১</sup>তোমরা এর সম্পর্কে আরও বিস্তারিত জানতে চাইলে টপকোডারের এই <https://www.topcoder.com/community/data-science/data-science-tutorials/binary-indexed-trees/> আর্টিকেলটি পড়তে পার।

```

8     idx -= (idx & -idx);
9 }
10
11     return sum;
12 }
13
14 void update(int idx, int val)
15 {
16     while (idx <= MaxVal)
17     {
18         tree[idx] += val;
19         idx += (idx & -idx);
20     }
21 }

```

## ০-১ Knapsack Memory Optimized iterative P-165 solution O(n \* W) // n ≤ 500 & w ≤ 2000000

```

int i,j,n,w;
scanf("%d %d", &n, &w);
int wt[n+2], val[n+2];
for(i=0; i<n; i++)
    scanf("%d %d", &wt[i], &val[i]);
int dp[W+2];
memset(dp, 0, sizeof(dp));
for(i=0; i<n; i++)
{
    for(j=W; j>=wt[i]; j--)
    {
        dp[j] = max(dp[i], val[i]+dp[j]);
        dp[j] = max(dp[j], val[i]+dp[j-wt[i]]);
    }
}
printf("%d\n", dp[W]);
}

```

কোড ৮.৬: dijkstra.cpp

```

1 struct Node {
2     int at, cost;
3     Node(int at, int cost) {

```

১১০

```

8         at = .at;
9         cost = .cost;
10    }
11  };
12
13 bool operator<(Node A, Node B) {
14     // Priority queue returns the greatest value.
15     // So we need to write the comparator in a way
16     // so that cheapest value becomes greatest value.
17     return A.cost > B.cost;
18 }
19
20 struct Edge {
21     int v, w;
22 };
23
24 vector<Edge> adj[100]; // adjacency list of weighted edges.
25 priority_queue<Node> PQ;
26 int dist[100];
27 int n;
28
29 void dijkstra(int s) {
30     for (int i = 1; i <= n; i++) {
31         dist[i] = 1000000000;
32     }
33     dist[s] = 0;
34     PQ.push(Node(s, 0));
35
36     while (!PQ.empty()) {
37         Node u = PQ.top();
38         PQ.pop();
39
40         if (u.cost != dist[u.at]) {
41             continue;
42         }
43
44         for (Edge e : adj[u.at]) {
45             if (dist[e.v] > u.cost + e.w) {
46                 dist[e.v] = u.cost + e.w;
47                 PQ.push(Node(e.v, dist[e.v]));
48             }
49         }
50     }
51 }

```

কোড নং: bellmanford.cdp

```
1 struct Edge {
2     int u, v, w;
3 };
4
5
6 vector<Edge> E; // weighted edge list
7 int dist[100];
8 int n;
9
10 void bellman_ford(int s) {
11     for (int i = 1; i <= n; i++) {
12         dist[i] = 1000000000;
13     }
14     dist[s] = 0;
15
16     for (int i = 1; i < n; i++) {
17         for (Edge e : E) {
18             if (dist[e.v] > dist[e.u] + e.w) {
```

৮.৫ অন পোরাম বা ফ্লয়েড ওয়ার্শাল অ্যালগরিদম (Floyd Warshall Algorithm)

আমাদের ডায়াকট্রো'র আলগরিদমের কমপ্লেক্সিটি ছিল  $O(m \log n)$  এর মতো। আমরা যদি সব নেট পেয়ারেই ডায়াকট্রো'র আলগরিদম চালাতাম তাহলে অল পেয়ার শটেল্স্ট পথ বের করতে সময় লাগত প্রায়  $O(m \log n)$  এর মতো। যদি প্রাফটি খুব একটা ঘন স্মৃতিবিট ( $m \approx n^2$ ) না হয় তাহলে  $n$  স্মৃতিবিটার ডায়াকট্রো'র আলগরিদম চালানোই ভালো। সত্যি কথা বলতে যেখানে ফ্লোড ওয়ার্শাল আলগরিদম চালাতে হবে সেখানে বার বার ডায়াকট্রো'র আলগরিদম চালানো হয়ে যাবার কথা। তাহলে আমরা ফ্লোড ওয়ার্শাল আলগরিদম শিখব কেন? এর একমাত্র কারণ হলো এর কোন খুবই ছোট ও সহজ। মাত্র পাঁচ লাইন আর সেই পাঁচ লাইনের মধ্যে তিনটি *for* লুপ। এটি মনে রাখাও খুব সহজ। প্রথমেই আমরা আলগরিদমটি দেখে নেই কোড ৮.৮ এ:

### কোড ৮.৮: apsp.cpp

```

5   for(k = 1; k <= n; k++)
6     for(i = 1; i <= n; i++)
7       for(j = 1; j <= n; j++)
8         if(w[i][j] > w[i][k] + w[k][j])
9           w[i][j] = w[i][k] + w[k][j];

```

১৪.৩ দ্বিমাত্রিক (Two Dimensional বা 2D)

এবাব  $2D$  তে একই সমস্যা কীভাবে সমাধান করা যায় দেখা যাক। মনে কর  $n \times m$  আকারের একটি অ্যারে আছে, তোমাকে প্রতিবার কোনো একটি  $p \times q$  আকারের সাব-অ্যারের জন্য সর্বনিম্ন সংখ্যা বা সর্বোচ্চ সংখ্যা জিঞ্জুসা করা হবে। কীভাবে সমাধান করবে? শুধু একটা কঠিন না। তুমি আগে প্রত্যেক সারির জন্য আগের উপরে  $q$  আকারের সাব-অ্যারের সর্বনিম্ন সংখ্যা বা সর্বোচ্চ সংখ্যা বের করে ফেল। তাহলে তুমি একটি  $n \times (m - q + 1)$  আকারের একটি সাব-অ্যারে পাবে। এবাব প্রত্যেক কলামের জন্য  $p$  আকারের সাব-অ্যারের সর্বনিম্ন সংখ্যা বা সর্বোচ্চ সংখ্যা বের কর। তাহলে পাবে  $(n - p + 1) \times (m - q + 1)$  আকারের সাব-অ্যারে। এটিই তোমাকে শেষ উত্তর দিছে। অর্থাৎ এই পরিবর্তিত অ্যারের কোনো একটি অবহান  $(a, b)$  তোমাকে  $(a, b) - (a + p - 1, b + q - 1)$  সাব-অ্যারের সর্বোচ্চ সংখ্যা বা সর্বনিম্ন সংখ্যা দিবে।

#### ১৫. লেস্ট কমন আনসেস্টর (Least Common Ancestor)

মনে কর একটি ট্রি দেওয়া আছে। এখন দুটি নোড দিয়ে জিজ্ঞাসা করা হবে তাদের লীস্ট কর্ম আনসেন্টর (least common ancestor) কে? লীস্ট কর্ম আনসেন্টর হলো সবচেয়ে নিচের এমন একটি নোড যেটি কুয়েরির দুই নোডেরই আনসেন্টর (ancestor)। আমাদের বর্ণনার সুবিধার জন্য ধরে নেই এই কুয়েরির দুইটি নোড হলো x এবং y,  $O(n)$  এ সমাধান করা তে খুব সহজ কিন্তু আমরা চাই আরও দ্রুত এই কুয়েরির উভয় দিতে। আমরা এখনে কীভাবে  $O(n \log(n))$  এ প্রিপ্রসেসিং করে  $O(\log(n))$  এ এই কুয়েরির উভয় দেওয়া যায় তা দেখব।  
 ধরে নেই  $\lceil \log(n) \rceil = 17$  তাহলে আমরা  $parent[17 + 1][n]$  আকারের একটি আরেঁ নিব।  
 এখন প্রতিটি নোড i এর জন্য আমরা  $parent[0][i]$  এ i এর প্যারেন্ট (parent) রাখব।  $root$  ( $root$ ) এর ক্ষেত্রে আমরা চাইলে  $root$ কেই রাখতে পারি বা কোনো একটি sentinel যেমন -1  
 ব্যবহার করতে পারি। তবে আমার মতে  $root$ কে রাখাই ভালো, অর্থাৎ 0 যদি  $root$  হয় তাহলে  
 $parent[0][root] = 0 = root$ .

সবই তো বোঝা গেলো কিন্তু  $parent[j][i]$  এর মানে কি?  $parent[j][i] = 1$  হলে  $j$ -  
এর  $2^0$  তম প্যারেন্ট। যেমন  $j = 0$  তে আছে এর  $2^0 = 1$  তম প্যারেন্ট।  $j = 1$   
থাকে এর প্যারেন্টের প্যারেন্ট অর্থাৎ বলতে পারো দাদা বা প্রাচ্ছপ্যারেন্ট (grandparent)।

$j = 2$  তে থাকবে তার দাদার দাদা (গ্র্যান্ডপ্যারেটের গ্র্যান্ডপ্যারেট)। এরকম করে। এই  
জিনিস তৈরী করার উপায় হলো তুমি DFS কর। DFS এর সময় যেই নোডে আসবে সেই নোডে  
 $parent[0 \dots 17][at]$  পূরণ করতে হবে। এটি পূরণ করার উপায় হলো  $parent[j][at] = parent[j-1][parent[j-1][at]]$ , মনে at এর  $2^{j-1}$  তম প্যারেটের  $2^{j-1}$  তম প্যারেট।  
এভাবে তুমি সব নোডের জন্য parent এর অ্যারে  $O(n \log(n))$  সময়ে পূরণ করে ফেলতে  
পারবে। যেহেতু আমাদের টি তে  $n$  টি নোড আছে, আর আমরা 2 এর ঘাতে প্যারেট দেবিহ তাই  
কোনো নোড হতে রুটে যাবার জন্য  $\log n$  এর বেশি ধাপের দরকার নেই। উদাহরণ দেওয়া যাব,  
মনে কর  $n = 10$ , তাহলে  $\lceil \log(10) \rceil = 4$  (আশা করি বোৰা যাচ্ছে যে আমরা 2 ভিত্তিক  
লগারিদম (logarithm) নিয়েছি)। তাহলে যেকোনো নোড হতে  $j = 4$  তম প্যারেট নেওয়া মান  
 $2^4 = 16$  তম প্যারেটে যাওয়া। যেহেতু আমাদের নোডই আছে মাত্র 10 টি তাই  $parent[4+1][i]$   
আকারের অ্যারেই যথেষ্ট।

এবার আসা যাক কুরেরির সময় কী করতে হবে সেই বিষয়ে। প্রথম কাজ হলো  $x$  এবং  $y$  এর মধ্যে যেটি বেশি গভীরতা বা ডেপ্থ (depth) এ আছে তাকে  $x$  এ নাও। এখন  $x$  কে  $y$  এর গভীরতার অ্যানেস্টেরে আনো। আনার উপায় সহজ, তুমি প্রথমে  $parent[17][x]$  দেখ, যদি এটি  $y$  এর গভীরতার থেকে কম গভীরতায় হয় তাহলে এর পরের প্যারেন্ট অর্থাৎ  $parent[16][x]$  ঢেকে কর, না হলে  $x = parent[17][x]$  কর। এভাবে 17 থেকে 0 পর্যন্ত লুপ চালিয়ে এই কাজ করলে  $x$  এবং  $y$  একই গভীরতায় চলে আসবে এবং এ জন্য তোমার সময় লাগবে  $O(\log(n))$ . যদি  $x \neq y$  একই হয়ে যায় তাহলে তো এটিই সীন্ট করণ আনেস্টের। আর যদি নাহয় তাহলে আবর্ত 17 হতে 0 পর্যন্ত লুপ চালাও। ধরা যাক এটি  $i$  এর লুপ। এবার দেখ  $parent[i][x]$  আর  $parent[i][y]$  একই কিনা। যদি একই হয় তাহলে  $i$  এর লুপ *continue* কর আর যদি না হয় তাহলে  $x = parent[i][x]$  এবং  $y = parent[i][y]$  কর। এভাবে 0 পর্যন্ত লুপ চালানোর পর, উত্তর হবে  $x \neq y$  এর সরাসরি প্যারেন্ট বা  $parent[0][x]$ . কেন? কারণ  $2^0 + 2^1 + \dots + 2^{n-1} = 2^n - 1$ . উত্তরটা একটু আবর্হ হয়ে গেলো কিন্তু আশা করি তোমরা চিন্তা করে বের করে ফেলতে পারবে।

### ১০.৩.১ কনভেক্স হাল (Convex Hull)

তোমাকে  $2D$  ছানাক্ষ ব্যবহায়  $n$  টি বিন্দু দেওয়া আছে, তোমাকে এর কনভেক্স হাল (convex hull) বের করতে হবে। কনভেক্স হাল কী? প্রদত্ত বিন্দুগুলোকে ঘেরাও করে সবচেয়ে ছোট যে কনভেক্স বহুভুজ (convex polygon) বানানো যায় তাই এই সব বিন্দুর কনভেক্স হাল। কনভেক্স বহুভুজ হলো এমন একটি বহুভুজ যার প্রতিটি কোণ  $180^{\circ}$  এর সমান বা ছোট। যদি কনভেক্স হালের কোনো তিনিটি বিন্দুকে একই রেখার উপর না দেখতে চাও তাহলে সবসময়  $180^{\circ}$  এর ছোট নিতে পার। একে এভাবে চিন্তা করতে পার- প্রদত্ত  $n$  বিন্দুতে তুমি পেরেক পুঁতো, এর পর একটি রাবার

গান্ধীকে প্রসারিত করে সব পেরেককে পোচায় দাও, তাহলে দেখবে তোমার রাবার খুব দৃঢ়ভাবে লেঁয়ে গুলো দিয়ে যায় এবং একটি কনভেক্স বহুভুজের রূপ নেয়। এটিই কনভেক্স হাল। প্রদত্ত বিন্দুগুলোকে ঘেরাও করে যেসব কনভেক্স বহুভুজ আকা যায় তাদের মধ্যে সবচেয়ে ছোট ক্ষেত্রফল এবং পরিসীমা এই কনভেক্স হালের।

এখন প্রশ্ন হলো আমরা কীভাবে কনভেক্স হাল বের করতে পারি? কনভেক্স হাল বের করার জন্য অনেকগুলো আলগরিদম আছে। সবচেয়ে জনপ্রিয় হলো গ্রাহামের স্ক্যান (Graham's scan). এখন আমাদের দেওয়া বিন্দুগুলোর মধ্যে সবচেয়ে কম  $y$  ওয়ালা বিন্দু বের করতে হবে, যদি একম অনেকগুলো থাকে তাহলে তাদের মধ্যে সবচেয়ে কম  $x$  ওয়ালা বিন্দু নেব। ধরা যাক এটি হলো  $O$ . এখন এই বিন্দুকে কেন্দ্র করে অন্যান্য বিন্দুগুলোকে আমরা CCW এ সর্ট করব। এক্ষেত্রে দুটি জিনিস খেয়াল রাখতে পার, এক: তোমরা চাইলে পরবর্তী হিসাবনিকাশগুলো সহজে করার জন্য  $O$  কে মূল বিন্দুতে ট্রান্সলেশন করে নিতে পার, দুই: তুমি সর্ট করার আগেই atan2 ব্যবহার কর বিন্দুর  $O$  এর সাপেক্ষে কোণ বের করে নিতে পার, বার বার comparison ফাংশনের তের এই কোণ না বের করে আগে থেকে বের করে রাখলে সময় কম লাগে। তবে আমরা চাইলে atan2 ব্যবহার না করেও সর্ট করতে পারি। ধরা যাক comparison ফাংশনে  $A$  ও  $B$  দুটি বিন্দু নিয়ে বলা হলো কোনটি আগে হবে? যদি  $OAB$  CCW হয় তাহলে  $A$  আগে হবে না হলে পরে। আরেকটা কথা, যদি দুটি বিন্দু  $O$  এর সাপেক্ষে একই কোণ তৈরি করে তাহলে যার দূরত্ব  $O$  হতে যাম সে আগে থাকবে। এখন এই সর্টেড বিন্দুগুলোকে একে একে নিতে হবে আর শীর্ষসাপেক্ষে একটি স্ট্যাকে রাখতে হবে। যদি স্ট্যাকের ফাঁকা হয় তাহলে সরাসরি স্ট্যাকে চুকিয়ে দাও আর যদি তান হয় তবে  $O$ , স্ট্যাকের সবচেয়ে উপরে থাকা বিন্দু আর বর্তমান বিন্দু এদের যাচাই করে দেখতে হবে যে এরা কি CCW নাকি CCW. যদি CCW হয় তাহলে স্ট্যাকের উপরের বিন্দুকে ধরে ফেলে নিতে হবে এবং এবাবের স্ট্যাকের উপরের বিন্দুকে নিয়ে আবার একইভাবে যাচাই করতে হবে। দিতে হবে এবং এবাবের স্ট্যাকের উপরের বিন্দুকে স্ট্যাকে চুকিয়ে নিতে হবে। এভাবে একে সব বিন্দু যাচাই করা শেষ হয়ে গেলে স্ট্যাকে কনভেক্স হাল পাওয়া যাবে।

যদিও গ্রাহামের স্ক্যান আলগরিদম বেশ সহজ এবং জনপ্রিয় কিন্তু এটি সংখ্যাগত হিসাবনিকাশের দিক থেকে অতটা সুস্থিত না। অর্থাৎ তোমার ছানাক্ষ যদি ফ্রেটিং পয়েন্ট সংখ্যায় থাকে তাহলে মাঝে মধ্যে আমেলা পাকাতে পারে ফ্রেটিং পয়েন্ট সংখ্যার হিসাবনিকাশের থাকে তাহলে মাঝে আমেলা পাকাতে পারে ফ্রেটিং পয়েন্ট সংখ্যায় অস্থিতিশীলতার জন্য। (সেজল্য যেই আলগরিদম ব্যবহার করা উচিত তা হলো মনোটেন চেইন অস্থিতিশীলতার জন্য।) সেজল্য যেই আলগরিদম ব্যবহার করা উচিত তা হলো মনোটেন চেইন অস্থিতিশীলতার জন্য। এটি একটি স্ট্যাকের উপরের সমান হলো  $y$  অনুযায়ী। এবাবে আগের মতো সর্ট করতে হবে অর্থাৎ প্রথমে  $x$  অনুযায়ী সহজ। প্রথমে আমাদের বিন্দুগুলোকে ছানাক্ষ অনুসারে সর্ট করতে হবে একে একে নিব। এবং তাদের  $x$  সমান হলো  $y$  অনুযায়ী। এবাবে আগের মতো সর্ট করা বিন্দুগুলোকে একে একে নিব। স্ট্যাকের উপরের 2 টি বিন্দু এবং বর্তমান বিন্দু নিয়ে যাচাই করে যদি দেখি CCW তাহলে স্ট্যাকের উপরের বিন্দুকে ফেলে দিব (এখন কিন্তু আমরা বাম দিক থেকে ডান দিকে যাচাই এবং আমরা শুধু উপরের বিন্দুকে ফেলে দিব (এখন কিন্তু আমরা বাম দিক থেকে ডান দিকে যাচাই এবং আমরা শুধু উপরের হাল বামচিছি)। এই প্রক্রিয়া শেষ হয়ে গেলে আমরা সর্টেড লিস্টের শেষ থেকে শুরুর দিকে উপরের হাল বামচিছি। এই প্রক্রিয়া শেষ হয়ে গেলে আমরা সর্টেড লিস্টের শেষ থেকে শুরুর দিকে CCW এ থাকে তাহলে যাব এবং আগের মতো যদি স্ট্যাকের উপরের দুটি বিন্দুর সঙ্গে বর্তমান বিন্দু CCW এ থাকে তাহলে যাব এবং আগের মতো যদি স্ট্যাকের উপরের দুটি বিন্দুর সঙ্গে বর্তমান বিন্দু CCW এ থাকে তাহলে যাব এবং আগের মতো যদি স্ট্যাকের উপরের দুটি বিন্দুকে ফেলে দিব। এভাবে একবাবার বাম থেকে ডানে এবং আরেকবাবার ডান থেকে স্ট্যাকের উপরের বিন্দুটি ফেলে দিব। এভাবে একবাবার বাম থেকে ডানে এবং আরেকবাবার ডান থেকে বাম গেলে আমরা উপরের হাল এবং নিচের হাল তৈরি করে ফেলতে পারব। এই আলগরিদম

আগের থেকে ছিত্তিশীল কারণ এখানে কোণ অনুসারে সর্ট করার কোনো ব্যাপার নেই। তোমার চাইলে উইকিপিডিয়াতে দেওয়া ইমপ্লিমেন্টেশন (implementation)<sup>১</sup> টি দেখে নিতে পার।

### ১০.৩.২ নিকটতম বিন্দুজোড় (Closest pair of points)

২ $D$  ছানাক্ষ ব্যবহায়  $n$  টি বিন্দুর ছানাক্ষ দেওয়া আছে। তোমাকে এদের মধ্যের দূরত্বের দিক থেকে নিকটতম বিন্দুজোড় বের করতে হবে অর্থাৎ যে দুটি বিন্দুর মধ্যের দূরত্ব সবচেয়ে কম সেই দুটি বিন্দু বের করতে হবে বা সেই দূরত্ব মাপতে আমরা ইউক্লিডীয় দূরত্ব (euclidean distance) ব্যবহার করব। দুটি বিন্দুর ছানাক্ষ যদি  $(x_1, y_1)$  এবং  $(x_2, y_2)$  হয় তাহলে তাদের মধ্যের ইউক্লিডীয় দূরত্ব হবে  $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ . একে সরলরোধিত দূরত্ব (straight line distance) ও বলা যায়। এই সমস্যা সমাধানের জন্য আমাদের প্রথমে যা করতে হবে তা হলো বিন্দুগুলোকে  $x$  অনুযায়ী সর্ট করতে হবে (ছোট থেকে বড়)। এরপর আমরা এই বিন্দুগুলোকে দুই ভাগে ভাগ করব, এক ভাগে (বাম ভাগে) থাকবে ছোট  $x$  আরেক ভাগে বড়গুলো। মেট্রিটি সমান দুই ভাগে ভাগ করতে হবে। অর্থাৎ বিজেডের ক্ষেত্রে একটি রেসিপি থাকতে পারে আর কি! এখন তোমাদের রিকার্সিভ (recursive) উপায়ে দুই দিকের জন্য নিকটতম জোড় বের করার ফাংশন কল করতে হবে। closestpair ফাংশন দুটি জিনিস দেবে- এক: তার পাওয়া বিন্দুগুলোর মধ্যে নিকটতম বিন্দুজোড়ের দূরত্ব এবং দুই: তার পাওয়া বিন্দুগুলোর  $y$  অনুযায়ী সর্টেড লিস্ট (বড় থেকে ছোট)। যদি তোমার ফাংশন মাত্র একটি বিন্দু পায় (base কেইস) তাহলে ধরা যাক সে  $\infty$  বলবে নিকটতম বিন্দুজোড়ের দূরত্ব হিসেবে আর একটি কুণ্ডল জোড় করব। দুই লিস্টের মাথা দেখবে এবং এখন যদি একের বেশি বিন্দু হয় তাহলে তো আমরা দুই ভাগে ভাগ করে রিকার্সিভ কল করেছিলাম। ধরা যাক আমরা দুই দিক থেকে নিকটতম বিন্দুজোড়ের দূরত্ব পেয়েছি  $d_1$  এবং  $d_2$ , আমরা  $d = \min(d_1, d_2)$  নিয়েই শুধু আগ্রহী। আরও মনে করা যাক, দুই দিকের  $y$  অনুসারে সর্টেড লিস্ট হলো  $P_1$  এবং  $P_2$ . এখন আশা করি বুবাতে পারচ কীভাবে এদের মিলিত য অনুসারে সর্ট করা লিস্ট পাওয়া যাবে? ঠিক মার্জ সর্ট (merge sort) এর মতো। দুই লিস্টের মাথা দেখবে এবং এখান য বেশি তাকে নিবে এভাবে চলতে থাকবে (আমরা কিন্তু বড় থেকে ছোট সর্ট করছি, যদিও দেখেকোনো এক ভাবে করলেই হলো)। এখন মনে কর বামের ভাবের সবচেয়ে বড়  $x$  হলো  $x_{divider}$ . এটি রিকার্সিভ কল করার আগে  $x$  অনুযায়ী সর্টেড লিস্ট হতে বের করে একটি লোকাল ভ্যারিয়েবল (local variable) এ রাখতে পারে। খেয়াল কর, রিকার্সিভ কল করার পর কিন্তু সেই লিস্ট  $y$  অনুযায়ী সর্ট হয়ে যাবে। সুতরাং আমাদের আগেই এই  $x_{divider}$  বের করে রাখতে হবে (অথবা আরও অনেক কৌশল খাটিনো যায় যা আমরা পরে সংক্ষেপে বলব)। রিকার্সিভ কল শেষে পাওয়া  $y$  অনুযায়ী সর্ট করা বিন্দুগুলোকে ব্যবহার করে এদের মধ্যের নিকটতম বিন্দুজোড়ের দূরত্ব বের করতে পারি। প্রথমে খেয়াল কর, কোনো বিন্দু  $x$  যদি  $x_{divider} - d$  এর থেকে ছোট হয় বা  $x_{divider} + d$  এর থেকে বড় হয় তাহলে সেই বিন্দুকে আমরা বিবেচনা না করলেও পারি (তবে সর্টেড লিস্ট পাওয়ার জন্য আমাদের সব বিন্দুই বিবেচনা করতে হবে)। কারণ আমরা এখন শুধু মাত্র এমন জোড়ের প্রতি

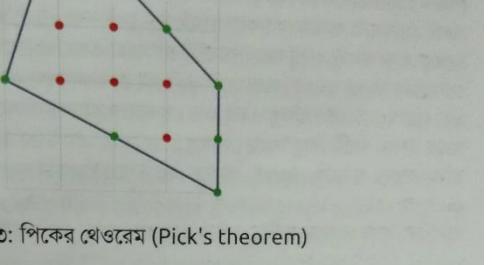
<sup>১</sup>[https://en.wikibooks.org/wiki/Algorithm\\_Implementation/Geometry/Convex\\_hull/Monotone\\_chain](https://en.wikibooks.org/wiki/Algorithm_Implementation/Geometry/Convex_hull/Monotone_chain)

অঞ্চলীয় যাদের একটি বিন্দু বাম ভাগে আরেক বিন্দু ডান ভাগে থাকে। বাম ভাগেই যদি দুটি বিন্দু থাকে তাহলে সেটা তো আমরা রিকার্সিভ উপায়ে বের করেই ফেলেছি। এখন মনে কর  $P_1$  থেকে আমরা যাদের বিবেচনা করব তারা হলো  $P'_1$  এবং একইভাবে  $P_2$  হতে  $P'_2$  এবং এরা  $y$  অন্যান্য স্টোর্ট। এখন আমাদের দুটি পয়েন্টার (pointer) লাগাবে যা দুই লিস্টের দুটি বিন্দুকে নির্দেশ করবে। শুরুতে এরা লিস্টের শুরুর উপাদানগুলোকে পয়েন্ট করবে। এখন দেখ যদি  $P_1$  লিস্টের পয়েন্টকৃত বিন্দুর  $y$  যদি  $P'_1$  এর পয়েন্টকৃত বিন্দুর  $y$  থেকে বেশি হয় তাহলে আমরা  $P'_1$  এর পয়েন্টকৃত বিন্দুকে প্রসেসিং করব। প্রসেসিং করা শেষে এই পয়েন্টারকে বাড়িয়ে দেব। কথা হলো কীভাবে প্রসেসিং করব? ধরা যাক  $P'_2$  এর পয়েন্টকৃত বিন্দুর  $y$  হলো  $y_1$ . তাহলে আমাদের  $P'_1$  এর সেসব বিন্দুর সঙ্গে তুলনা করতে হবে যাদের  $y$   $[y_1 - d, y_1 + d]$  এর মধ্যে থাকবে। প্রমাণ করা যায় যে এরকম বিন্দুর সংখ্যা আসলে 6 টির বেশি হতে পারে না। সুতরাং এভাবে  $O(n)$  সময়ে তৈরো  $P_1$  আর  $P_2$  এর মধ্যের নিকটতম বিন্দুজোড় বের করে ফেলতে পারবে। সুতরাং আমাদের পুরো আলগরিদমের কমপ্লেক্সিটি হবে  $O(n \log n)$ ।

তোমারা  $T$  হলে  $S \sqsubset T$  এর সাহায্যে আরও সহজে (এবং শব্দে) স্মারণ করো। তুমি  $x$  অনুযায়ী স্টেটে বিন্দুগুলো  $x$  অনুযায়ী সাজানো থাকবে অপরটি  $y$  অনুযায়ী। সেট (set) এর প্রয়োজন, একটি সেটে বিন্দুগুলো  $x$  অনুযায়ী সাজানো থাকবে অপরটি  $y$  অনুযায়ী। আমরা তাদের নাম দেই যথাক্রমে  $X$  এবং  $Y$ । আর এখন পর্যন্ত প্রাণ্য নিকটতম বিন্দুজোড়ের দূরত্ব ধরা যাক  $d$  তে থাকবে যাতে শুরুতে  $\infty$  মান থাকবে। এখন প্রথমে আমাদের দুটি সেটই ফোক এবং ধরা যাক  $d$  তে থাকবে যাতে শুরুতে  $\infty$  মান থাকবে। এখন প্রথমে আমাদের দুটি সেটই ফোক এবং ধরা যাক  $d$  তে থাকবে যাতে শুরুতে  $\infty$  মান থাকবে। এখন আমরা এই লিস্ট থেকে একে একে বিন্দুগুলোকে নির্ব। ধরা যাক এই বিন্দুটি হলো  $(x, y)$ , তাহলে আমাদের প্রথমে  $X$  এ দেখতে হবে  $x - d$  এর থেকেও ছোট  $x$  ওয়ালা কোনো বিন্দু আছে কিনা থাকলে তাকে  $X$  এবং  $Y$  উভয় থেকেই মুছতে হবে। এরকম সব বিন্দু মোছা হয়ে গেলে আমাদের  $Y$  নিয়ে কাজ করতে হবে। আমরা  $Y$  এ lower bound ব্যবহার করে  $y - d$  থেকে  $y + d$  এর ডেতেরে  $y$  এর মান ওয়ালা যত বিন্দু  $Y$  এ আছে তাদের সঙ্গে বর্তমান বিন্দুর দূরত্ব বের করে নিকটতম বিন্দুজোড়ের দূরত্ব  $d$  কে আপডেট করব এবং সেই সঙ্গে  $X$  এ  $Y$  এ বর্তমান বিন্দু ছুঁকিয়ে দেব। সত্যি কথা বলতে আমাদের  $X$  সেটের দরকার নেই, তোমারা  $x$  অনুযায়ী স্টেটে অ্যারেতে দুইটি হাত রেখেই এই কাজ করতে পার।

ধৰা যাব কোনোদের অনেকগুলো রেখাখণ্ড (line segment) দেওয়া আছে, বলতে হ

মধ্যে কতগুলো ছেদ বিন্দু আছে, অর্থাৎ কতগুলো রেখাখণ্ডগুলি পয়েন্ট সংখ্যার হিসাবনিকাশ বালেনো না পাকাতে পারে সেজন্য অনেক সময় বলা হয় দুইয়ের মেঘে রেখাখণ্ড পরস্পরকে একই বিন্দুতে ছেদ করে না। এই সমস্যা সমাধানের জন্য আমাদের একটি ব্যালেন্সড বাইনারি সার্চ ট্ৰি (balanced binary search tree) এবং একটি প্রায়োগিক কিউ



তার ঠিক আগে এবং পরের বেখাঁশগুলো বের করে ফেললে

গুণাত্মক (Dick's theorem)

পিকের থেওরেম (Pick's theorem) বলে 2D ছানাক এবং আঁকি অর্থাৎ এমন একটি বহুভুজ

সঙ্গে দেছ করে না বা স্পর্শণও করে না (পশ্চাত্যাশন দ্রুত বাই তামার প্রতিক্রিয়া হবে এবং পুরুষ সংখ্যাবিশিষ্ট ছানাকে থাকতে তাহলে  $A = i + b - 1$  হবে (বহুভুজের সব শৈর্ষ বিন্দুকে পূর্ণ সংখ্যাবিশিষ্ট ছানাকে থাকতে হবে)। এখানে  $A$  হলো বহুভুজের ফ্রেক্রফল,  $i$  হলো বহুভুজের অভ্যন্তরে(internal) কতগুলো পূর্ণ সংখ্যাবিশিষ্ট ছানাক আছে এবং  $b$  হলো বহুভুজের পরিসীমার উপরে কতগুলো পূর্ণ সংখ্যাবিশিষ্ট ছানাক আছে। যেমন চিত্র ১০.৩ এ লাল বিন্দুগুলো হলো অভ্যন্তরের বিন্দু  $i = 7$ , সবুজ বিন্দুগুলো ছানাক আছে। যেমন চিত্র ১০.৩ এ পুরু বিন্দুগুলো হলো বহুভুজের ফ্রেক্রফল  $A = 10$ . এখানে বলে রাখা যায় পরিসীমার উপরের বিন্দু  $b = 8$  এবং পুরু বিন্দুগুলো হলো বহুভুজের ফ্রেক্রফল বের করা কিন্তু খুব একটা কঠিন না, তুমি কল্পনা কর  $y = 2$  যে, চিত্রের বহুভুজের জন্য ক্ষেত্রফল বের করা কিন্তু খুব একটা কঠিন নিচে একটি বিভুজ। আর বিভুজের ফ্রেক্রফল তো বের করা এই রেখার উপরে একটি বিভুজ এবং নিচে একটি বিভুজ। আর বিভুজের ফ্রেক্রফল তো বের করা এই রেখার উপরে একটি বিভুজ এবং  $b$  এর মান জানি, পিকের সুত্রে বসিয়ে দেখি

বুরুই সোজা। যাই হোক,  $10 = 7 + \frac{8}{5} - 1$  একদম সঠিক।  
 এটি কাজ করে কিনা?  $10 = 7 + \frac{8}{5} - 1$  একদম সঠিক।  
 এখন একটি সমস্যা দেখা যাক। মনে কর তোমারের 2D স্থানাঙ্ক ব্যবস্থায় একটি ত্রিভুজ দেওয়া  
 আছে। এই ত্রিভুজের উপর (বাহুর উপর) এবং এই ত্রিভুজের অভ্যন্তরে কতগুলো বিন্দু (পূর্ণসাধারণ্যক  
 আছে। কিন্তু বা পূর্ণ স্থানাঙ্কিত স্থানাঙ্ক বা ল্যাটিসে (lattice point) আছে তা বের করতে হবে।  
 বিন্দুর পূর্ণ স্থানাঙ্কিত স্থানাঙ্ক বা ল্যাটিসে (lattice point) আছে। সূত্রের দিকে তাকালে আমরা বুঝতে পারব যে শুধু মাত্র এই  
 কিন্তু পিকের খেঙেরেমের গুঁড় আছে। সূত্রের দিকে তাকালে আমরা বুঝতে পারব যে শুধু মাত্র এই  
 ত্রিভুজের খেঙেরেল আমাদের জানা। আমরা যদি কোনোভাবে ভেতরে কয়টি বিন্দু আছে সেটি বা

বাহুগুলোর উপরে কয়টি বিন্দু আছে সেটি বের করতে পারি তাহলে অপরাটিও বের হবে যাবে। বাহুগুলোর উপরে কয়টি বিন্দু আছে তা বের করা সহজ। আমরা যদি একটি বাহুর উপরে কয়টি বিন্দু আছে তা বের করতে পারি তাহলে তিভুজের টিনিটি বাহুর উপরে কয়টি বিন্দু আছে তা বের করে ফেলতে পারব। ধরা যাক আমরা বের করতে চাই  $(x_1, y_1) - (x_2, y_2)$  এই বাহুর উপরে কয়টি বিন্দু আছে। আমরা এখন একটু একটু করে সমস্যাটিকে সহজ করব, প্রথমে দেখ  $(x_1, y_1) - (x_2, y_2)$  রেখাখণ্ড দেখা আর  $(0, 0) - (x_1 - x_2, y_1 - y_2)$  এই রেখাখণ্ড দেখা একই কথা। আবার  $(0, 0) - (|x_1 - x_2|, |y_1 - y_2|)$  দেখাও কিন্তু একই কথা। সুতরাং আমাদের আসলে বের করতে হবে  $(0, 0) - (x, y)$  বাহুর উপরে কয়টি বিন্দু আছ যেখানে  $x, y \geq 0$ . এর উন্নর হলো  $\gcd(x, y) + 1$ . কেন? ধর  $g$  হলো তাদের গ.স.ও. ( $\gcd$ ), তাহলে এর উপরে থাকা বিন্দুগুলো হলো:  $(x = g \times x/g, y = g \times y/g), ((g-1) \times x/g, (g-1) \times y/g), \dots (0 \times x/g, 0 \times y/g)$ . এভাবে আমরা  $b$  এর মান বের করে ফেলতে পারি।  $A$  আর  $b$  জানলে তো  $i$  বের করাই যায়।

୧୦.୭.୯ ପରିଷ୍କାର କାର୍ଯ୍ୟ ପାଇଁ

একটি বহুভুজের প্রত্যেক ত্বরণ পরিসর  $\pi$  এর মধ্যে অবস্থিত। আমরা জ্ঞান করতে পারে যে  $P_0P_1P_2, \Delta P_0P_1P_2, \dots, \Delta P_0P_{n-2}P_{n-1}$  এই বিশুঙ্গলোর ক্ষেত্রফল যাগ করতে হবে। তো পুরো বহুভুজের ক্ষেত্রফল বের হয়ে যাবে। না তা ঠিক না। এটি ঠিক হবে ক্ষেত্রের বহুভুজের জন্য। ক্ষেত্রের বহুভুজ (Concave polygon) এর জন্য এটি সত্য না ও হতে পারে। একটি বহুভুজের কোনো কোণটি যদি  $180^\circ$  এর চেয়ে বড় না হয় তাহলে তাকে বন্দেরের বহুভুজ বলে। আর যদি কোনো একটি  $180^\circ$  এর থেকে বড় হয় তাহলে তাকে ক্ষেত্রের বহুভুজ বলে। তাহলে আমরা কীভাবে বের করতে পারি? উভয় হলো চিহ্নিত ক্ষেত্রফল (signed area) ব্যবহার করে। চিহ্নিত ক্ষেত্রফল (signed area) কী? আমরা এই অধ্যায়ের শুরুর দিকে বিশুঙ্গের ক্ষেত্রফল বের করার জন্য নির্ণয়িক ব্যবহার করেছিলাম। সেভাবে বের করার জন্য বলেছিলাম যে পরম মান নিতে। কিন্তু তুমি যদি পরম মান না নাও এবং উপরের মতো  $\Delta P_0P_1P_2, \Delta P_0P_2P_3, \dots, \Delta P_0P_{n-2}P_{n-1}$  এর এই চিহ্নিত মানগুলো যোগ কর তাহলেই আমরা পুরো বহুভুজের চিহ্নিত ক্ষেত্রফল পেয়ে যাবে। অর্থাৎ এই চিহ্নিত মানগুলো যোগ করে যদি তুমি পরম মান নাও তাহলে তুমি ক্ষেত্রফল পাবে। তুমি চাইলে এই কাজ  $P_0$  কে কেন্দ্র করে মাঝে মূলবিন্দুকে কেন্দ্র করেও করতে পার। আমি এভাবেই করি, এক্ষেত্রে স্থূল খুচ ছোট হয়ে যায়।  $\sum (x_i y_{i+1} - y_i x_{i+1})$  (যি এর মান  $n - 1$  হলে  $i + 1 = 0$  ধরতে হবে কিন্তু)। এটি আমার জন্য নেই। রাখা আশেক সহজ। অন্যান্য অনেক কাজেই এই চিহ্নিত মান ব্যবহার করে অনেক সহজে সমাধান করা যায়।

বাম পাশের ক্ষেত্রে বাম আচ ফন্ডেটের বহুভুজের ক্ষেত্রে হয় তাহলে কিন্তু যাগামী  
বশ সহজ। মনে কর তোমাদের বিস্তৃত হলো  $(x', y')$  তাহলে  $y = y'$  এ বহুভুজের দুটি ছেবনিদু  
র করতে হবে। যদি তোমার  $x'$  এই দুই ছেবনিদুর  $x$  এর মধ্যে থাকে তাহলে বলা যাবে যে

আমাদের বিন্দু বহুভুজের ডেতের আছে। বহুভুজের সঙ্গে কোনো একটি  $y = y'$  রেখার ছেবিন্দু  
বের করা কিন্তু কঠিন কিছু না। তুমি সব বাহু দেখবে এবং তাদের জন্য সমাধান করবে, যদি কোনো  
বীর্ম বিন্দুতে হেড করে তাহলে একটু সাবধান হতে হবে আর কি! যাই হোক, আমাদের এই পদ্ধতি  
কিন্তু কনকেইভ বহুভুজে কাজ করবে না। কনভেক্ট্র বহুভুজের ক্ষেত্রে আরও একটি সহজ উপায়  
আছে। সেটি হলো মনে কর তোমারা জানতে চাইছ P বিন্দু কি বহুভুজের ডেতেরে কি না। তোমার  
যা করবে তা হলো  $PP_0P_1, PP_1P_2$  এরকম করে সব ভিন্নভোজের ক্ষেত্রফল বের করে যোগ করে  
(চিহ্নযুক্ত না)। এখন দেখবে এই যোগফল কি বহুভুজের ক্ষেত্রফলের সমান কি না। সমান হচ্ছে  
তেরে, আর সমান না হলে বাইরে।

কোনো শীর্ষ দিয়ে যেন না যায়। তুম এই কাজ খুব সহজে একটি দৈর (random) দিক নির্বাচ করে করতে পার। যদি সেই দিকে কোনো শীর্ষ বিন্দু থাকে তাহলে আরেকটি দৈর দিক নিবে, এভাবে চলতে থাকবে। আসলে দুই একবারের বেশি লাগার কথা না। এখন তোমাকে দেখতে হবে যে এই রশ্মি তোমার বহুভুজকে কয় জায়গায় ছেদ করে। যদি এই সংখ্যা বিজোড় হয় তার মানে তুম বহুভুজের ভেতরে আছ আর যদি জোড় সংখ্যক হয় তার মানে তুম বাইরে আছ। এর খেকেও সহজে উপর আছে সমাধান করার। সেটি হলো ওয়াইন্ডিং সংখ্যা (winding number), তোমার বিন্দু চারিদিকে বহুভুজ কয় পাক মারে সেটিই হলো ওয়াইন্ডিং সংখ্যা (winding number), তোমার বিন্দু সাপেক্ষে সবগুলো বাহুর জন্য চিহ্নিত কোণের যোগফল বের করালেই তুমি সমাধান করে দেলতে পারবে। কিন্তু এটি একটু ঝামেলা মনে হয় আমার কাছে, আর তাছাড়া দরকার না হলো আমি ফ্রোটিং পয়েন্ট সংখ্যার হিসাবনিকাশ হতে বিরত থাকি। সেজন্য আমি আসলে উপরে বলা অটিং আলগরিদম (ray shooting algorithm) এর মতো করে সমাধান করে থাকি। আমি প্রদত্ত বিন্দু হতে ঠিক ডান দিকে একটি রশ্মি টানি। যেহেতু তোমার কুরোরি (query) বিন্দুর সময় ওয়ালা একটি বিন্দু বহুভুজের শীর্ষ বিন্দু হতে পারে সেহেতু আমরা আমাদের প্রদত্ত বিন্দুটির দূরান্ককে  $y - \epsilon$  হিসেবে ভাবতে পারি, অর্থাৎ  $(x, y)$  যদি ভেতরে থাকে তাহলে  $(x, y - \epsilon)$  ভেতরে থাকবে, সুবিধা হলো  $y - \epsilon$  এ আঁকা  $x$  অক্ষের সমাত্রাল রেখা কোনো শীর্ষ বিন্দু দিয়া যাব না (এক্ষেত্রে অবশ্য তোমাকে দেখতে হবে এই বিন্দুটি তোমার বহুভুজের উপরে আছে কিন্তু থাকলে এই অল্প পরিমাণ সরানোর ফলে কিন্তু উভয় আলাদা হয়ে যেতে পারে)। এখন প্রতিটি বিন্দুতে হবে আর দেখতে হবে এটি এই রশ্মিকে ছেদ করে কিনা। যদি করে তাহলে সেই বাহুটি নির্দেশ করে উপরে যাচ্ছিল নাকি উপর থেকে নিচে (আমরা আসলে বাহুগুলোকে একটি ক্রমে নির্বো  $P_0P_1$ ,  $P_1P_2$  এরকম করে পর পর)? ধরা যাক এই দুটি ক্ষেত্রে যথাক্রমে আমরা ১ যোগ ও বিয়োগ করে আসলে এক অর্থে এটি জোড় বিজোড় বের করা। তাহলে সব বাহু বিবেচনা করার পর যদি দেখা আসলে যোগফল শূন্য তাহলে আমরা বুবুব যে আমাদের বিন্দু বাইরে আছে, আর যদি শূন্য না হয় তাহলে আমরা বুবুব যে আমাদের বিন্দু বাইরে আছে।

মনে কর তোমাদের কিছু বহুভূজ দেওয়া আছে, বলা হলো এদের ইউনিয়ন (union) এ ফ্রেক্ষাল বের করতে। ইউনিয়ন (union) বলতে অমরা বুঝি যদি কোনো জিনিস একধিকবা

আমরা কিন্তু লিখতে পারি  $H_0 = H_1 \times P + s_0 - s_n P^n$ . অর্থাৎ আমরা যদি  $H_1$  জানি তাহলে সূব সহজে  $H_0$  বের করে ফেলতে পারি। এর মানে আমরা পেছন থেকে হ্যাশিং ফাংশন এর ভালো বের করতে থাকলে খুব কম সময়েই সব জায়গার হ্যাশ মান বের করতে পারি। অথবা তুমি চাইলে পলিনোমিয়ালকে উল্লিখন দিতে পারো অর্থাৎ  $a_0 P^{n-1} + a_1 P^{n-2} + \dots$  তাহলে তুমি সাময়িক হেকেই যেতে পারবে। তাহলে এভাবে তোমার  $S$  এর সব জায়গার জন্য হ্যাশ মান বের করে সহজে মাত্র  $|S|$  সময়। আবার একটি জিনিস খেয়াল কর এখানে  $T$  কিন্তু নির্দিষ্ট, তাই এবে কিন্তু কোনো একটি বাকেটে ফেলা জরুরি না। তুমি চাইলে  $mod$  না করেই এই কাজ করতে পারো। মানে তুমি long বা long long ডেটা টাইপে যা হিসাব করার করবে। ওভারফ্লো (overflow) হল হবে, এসব নিয়ে মাথা ব্যাখা করতে হবে না। কারণ একই জিনিসকে যদি তুমি একইভাবে হ্যাশ কর তাহলে ওভারফ্লো হয়ে একই সংখ্যা হবে।

১১.২ নুথ-মরিস-প্র্যাট (Knuth-Morris-Pratt) বা KMP  
অ্যালগরিদম

আমরা কিছুক্ষণ আগে একটি স্থিতিগত ডেতের আবেকাট প্রত্যুৎসূ সাৰ-প্রস্তুৎসূ আকৰণে আছি কিনা তাৰে কৱলাম। তবে সমস্যা হলো আমরা জানি না আগেৰ পদ্ধতিতে কত সময় লাগবে। মানে আমৰ আশা কৱতে পাৰি যে এটি লিনিয়ার (linear) বা  $O(n)$  সময় নিবে তবে এটি যে সবসময়  $O(n)$  সময় নিবে তাৰ কোনো ঠিক নেই। হয়তো তোমাৰ হ্যাশ পদ্ধতিৰ উপৰ ভিত্তি কৰে এমন একটি ইলগুট দেওয়া সম্ভৱ হৈথানে অনেক সময় নিবে। কিন্তু আমরা এই সমস্যাকে হ্যাশ ছাড়া  $O(n)$  সময়ে সমাধান কৱতে পাৰি। এ জন্য বহুল প্ৰচলিত অ্যালগৱিদম হলো KMP বা নৃথ-মৰিস-প্ৰাট (Knuth-Morris-Pratt) অ্যালগৱিদম। এটি একটু কঠিন অ্যালগৱিদম। অ্যালগৱিদমটি কৃত হৈছিল এটি কিন্তু এটি বোৰা বিশেষ কৰে এটি কেন  $O(n)$  সময়ে কাজ কৰে তা বোৰা একটু কঠিন।

মনে পিস্ট আর দেখাব। বটে একটি স্ট্রিং  $T$  (Text) এর মধ্যে একটি স্ট্রিং  $P$  (Pattern) আছে কিনা তা বের করতে চাই। এজন্য আমাদের প্রথমে  $P$  এর প্রিফিক্স ফাংশন (prefix function) বের করতে হবে। আশা করি তোমাদের মনে আছে যে প্রিফিক্স (prefix) কী বা সাফিক্স (suffix) কী। প্রিফিক্স হলো কোনো স্ট্রিং এর শুরুর অংশ আর সাফিক্স হলো শেষের অংশ। যেমন  $xiox$  একটি স্ট্রিং হলে এর প্রিফিক্স হবে  $\{x, xi, xio, xiox\}$  আর সাফিক্স হবে  $\{xiox, iox, ox, x\}$ । প্রোপার প্রিফিক্স (Proper prefix) হলো ওই স্ট্রিং বাদে ওই স্ট্রিংয়ের অন্তর্যান প্রিফিক্স। যেমন এই স্ট্রিংয়ের জন্য প্রোপার প্রিফিক্স হলো  $\{x, xi, xio\}$ . তাহলে প্রিফিক্স ফাংশন কী? যাৰা যাব আমৰা প্রিফিক্স ফাংশনকে দিয়ে প্রকাশ কৰব। তাহলে  $p(i)$  হবে  $P[0 \dots i]$  এর সবচেয়ে বড় প্রোপার প্রিফিক্সের "দৈর্ঘ্য" যেন তা তাৰ সাফিক্সও হয়। যেমন যদি  $P[0 \dots 5]$  হয়  $aabccaaab$  তাহলে 3 দৈর্ঘ্যের প্রোপার প্রিফিক্স পাওয়া যাবে যা সাফিক্সও এবং এটি হলো  $aab$ । এখানে প্রোপার প্রিফিক্স বলতে কাৰণ হলো অবি তো চাইলে পুৰো স্ট্রিং নিয়ে বলতে পারতাম যে এটি সাফিক্সও প্রিফিক্সও। বলতে কাৰণ হলো অবি তো চাইলে পুৰো প্রোপার প্রিফিক্স। তাহলে আমৰা একটি স্ট্রিংয়ের সব কিছি এটি আমৰা দাই না, এজন্য আমি বলেন্নো প্রোপার প্রিফিক্স। তাহলে আমৰা একটি স্ট্রিংয়ের সব কিছি আমৰা দাই না বলে কৰি।

289

index	0	1	2	3	4	5	6
P	a	b	a	b	a	c	a
=	-1	-1	0	1	2	-1	0

କିଛୁକଣ ଆଗେ ବଲା ପ୍ରଫିକ୍ରୁ ଫାଂଶନେର ସଂଜ୍ଞାର ସଦେ ଦେଖିବେ ଟେବିଲ ୧୧.୧ ଏଇ ପାଇଁ ମାନେ ମିଳ ନେଇ । କିନ୍ତୁ ଖୁବ ବେଶି ଯେ ଅମିଲ ତାପ କିନ୍ତୁ ନା । ଏଠି ସଂଜ୍ଞା ମୋତାବେକ ମାନେର ଥେକେ ଏକ କମ ଆସିଲେ ଆମରା ଏଥାନେ ସ୍ଟ୍ରିଂଟିକେ  $0 - index$  ହିସେବେ ବିବେଚନା କରଇଛି ଏବଂ  $\pi(i)$  ଏଇ ମାନ ଏମିହିବେ ଯେଣ  $P[0 \dots \pi(i)] = P[i - \pi(i) + 1 \dots i]$  । ଅର୍ଥାତ୍  $\pi$  କେ ଆମରା ଠିକ ଦୈର୍ଘ୍ୟ ଦିଲେ ନା ବରା ଇନଡେକ୍ସ (index) ଦିଲେ ସଂଜ୍ଞାଯିତ କରବ । ଏହି ଟେବିଲ ଏ  $\pi(0) = \pi(1) = -1$  କେନ୍ତା ଏକାବଲା ଦରକାର । କାରଣ ହଲୋ ଆମରା ମନେ କରନ୍ତେ ପାରି ଯେ  $P[0 \dots -1]$  ହଲୋ ଏକଟି ଫାଁକା ସ୍ଟ୍ରିଂ ଏବଂ ଯେହେତୁ  $a$  ବା  $ab$  ଏଇ ଏମନ କେନୋ ପ୍ରୋପାର ପ୍ରଫିକ୍ରୁ ନେଇ ଯା ସାଫିର୍ରୁ ଓ ତାଇ ଆମରା ଧରେ ନେଇ ଯେ ଫାଁକା ସ୍ଟ୍ରିଂ ହବେ ଏହି ପ୍ରଫିକ୍ରୁ ବା ସାଫିର୍ରୁ । ଆସିଲେ ସତ୍ୟ ବଲତେ ଏଠି ବଲାର ଜନ୍ୟ ବଲା,  $-1$  ନା ଦିଲେ ପରବର୍ତ୍ତି ଅଂଶ କୋଡ କରନ୍ତେ ଝାମେଲା ହବେ ବା ଯଦି ଆମରା  $0 - index$  ନା ମନେ କରେ ଯଦି  $1 - index$  ନିଭାବ ତାହାଲେ ପୁରୋ ଜିଲ୍ଲା ଅନେକ ସହଜ ହତୋ ଏବଂ ଏଥାନେ  $-1$  ନା ହୁୟେ  $0$  ହତୋ । ଯାଇ ହେବ, ତୁମ୍ହି ସଥିର ପୁରୋ ଅୟାଲଗରିଦମତି ବୁଝେ ଯାବେ ତଥନ ଏସବ କେନ କୀ କରାଇ ତାପ ବୁଝାତେ ପାରବେ ତାହିଁ ଏସବ ନିର୍ଦ୍ଦେଶ ଏଥିର ଅତ ବେଶି ଚିନ୍ତା କରାର କିଛି ନେଇ । ଏଥନ ତାହାଲେ ଟେବିଲ ୧୧.୧ ଏଇ ପାଇଁ ମାନେ ଏକବାର ଢାରୁ ଲିଲିସେ ନାଓ । ଦେଖ ବାଦ ବାକି ସବ ମାନ ଠିକ ଆଛେ କିନା । ଏଥନ ଆମାଦେର ପ୍ରଶ୍ନ ହଲୋ ଏହି  $\pi$  ଏଇ ସବ ମାନ କୀତାବେ ଆମରା ଲିନିଆର ସମଯେ ବେର କରନ୍ତେ ପାରି ।

অপ্রমত  $\pi(0) = -1$ , এখন তুমি সর্বশেষ  $\pi$  এর মানকে একটি ভ্যারিয়েবলটি *now*. এখন ১ হতে  $|P| - 1$  পর্যন্ত  $i$  এর একটি লুপ চালাও। আমরা  $\pi(i)$  বের করতে চাই। এজন্য আমাদের যা করতে হবে তা হলো  $P[now + 1]$  এবং  $P[i]$  সমান কিনা তা দেখতে হবে। যদি না হয় তাহলে  $now = \pi(now)$  করব। আর যদি সমান হয় বা  $now = -1$  হয় তাহলে এসব না করে এই লুপ থেকে বের হতে হবে। তবে আমাদের আবারও  $P[now + 1]$  এবং  $P[i]$  তুলনা করব এবং যদি সমান হয় তাহলে  $now$  কে এক বাড়াব এবং  $\pi(i)$  এই মান রাখব। আর যদি সমান না হয় তাহলে  $now = \pi(i) = -1$  করব। এতক্ষণ যা বললাম তা এবং রকম অ্যালগরিদমের বর্ণনা। কিন্তু আমাদের বুঝতে হবে কেন আমরা এরকম করছি।

আমরা প্রথমে terminal কেইস now = -1 নিয়ে চিন্তা না করে একটি সাধারণ কেইস (general case) নিয়ে চিন্তা করে দেখি। মনে কর কোনো এক i এর জন্য  $\pi(i)$  জানি, এখন আমরা বের করতে চাই  $\pi(i+1)$  এর মান।  $\pi(i)$  মানে কী? এর মানে হলো  $P[0\dots\pi(i)] = P[i-\pi(i)+1\dots i]$  এবং এরকম সবগুলোর মধ্যে  $\pi(i)$  সর্বোচ্চ (অবশ্যই i এর থেকে ছোট) এখন এরকম আমরা সবচেয়ে বড়  $\pi(i+1)$  বের করতে চাই। খেয়াল কর যদি  $P[\pi(i)+1] = P[i+1]$  হতো তাহলে আমরা খুব সহজেই বলতে পারতাম যে  $\pi(i+1) = \pi(i) + 1$  কারণ এর থেকে বড় কিন্তু হওয়া সম্ভব না, হলে  $\pi(i)$  আরও বড় হওয়া সম্ভব হতো তাই না? বা অন্যভাবে

বলতে হলে বলতে হয়  $\pi(i+1) - 1$  কিন্তু  $\pi(i)$  এর একটি candidate. সুতরাং আমরা যদি দেখি  $P[\pi(i)+1] = P[i+1]$  তাহলে  $\pi(i+1) = \pi(i) + 1$ . এখন প্রশ্ন হলো যদি না হয়? আমাদের লক্ষ্য  $P[0\dots i+1]$  এর একটি সাফিক্স বের করা যা প্রিফিক্সও বা অন্যভাবে বলতে হলে বলা যায়  $P[0\dots i]$  এর একটি সাফিক্স বের করা যা প্রিফিক্সও এবং সেই প্রিফিক্সের পরের ক্যারেক্টার  $P[i+1]$  এর সমান। আমরা একটি প্রিফিক্স ইতোমধ্যেই চেষ্টা করেছি আর তা হলো  $P[0\dots \pi(i)]$ . এর পরের বড় candidate সাফিক্স কোনটি হবে? সেটি কিন্তু ইতোমধ্যেই বের করে রেখেছে আর তা হলো  $\pi(\pi(i))$ . কেন? খেয়াল কর আমরা চাই  $\pi(i)$  এর থেকে ছোট সাফিক্স যা প্রিফিক্সও। ধরা যাক এরকম কোনো একটি সাফিক্স বা প্রিফিক্স হলো  $Z$ . এই  $Z$  এর বৈশিষ্ট্য হলো এটি একই  $S$  সঙ্গে  $P[0\dots \pi(i)]$  এর প্রিফিক্স এবং সাফিক্স। আসলে এদের মধ্যে সবচেয়ে বড়টি আমাদের দরকার। আর সেটাই কিন্তু  $\pi(\pi(i))$  তাই না? উদাহরণ দেওয়া যাব। মনে কর আমরা *ababa* এর জন্য  $\pi(4)$  জানি আর তা হলো 2. এখন মনে কর আমাদের পরের ক্যারেক্টার হলো *c* যা  $P[3]$  এর সমান না। তাই আমাদের *aba* এর থেকে ছোট এমন একটি স্ট্রিং দরকার যা *ababa* এর একই  $S$  সঙ্গে সাফিক্স ও প্রিফিক্স। এবং সেটি কিন্তু আবার *aba* এরও প্রিফিক্স ও সাফিক্স। তাই আমরা যদি  $\pi(2)$  দেখি তাহলে *a* পাব যা *ababa* এর প্রিফিক্স ও সাফিক্স। তো আমরা যদি এটুকু বুঝে থাকো তাহলে আশা করি কোড ১১.১ ও বুবাতে পারবে। একটি জিনিস বলা হয়নি তা হলো আমরা এতক্ষণ সাধারণ কেইস নিয়ে চিন্তা করেছি। Terminal কেইস নিয়ে বলা হয়নি। খেয়াল কর কিছুক্ষণ আগের উদাহরণে যখন আমরা দেখেব যে  $P[1]$  ও  $c$  না তখন আমরা আবার  $\pi(0)$  করব আর এক্ষেত্রে আমরা পাব  $-1$ . এখন প্রথম কথা হলো এই লুপ আজীবন চলতে পারে না, এক সময় আমাদের শেষ করতে হবে আর এছাড়াও তোমরা  $\pi(-1)$  কল করতে পারে না কারণ এটি বলে কিছু নেই, তাই যখন  $now = -1$  হয়ে গেছে তখন আমরা লুপ থেকে বের হয়ে গেছি। তবে এই লুপ থেকে বের হওয়ার দুটি মানে আছে এক  $now + 1$  এর সঙ্গে মিলে গেছে দুই মিলে নি মানে  $-1$  হবে। এটি যাচাই করার জন্যই আমাদের এই লুপের শেষে একটি if-else আছে।

## ক্ষেত্র ১১.১: prefix function.c

```
1 int pi[100];
2 char P[100];
3
4
5 int prefixFunction() {
6     int now;
7     pi[0] = now = -1;
8     int len = strlen(P);
9     for (int i = 1; i < len; i++) {
10         while (now != -1 && P[now + 1] != P[i])
11             now = pi[now];
12         if (P[now + 1] == P[i]) pi[i] = ++now;
13     }
14 }
```

তবে এখনো আমাদের ম্যাটিং (matching) সমস্যার সমাধান হয়না। আমরা কেবলমাত্র আমাদের প্যাটার্ন (pattern) অর্থাৎ  $P$  এর প্রিফিক্স ফাংশন বের করলাম। এখন আমাদের কাজ হলো  $P$  কি  $T$  এর ভেতর আছে কিনা তা বের করা। এজন্য আমরা কিছুটা আগের মতোই কাজ করব। প্রথমে  $now = -1$  নাও এবং  $T$  এর উপর দিয়ে ০ হতে  $|T| - 1$  পর্যন্ত একটি লুপ চালাও।  $i$  এর লুপে যখন ঢুকে তখন  $now$  নির্দেশ করবে  $T[0 \dots i - 1]$  এর দীর্ঘতম সাফিক্স যা  $P$  এর প্রিফিক্স। এখন আমাদের বিবেচনা করতে হবে  $T[i]$ । আগের মতো প্রথমে দেখো যে  $P[now + 1]$  কি  $T[i]$  এর সমান কিনা। হলে তো  $now$  কে এক বাড়িয়ে দিবে। আর যদি না হয় তাহলে  $now = \pi(now)$  করবে এবং আবারও একই জিনিস যাচাই করবে। আর যদি  $now = -1$  হয়ে যায় তাহলে কী করতে হবে তা তো বুঝতেই পারছ। কোড ১১.২ এ আমরা এটি কোড করে দেখালাম।

## কোড ১১.২: kmp.cpp

```

1 int pi[100];
2 char P[100], T[100];
3
4 int kmp() {
5     int now;
6     now = -1;
7     int n = strlen(T);
8     int m = strlen(P);
9     for (int i = 0; i < n; i++) {
10         while (now != -1 && P[now + 1] != T[i])
11             now = pi[now];
12         if (P[now + 1] == T[i]) ++now;
13         else now = -1;
14         if (now == m) return 1;
15     }
16     return 0;
17 }

```

এখন কথা হলো এর টাইম কমপ্লেক্সিটি কত? দুটি লুপ দেখে যদি ভাব এটি  $O(n^2)$  তাহলে  
 ভুল ভাববে। খেয়াল কর for লুপের একটি ইটারেশন (iteration) এ now এর মান খুব জোড়  
 এক বারে। আবার while লুপে now এর মান কখনও বাড়ে না, শুধুই কমে। তাই while লুপ  
 আসলে সর্বমোট লিনিয়ার সময় চলে। অর্থাৎ আমাদের প্রিফিল্ড ফাংশন বের করার কোড সময় নেটে  
 $O(|P|)$  আর ম্যাট্রিচিয়ের কোড সময় নেয়  $O(|T|)$ .

## ১১.২.১ KMP সম্পর্কিত কিছু সমস্যা

ধৰা যাক  $P$  কি  $T$  এর মধ্যে আছে কিনা শুধু এটাই জিজ্ঞাসা করেনি বরং কত বার আছে তাও জানতে চেয়েছে। তুমি কী করবে? একটি সহজ বুদ্ধি হলো  $P \# T$  এর প্রিফিল্র ফাংশন (একে ফেইলিউর ফাংশন বা ইংরেজীতে failure function ও বলে) বের করা। এখানে  $\#$  হলো এমন একটি ক্যারেক্টোর যা  $P$  বা  $T$  কারও ভেতরে নেই। তাহলে প্রিফিল্র ফাংশনে যতবার  $|P|$  আসবে সেটই তোমার উন্নত। এছাড়াও আরেকটি উপায় হলো উপরে আমরা যখন  $P$  কে  $T$  এর ভেতরে ঝুঁজেছিলাম তখন যে  $now = |P|$  হলেই ১ রিটার্ন করেছিলাম তা না করে আমরা তখন একটি counter এর মান বাড়াবো এবং  $now = \pi(now)$  কর করব।

ଆରେକଟି ସମସ୍ୟା ଏକମ ହତେ ପାରେ ଯେ ଆମାଦେର ଶୁଣ୍ଟି  $P$  କଠିତର ପାଓୟା ଗେହେ ତାହିଁ ଜାନନ୍ତେ ଚାଯନି ବରଂ  $P$  ଏଇ ସବ ପ୍ରିଫିକ୍ସ୍ କଠିତର  $T$  ତେ ଆହେ ତା ଜାନନ୍ତେ ଚାଓୟା ହେଁବେ। କୀବାବେ କରବେ? ଯା କରତେ ହେବେ ତା ହଲୋ  $T$  ତେ  $P$  ଖୋଜାଇ ବେଳେ ଲୁପ୍ରେ ଥେବେ ପ୍ରତିବାର ଏକଟି ଅର୍ଥାତ୍ ଏକଟି *now* ତଥା ହାଲରେ ମାନ ଏକ ବାଢ଼ାଇବେ। ଅର୍ଥାତ୍ ଆମରା *now* ପରସ୍ତ ପ୍ରିଫିକ୍ସ୍ ପେରେଇ ଏଟି ବୋଲାବାତେ କିନ୍ତୁ ଶୁଣ୍ଟି ଏଟି କରାଲେ କିନ୍ତୁ ହେବେ ନା । ଉଦାହରଣସ୍ଵରୂପ  $P = aa$  ମନେ କର ଆର  $T = aaaaa$ . ଏଥିନ ଏଟି ତୋ ବୁଝେଇ ଯେ ପ୍ରାୟେ ସବସମୟ ଆମରା *now = 1* ଏଇ ମାନ ବାଢ଼ାବ କିନ୍ତୁ *now = 0* ଏଇ ମାନ କିନ୍ତୁ ତେମନେ ବାଢ଼ାବେ ନା ଯଦିଓ ପ୍ରିଫିକ୍ସ୍  $a$  ବହବାର  $T$  ତେ ଦେଖା ଯାଇ । ସୁତରାଂ ଆମାଦେର ଯା କରବେ ହେବେ ତା ହଲୋ ବାଢ଼ାବେ ନା ଯଦିଓ ପ୍ରିଫିକ୍ସ୍  $a$  ବହବାର  $T$  ତେ ଦେଖା ଯାଇ । ଏବଂ ପ୍ରଥମ ଏଇ  $T$  ତେ  $P$  ଖୋଜାଇ କାଜ ଶେଷ ହଲେ,  $P$  ଏଇ ଶେଷ ଥେକେ ଶୁଣ୍ଟି ଲୁପ୍ଟ ଚାଲାତେ ହେବେ । ଏବେ ଏବଂ ପ୍ରଥମ ଏଇ  $T$  ତେ  $P$  ଖୋଜାଇ କାଜ ଶେଷ ହଲେ,  $P$  ଏଇ ଶେଷ ଥେକେ ଶୁଣ୍ଟି ଲୁପ୍ଟ ଚାଲାତେ ହେବେ । କେମି? କାରିଙ୍ଗ ହଲୋ? ଏ ଶେଷ ହେଁବେ ଜଣ୍ଯ  $count[\pi(i)]$  ଏଇ ମାନ  $count[i]$  ପରିମାଣ ବାଢ଼ାଇବେ । କେମି? କାରିଙ୍ଗ ହଲୋ? ଏ ଶେଷ ହେଁବେ ସବଚରେ ବେଳ ସାଫିକ୍ସ୍ ଯା ପ୍ରିଫିକ୍ସ୍ ଓ ତାତେ କିନ୍ତୁ ତୁମ ଆରା ଓ  $count[i]$  ବାର ଯେତେ ପାରାତେ ଯା ଆମେ ହିସାବ କରିଲି ।

ধৰ একটি স্ট্রিং দিয়ে বলা হলো এতে কয়টা বৃত্তি সাবস্টিং আছে। এই প্রিফিক্স ফাংশন আমরা জানি। এ থেকে কীভাবে করা যায়? মনে কর স্ট্রিংটি হলো  $S$  এবং এর প্রিফিক্স ফাংশন আমরা জানি। এ থেকে আমরা সবচেয়ে বড় মান  $k$  বের করলাম। এর মানে কী? এর মানে হলো এই স্ট্রিংয়ের  $k+1$  হতে  $|S|$  দৈর্ঘ্যের মে প্রিফিক্স তা এই স্ট্রিংয়ে আর কথা ও আসেন। কিন্তু  $S[1]$  থেকে মেসৰ অনন্য হতে পারে কীভাবে এর করব? সহজ,  $S[1\dots]$  এর জন্য আবার প্রিফিক্স (unique) সাবস্টিং আছে সেসব কীভাবে এর করব? সহজ,  $S[1\dots]$  এর জন্য আবার প্রিফিক্স ফাংশন বের কর। এভাবে  $S$  এর প্রতিটি সাফিক্সের জন্য অনন্য (unique) প্রিফিক্সের সংখ্যা যোগ করলে আমরা মোট বৃত্তি সাবস্টিং-এর সংখ্যা পেয়ে যাব। এজন্য আমাদের সময় লাগছে  $O(n^2)$ । করলে আমরা মোট  $n$  বৃত্তি সাবস্টিং করলেও হবে এমন কোনো ছোট স্ট্রিং  $P$  আছে যা কোথা বাবে আর বাবে

একটি ট্রিঙ্গুলেশন  $S$  দেওয়া আছে যাক।  
 পুনরাবৃত্তি করলে  $S$  পাওয়া যায়। যদেন  $S = ababab$  এখনে  $P = abababababab$ ।  
 এর সমাধান হলো  $n$  দেখতে হবে যে  $n - \pi(n-1) + 1$  কিন্তু  $n$  কে ভাগ করে কিনা।  
 করলে সেই দৈর্ঘ্যের প্রিফিলাই হবে উভর। এর প্রমাণ একটু কঠিন। তোমরা একটু চিন্তাভাবনা করে  
 proof by contradiction পদ্ধতিতে এই জিনিস প্রমাণ করতে পার।

এক বারে। আবার while লুপে now এর মান কখনও বাড়ে না, শুধুই কমে। তাই while লুপ আসলে সর্বমোট লিনিয়ার সময় চলে। অর্থাৎ আমাদের প্রিফিল ফাংশন নের করার কোড সময় নেয়  $O(|P|)$  আর ম্যাটিংয়ের কোড সময় নেয়  $O(|T|)$ .

### ১১.২.১ KMP সম্পর্কিত কিছু সমস্যা

ধৰা যাক  $P$  কি  $T$  এর মধ্যে আছে কিনা শুধু এটাই জিজ্ঞাসা করেনি বরং কত বার আছে তাও জানতে চেয়েছে। তামি কী করবে? একটি সহজ বুদ্ধি হলো  $P \# T$  এর প্রিফিল্যু ফাংশন (একে ফেইলিউর ফাংশন বা ইংরেজিতে failure function ও বলে) বের করা। এখানে  $\#$  হলো এমন একটি ক্যারেটোর যা  $P$  বা  $T$  কারও ভেতরে নেই। তাহলে প্রিফিল্যু ফাংশনে যতবার  $|P|$  আসবে সেটাই তোমার উত্তর। এছাড়াও আরেকটি উপায় হলো উপরে আমরা যখন  $P$  কে  $T$  এর ভেতরে খুঁজেছিলাম তখন যে  $now = |P|$  হলৈই ১ রিটার্ন করেছিলাম তা না করে আমরা তখন একটি counter এর মান বাঢ়াবো এবং  $now = \pi(now)$  কল করব।

ଆରେକଟି ସମସ୍ୟା ଏରକମ ହତେ ପାରେ ଯେ ଆମାଦେର ଶୁଣୁ $P$  କଠବାର ପାଓୟା ଗେଛେ ତାହିଁ ଜାନତେ ଚାଯାନି ବରଙ୍ଗ  $P$  ଏର ସବ ପ୍ରିଫିକ୍ସ୍‌ର କଠବାର  $T$  ତେ ଆହେ ତା ଜାନତେ ଚାଓ୍ୟା ହୋଇଛି। କୀଟାବେ କରବେ? ଯା କରତେ ହବେ ତା ହଲୋ  $T$  ତେ  $P$  ଖୁଣ୍ଜାର while ଲୂପର ଶେଷେ ପ୍ରିତବାର ଏକଟି ଅୟାରେତେ now ତମ ହୃଦୟରେ ମାନ ଏକ ବାଡ଼ାତେ ହବେ । ଅର୍ଥାତ୍ ଆମରା now ପରିଷ୍ଠା ପ୍ରିଫିକ୍ସ୍ ପୋହେଛି ଏଟି ବୋଲାଇ । ଏଥିର ଶୁଣୁ ଏଟି କରିଲେ କିନ୍ତୁ ହେବା ନା । ଉଦାହରଣବରୁପ  $P = aa$  ମନେ କର ଆର  $T = aaaa$  । ଏଥିନ ଏଟି ତୋ ବୁଝୋଇଛି ସେ ପ୍ରାୟ ସବସମୟ ଆମରା now = 1 ଏର ମାନ ବାଡ଼ାବ କିନ୍ତୁ now = 0 ଏର ମାନ କିନ୍ତୁ ତେମନେ ବାଡ଼ବେ ନା ସିଦ୍ଧି ପ୍ରିଫିକ୍ସ୍  $a$  ବହୁବାର  $T$  ତେ ଦେଖା ଯାଏ । ଶୁରୁତ୍ବ ଆମାଦେର ଯା କରତେ ହେବା ତା ହଲୋ ବାଡ଼ବେ ନା ସିଦ୍ଧି ପ୍ରିଫିକ୍ସ୍  $a$  ବହୁବାର  $T$  ତେ ଦେଖା ଯାଏ । ଏବଂ ପ୍ରତି  $i$  ଏର  $T$  ତେ  $P$  ଖୁଣ୍ଜାର କାଜ ଶେଷ ହଲେ,  $P$  ଏର ଶେଷ ଥେକେ ଶୁରୁତ୍ବ ଲୁପ ଚାଲାତେ ହବେ । କେବେ? କାରିଙ୍ଗ ହଲୋ ? ଏ ଶେଷ ହେଯା ଜଣ୍ଯ count[ $\pi(i)$ ] ଏର ମାନ count[i] ପରିମାଣ ବାଡ଼ାତେ ହବେ । କେବେ? କାରିଙ୍ଗ ହଲୋ ? ଏ ଶେଷ ହେଯା ସବଚେଳେ ବଢ଼ ସାଫିକ୍ସ୍ ଯା ପ୍ରିଫିକ୍ସ୍ ଓ ତାତେ କିନ୍ତୁ ତୁମ୍ହାର ଓ count[i] ବାର ମେତେ ପାରତେ ଯା ଆଗେ ହିସାବ କରନି ।

একটি স্ট্রিং  $S$  দেওয়া আছে।  
 পুনরাবৃত্তি করলে  $S$  পাওয়া যায়। যদেন  $S = ababab$  এখানে  $P = ab$  কে তৈরি করে থাকে।  
 $S$  পাওয়া যায়। এর সমাধান হলো দেখতে হবে যে  $n - \pi(n-1) + 1$  কি  $n$  কে ভাগ করে কিম।  
 করলে দেই দৈর্ঘ্যের প্রতিক্রিয়া হবে উভর। এর প্রমাণ একটু কঠিন। তোমরা একটু চিন্তাভাবনা করে  
 proof by contradiction পদ্ধতিতে এই জিনিস প্রমাণ করতে পার।

۱۵۲

### ১১.৩ Z অ্যালগারদম

KMP তে যেমন প্রিফিল ফাংশন ছিল এখানে আছে 'z' ফাংশন।  $z(i)$  এর মানে হলো স্ট্রিংয়ের  $i$  তম ইনডেক্স হতে শুরু করে কত বড় সাব-স্ট্রিং পাওয়া যায় যা মূল স্ট্রিংয়ের প্রিফিল। যেমন  $ababc$  এই স্ট্রিংয়ের জন্য 'z' ফাংশনের মানগুলো হবে  $\{0, 0, 2, 0, 0\}$ . এখানে  $z(0)$  এর মান  $0$  করা হচ্ছে। কারণ কিছুটা KMP এর  $\pi(0)$  এর মতো। আর তাছাড়া এটি আমাদের কোড সহজ করবে।

এখন আসা যাক এটি কীভাবে বের করা যায় সেই প্রশ্নে। অবশ্যই  $O(n^2)$  সময়ে বের করা কোনো ব্যাপার না। আমরা চাই লিনিয়ার সময়ে একটি স্ট্রিং  $S$  এর 'z' ফাংশন বের করতে। আমাদের এজন্য দুটি ভ্যারিয়েবলের দরকার একটি  $left$  এবং  $right$ । প্রথমে আমরা  $z(0) = left = right = 0$  সেট করব। এখন আমরা  $i = 1$  হতে  $|S| - 1$  পর্যন্ত একটি লুপ চালাব। লুপের ভেতর কী করব তা জানার আগে  $left$  আর  $right$  এর মানে জানলে ভালো হয়। লুপের  $i$  তম অবস্থানে এই দুটি ভ্যারিয়েবল প্রকাশ করে যে  $[0, i - 1]$  এই সীমার কোন মান  $x$  এর জন্য  $x + z(x)$  এর মান সর্বোচ্চ হয়। অর্থাৎ  $S[left \dots right]$  সাব-স্ট্রিংটি  $S$  এর একটি প্রিফিল্ড হবে এবং  $0 < left < i$  হবে আর এরকম সব  $candidate$  এর মধ্যে  $right$  সর্বোচ্চ হয়। অর্থাৎ প্রতিবার লুপের ভেতরে  $z(i)$  এর মান বের হয়ে গেলে আমাদের দেখতে হবে যে  $i + z(i) - 1$  কি  $right$  এর থেকে বেশি কিনা। বেশি হলে  $left = i, right = i + z(i) - 1$  করতে হবে।

এখন কথা হলো এই  $z(i)$  এর মান কীভাবে বের করা যায়। প্রথমে আমরা দেখব যে  $i \leq right$  কিনা। হলে আমরা  $z(i)$  এর মানকে এক লাফে অনেক দূর বাড়াতে পারব। কী রকম? খেয়াল কর  $S[left...right]$  হলো  $S$  এর একটি প্রিফিক্স বা  $S[0...(right - left)]$  এর সমান। যেহেতু  $left < i$  এবং  $i \leq right$ ? তাহলে এটি বলা যায় যে  $S[i...right]$  হলো  $S[(i-left)...(right-left)]$  এর সমান। এবং যেহেতু  $i-left < i$  তাই আমরা কিন্তু  $z(i-left)$  এর মান আগে থেকেই জানি। এবং আমরা বলতে পারি যে  $z(i)$  এর মান কিছুটা  $z(i-left)$  এর মতো হবে। কারণ ওই যে বললাম  $S[i...right]$  হলো  $S[(i-left)...(right-left)]$  এর সমান। এখন খেয়াল কর আমরা সরাসরি  $z(i) = z(i-left)$  করতে পারব না। কেন? কারণ আর যাই হোক  $i + z(i) - 1 > right$  হতে পারবে না কারণ আমরা  $S[right]$  এর পরের তথ্য জানি না। তাই যা করতে হবে তা হলো  $z(i) = \min(z(i-left), right - i + 1)$  করতে হবে। এটি  $z(i)$  এর একটি lower limit। অর্থাৎ  $z(i)$  কমপক্ষে এত হবেই। এর থেকে বড় হবে কিনা তা নিশ্চিত না। তাই যা করতে হবে তা হলো একটি লুপ চালিয়ে আরও বড় করা যায় কিনা তা দেখতে হবে। এভাবে  $z(i)$  এর মান বের করতে হবে। এর কোড ১১.৩ এ দেওয়া হলো।

### কোড ১১.৩: zfunction.cpp

```

8 void zfunction() {
9     int left, right;
10    z[0] = left = right = 0;
11    int len = strlen(S);
12    for (int i = 1; i < len; i++) {
13        if (i <= right)
14            z[i] = min(z[i - left], z[right - i + 1]);
15        while (i + z[i] < len
16              && S[i + z[i]] == S[z[i]])
17            z[i]++;
18        if (i + z[i] - 1 > right)
19            left = i, right = i + z[i] - 1;
20    }
21 }

```

উদাহরণ দেখা যাক, মনে কর  $ababab$  এর 'z' ফাংশন বের করছি, আমরা  $z(2) = 4$  বের করে ফেলার পর  $z(4)$  এর মান কিন্তু সেই  $z(2)$  এর তথ্য থেকে 2 পেয়ে যাব। কীভাবে? দেখ 4 হলে  $left = 2$  আর  $right = 2 + 4 - 1 = 5$  এর ভেতরে। সুতরাং আমরা  $z(4) = \min(z(2) = 4, 5 - 4 + 1 = 2) = 2$  করব। এর পর আর পরের while লুপ চলবে না কারণ  $4 + z(4) < len$  না। এর মানে  $z(4)$  বের করতে আমাদের কোনো কাজই করতে হচ্ছে না।

এখন কথা হলো এটি কেন লিনিয়ার? খেয়াল কর for লুপের ভেতরে রেই if আছে সেখানে  
 যদি  $z(i) = right - left + 1$  দিয়ে bound হয় অর্থাৎ  $i$  হতে শুরু করা স্ট্রিংটা  $right$  গিয়ে আটকে যায়, তাহলে আমরা while লুপ দিয়ে  $right$  এর মান বাড়ানোর চেষ্টা করি। আর যদি  
 $right$  এর আগেই bound হয়ে যায় তাহলে কিন্তু এই while লুপের সমতার শর্ত সত্য হবে না  
 তাই কোনো কাজ না করেই এই লুপ শেষ হয়ে যাবে। অর্থাৎ এই while লুপ কিন্তু সবসময়  $right$   
 এর মান বাড়াবে। আর করতই বা বাড়াবে?  $len = |S|$  এর থেকে তো আর বড় না তাই না? তাঁ  
 এটি লিনিয়ার।

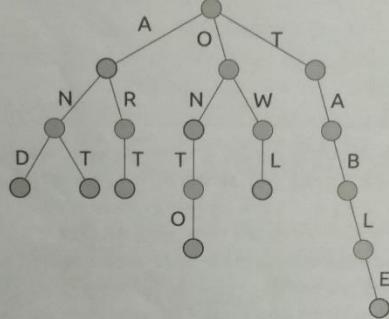
### ১১.৩.১ Z অ্যালগরিদম সম্পর্কিত কিছু সমস্যা

এখন এর মাধ্যমে কী কী সমস্যা সমাধান করা যায়? KMP দিয়ে সমাধান করা যাবে। বেশির ভাগ সমস্যাই সমাধান করা সম্ভব। যেমন  $P$  কি  $T$  এর ভেতর আছে কিনা? এটি সমাধানের জন্য  $S = P + \# + T$  করে দেখ যে কতগুলো  $z(i)$  এর মান  $|P|$  তালিই হয়ে থালে। একটি স্ট্রিং নিয়ে একইভাবে একটি স্ট্রিংয়ে কতগুলো স্বতন্ত্র সার-স্ট্রিং আছে তাও বের করা সহজ। একটি স্ট্রিং নিয়ে তার ' $z'$  ফাংশনের মান বের কর। ধরা যাক সর্বোচ্চ মান  $x$ , এর মানে  $S[1\dots x]$ । এ অবস্থা কখনে এক সময়  $S[0\dots x-1]$  এই সার-স্ট্রিং পাব। সুতরাং এই সার-স্ট্রিং নিয়ে এখন না চিন্তা করে

$S[0\dots(x-|S|-1)]$  এই সাব-ট্রিংগুলো নিয়ে চিন্তা করব। অর্থাৎ এখন আমাদের স্বতন্ত্র সাব-ট্রিংয়ের count ( $|S|-1$ )  $- x + 1$  বাড়াব। আব এর পরে  $S[1\dots]$  নিয়ে চিন্তা করব। এভাবে  $|S|$  বার  $S$  এর  $|S|$  টি সাফল্য নিয়ে কাজ করব।

## ੧੧.੮ ਟ੍ਰੈਇ (Trie)

এটি কথা বলে বোঝানোর থেকে ছবিতে বোঝানো সহজ। চিত্র ১১.১ এ {a, and, ant, an, on, onto, owl, table} শব্দসমূহের জন্য ট্রাই একে দেখানো হলো।



ନକ୍ଶା ୧୧.୧: {a, and, ant, art, on, onto, owl, table} ଶବ୍ଦସମୂହରେ ଜନ୍ୟ ଟ୍ରୋ

চিত্র থেকে খুব সহজেই বোায়া যাওয়ার কথা ট্রাই আসলে কী। এটি ট্রি (tree) আকারে তোমাদে দেওয়া সব শব্দকে উপস্থাপন করে। সাধারণ প্রিফেরেন্স জন্য একটিই মাত্র নেট থাকে। যেমন উপরের উদাহরণ এ ant আর art দুটির জন্যই একটি নোড দুটি একই। আমরা বেশি কথা না বলে সরাসরি কোতে চলে যাব এবং আসলে এই কোত ব্যাখ্যা করারও কিছু নেই। আশা করি তোমার কোত ১১.৪ দেখে বুঝতে পারবে আমরা কী করছি এবং কেন করছি। আর তোমাদেরকে যদি বল হয়- আচ্ছা অনুক শব্দ এই ট্রাইয়ে আছে কিনা- তাও আশা করি বের করতে পারবে। তাও বলি কৃত থেকে ট্রাভার্স (traverse) শুরু করবে আর দেখবে বর্তমান নোডে বর্তমান ক্যারেঞ্চার দিয়ে লেবেলে কাছ বাছ (edge) আছে কিনা না থাকলে সেই শব্দ নেই। আর থাকলে এভাবে এগোতে থাকতে পারবে। শোন গিয়ে দেখতে হবে শোনের নোডে is/IS এবং মার্ক করা আছে কিনা।

```

> #define MAX_NODE 100000
2 #define MAX_LEN 100
3
4 char S[MAX_LEN];
5 // assuming words only have small letters ['a', 'z']
6 int node[MAX_NODE][26];
7 int root, nnode;
8 int isWord[MAX_NODE];
9
10 // call before inserting any words into trie.
11 void initialize() {
12     root = 0;
13     nnode = 0;
14     for (int i = 0; i < 26; i++)
15         // -1 means no edge for ('a' + i)th
16         // character
17         node[root][i] = -1;
18 }
19
20 void insert() {
21     scanf("%s", S);
22     int len = strlen(S);
23     int now = root;
24     for (int i = 0; i < len; i++) {
25         if (node[now][S[i] - 'a'] == -1) {
26             node[now][S[i] - 'a'] = ++nnode;
27             for (int j = 0; j < 26; j++)
28                 node[nnode][j] = -1;
29         }
30         now = node[now][S[i] - 'a'];
31     }
32     // mark that a word ended at this node.
33     isWord[now] = 1;
34 }

```

৪.৭ আটকুলেশন ভাটের্স (Articulation vertex)  
আটকুলেশন বাহু (Articulation edge)

একটি আনডিরেন্টেড গ্রাফ (undirected graph) এ যদি কোনো নোডকে মুছে ক্ষেত্র গ্রাফটি ডিসকানেক্টেড (disconnected) হয়ে যায় তাহলে তাকে আর্টিকুলেশন ভাট (articulation vertex) বলে। একইভাবে যদি কোনো বাহুকে মুছে ফেললে গ্রাফ ডিসকানেক্টেড হয়ে যায় তাহলে তাকে আর্টিকুলেশন বাহু (articulation edge) বা আর্টিকুলেশন ব্রিজ (articulation bridge) বলে। কখনও কখনও এদের কাটনোড (cut node) বা কাটেজ (cut edge) ও বলে। DFS ব্যবহার করে খুব সহজেই আর্টিকুলেশন ভাটের বাহু বের করে ফেলা যায়। DFS এর একটি শক্তিশালী প্রয়োগ হলো এটি। এটা করার জন্য আমাদের কয়েকটি জিনিসের সঙ্গে পরিচিত হতে হবে: dfsStartTime, dfsEndTime এবং low. আর্টিকুলেশন ভাটের বাহু বের করতে এদের সবগুলোই যে দরকার তা নয়, কিন্তু এদের ব্যবহার করে আমরা বিশেষ জটিল জটিল সমস্যা সমাধান করে ফেলতে পারি। বিশেষ করে ইনফরমেটিভ অলিম্পিয়াডে এধরনের অনেক সমস্যা দেখা যায়। যতদূর মনে পরে 2006 সালের IOI এ এরকম একটি সমস্যা ছিল। যাই হোক, dfsStartTime ও dfsEndTime খুবই সহজ জিনিস। তুমি dfsTime বলে। একটি ভ্যারিয়েশন রাখবে যার প্রাথমিক মান হবে 0। এর পর তোমরা DFS করার সময় যখনই কোনো একটি নতুন unvisited নোডে আসবে তখনই dfsTime কে এক বাড়াবে আর dfsStartTime এ এবং এর বর্তমান সময় নোট করবে আর কোনো নোডের সব চাইল্ড (child) এর visit শেষ হয়ে গেলে dfsTime এর বর্তমান মান dfsEndTime এ মার্ক করে রাখবে। এটা তো শেল dfsStartTime আর dfsEndTime, low একটু জটিল জিনিস। আমরা তো জানি DFS পুরো গ্রাফে একটি ট্রি এর মতো করে আগায়। মনে কোনো নোডের যদি unvisited অ্যাডজাসেন্ট ভাটের থাকে তাহলে আমরা সেটা visit করি (নিচে নামি) আর যদি কোনো unvisited অ্যাডজাসেন্ট ভাটের না থাকে তাহলে ফেরত যাই (প্যারেন্ট (parent) এ ফেরত যাই)। যদি সেই নোড থেকে কোনো visited নোডে যাওয়া যায় তা অবশ্যই এর অ্যানসেস্টর (ancestor) হবে অর্থাৎ ওই নোড থেকে রুটের পাথের মধ্যেই থাকবে অথবা তার নিজের সাবট্রি (subtree) তে থাকবে। কেন? চলো চিঠি করে দেখা যাক। মনে কর �DFS করার সময় আমরা A হতে ইতোমধ্যেই visited হওয়া একটি নোড B তে বাহু পেয়েছি। ধরা যাক B উপরে বলা দুর্বলনের মাঝে কোনোটিই পরে না। তাহলে B কী ধরনের নোড? এটি হবে A এর কোনো আনসেস্টর (ধরা যাক C) এর মেই সাবট্রি তে A আছে সেটি বাদে অন্য যেসব সাবট্রি আছে তাদের কোনো একটির অর্থ। যদি তাই হয় তাহলে C থেকে যখন B কে visit করা হয়েছে তখন সেইসব থেকে আমরা A তেও আসতাম। অর্থাৎ A আর B দুজনই C এর একই সাবট্রি থাকাৰ কথা ছিল। যেহেতু তা না, তাই আমরা বলতে পারি B হয় A এর অ্যানসেস্টর অথবা A এর সাবট্রি এবং অর্থ।

`low[u]` হলো  $u$  নোড বা  $u$  এর সাবট্রিমে থাকা নোডগুলো থেকে সবচেয়ে উপরে (ক্লটের কাছে) নিচে নোডে যাওয়া যায় তার `dfsStartTime`.

$\text{low}[u]$  বের করার জন্য যা করতে হবে তা হলো: প্রথমে একে  $\text{dfsStartTime}$  দিয়ে ইনিশিয়ালাইজেশন করতে হবে। এর পর একে একে এর সব অ্যাডজাসেন্ট নোডগুলোকে দেখতে হবে। ধরা যাক  $v$  হলো  $u$  এর অ্যাডজাসেন্ট কোনো ভার্টের। যদি  $v$  ইতোমধ্যেই  $\text{visited}$  হয়ে যাব তাহলে  $h[v]$  হবে  $v$  এর সবটি এর কেট বা  $\text{প্যারেন্ট}$  অথবা আমনসেন্টর। যদি আমনসেন্টের হয় কিন্তু  $\text{প্যারেন্ট}$  না হয় তাহলে তার  $\text{dfsStartTime}$  দিয়ে  $\text{low}[u]$  কে আপডেট করতে হবে। আর যদি  $\text{প্যারেন্ট}$  হয় এবং তখন যদি আর্টিকুলেশন বাহু বের করতে চাও তাহলে একটু সতর্ক হতে হবে।  $\text{প্যারেন্ট}$  থেকে যেই বাহু দিয়ে আমরা  $u$  তে এসেছি যদি সেই বাহু হয় এটি তাহলে আমরা কোনো কিছু করব না, আর যদি এটি তিনি বাহু হয় তাহলে আগের মতো আপডেট করতে হবে। যদি আমাদের গ্রাফ মাল্টিএজ গ্রাফ না হয় তাহলে আমাদের এটা নিয়ে ভাবার কিছু নেই। এখন যদি আমাদের  $v$  নোডটি আগে থেকে  $\text{visited}$  না হয় তাহলে তার DFS করতে হবে রিকর্সিভ উপায়ে এবং  $\text{low}[v]$  দিয়ে  $\text{low}[u]$  কে আপডেট করতে হবে। এখনে আপডেট করা মানে হলো সর্বনিম্ন মাল্টি বের করা। এই  $\text{low}[v]$  ভালো কিন্তু কোনো একটি নোডের  $\text{dfsStartTime}$ , আমাদের এই মান ব্যত কর হবে তাই সেই নোড রুটের কাঞ্চাকি হবে।

এখন আমাদের সব দরকারি মান বের করা হয়ে গেছে। এই মানগুলো দেখে আমরা বলতে  
পারব কেনন্তা আর্টিকুলেশন ভার্টেক্সের আর কেনন্তা আর্টিকুলেশন বাহু। আর্টিকুলেশন বাহু বের করা  
ভুলন্যুক্ত সহজ। প্রতিটি কাছ নাও, মনে করা যাক  $u - v$ । যদি দেখো  $low[u] < low[v]$  তাহলে  
এটি আর্টিকুলেশন বাহু। কাছ  $v$  এর সাথে হতে কিনো back edge নেই যা  $u$  বা  $v$  এর উপরের  
সঙ্গে চুক্তি। অর্থাৎ  $u - v$  বাহু কেটে দিলে তারা স্কলেনেটেড হয়ে যাবে।  $low[u] = low[v]$

যদি DFS এর শুরুর নোড হয় তাহলে কী হবে? দেখতে হবে যে DFS টি তে এর একের বেশি

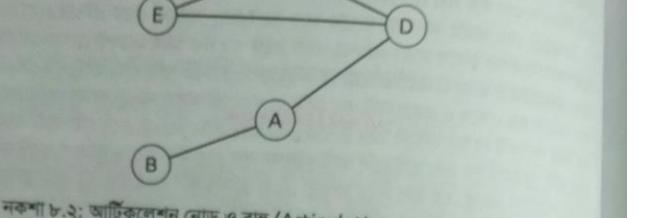
সাবট্রি বা চাইল্ড আছে কিনা। থাকলে  $u$  একটি আর্টিকুলেশন নোড। এখন  $u$  যদি রূট নোড না হয় সেইক্ষেত্রে চিন্তা করা যাক। মনে কর  $u - v$  একটি বাহ। এটি যদি back edge হয় তাহলে  $u$  কর্তৃ কে আবাসন করা যাবে না। ধরা যাক এটি back edge না, অর্থাৎ DFS টি এর বাহ। যদি

আর  $v$  কে আলাদা করা যাবে না। যদ্বা যাক  $\text{A}[0] \text{DFS}(v)$ ।  
 সেখো  $\text{dfsStart}[u] \leq \text{low}[v]$  তার মানে হলো আমরা যদি  $u$  কে ডিলিট করে দেই তাহলে  $v$  ডিসকনেক্ট হয়ে যাবে (খেয়াল আর্থক্ত হবে  $v$  যেন কুণ্ঠ না হয়, চিন্তা করে দেখতে পারো কেন কুণ্ঠের ক্ষেত্রে এটি কাজ করবেন না)। আর্থক্তে আমরা আর্টিকুলেশন ভার্টেক্সে বের করে ফেলতে পারি।  
 একটি উদাহরণ দেওয়া যাক। চিন্তা ৮.২ এ আমরা  $A$  হতে DFS শুরু করব আর আমাদের  
 প্রস্তুতি হবে নিচেরগুলোর নামের জৰুৰে। মানে  $D$  এর অ্যাডজেসেন্ট লিস্ট হবে  $A, C, E,$

অ্যালজোরিদমটি কিন্তু হবে নোডগুলোর মধ্যে একটি সূতরাং এর ভাবে প্রতিটি নোডের  $\text{dfsStartTime}$  কত হবে? প্রথমে আমরা আছি  $A$  তে, সূতরাং এর  $\text{dfsStartTime}$  আর  $\text{low}$  হবে 1. এর পর আমরা যাব  $B$  তে এর  $\text{dfsStartTime}$  ও  $\text{low}$  হবে 2. এর পর আমরা যাব  $D$  তে, এখানে  $\text{dfsStartTime}$  ও  $\text{low}$  হবে 3. এর পর যাওয়া হবে  $C$  2. এর পর  $\text{dfsStartTime}$  ও  $\text{low}$  হবে 4। এর পর  $E$  তে এসে এর  $\text{dfsStartTime}$  ও  $\text{low}$  হবে 5. তবে পরকনেই যখন আমরা  $E - D$  back edge আবিকার করব তখন  $E$  এর  $\text{low}$  কমে  $D$  ।

মাল্টি-এজ (multi edge) শ্বাক মানে দুটি নোডের মাঝে একাধিক বাই থাকতে পারে।

Figure 1. A schematic diagram of the hierarchical structure of the brain. The brain is represented as a tree-like structure with a central node at the top, which branches down into two nodes, each of which further branches into two nodes, and so on. This represents the hierarchical organization of the brain's functional systems.



Start Times (noon, 1pm, 2pm, 3pm, 4pm, 5pm, 6pm, 7pm, 8pm, 9pm, 10pm, 11pm, 12midnight and edge)

low দিয়ে আপডেট করে 3 করে ফেলে।  $D$  বা  $A$  এর low এর কোনো আপডেট হবে না। তাহলে  $\text{dfsStartTime}$  ও  $\text{low}$  হলো  $A(1, 1), B(2, 2), C(4, 3), D(3, 3), E(5, 3)$ । এখন যদি  $A - B$  অর্থাৎ  $A - D$  রাখতুলে দেখো তাহলে রুবেরে এরা আর্টিকুলেশন বাহু কেনে?  $A - B$  এর ক্ষেত্রে  $\text{dfsStart}[A] < \text{low}[B]$  একইভাবে  $A - D$  এর ক্ষেত্রেও তবে, অন্য বাছাতুলের ক্ষেত্রে এই সূর মানে না, যেমন  $C - D$  এর ক্ষেত্রে  $\text{dfsStart}[D] = \text{low}[C]$ । এখন আর্টিকুলেশন নেটওর্কে বের করা যাক।  $A$  যেহেতু রুট আর এর একাধিক সাবট্রি আছে তাই এটি চোখ বজ করে আর্টিকুলেশন নোড। তোমরা যদি  $D - C$  বাহু দেখো তাহলে  $\text{dfsStart}[D] \leq \text{low}[C]$  পাবে তাই  $D$  একটি আর্টিকুলেশন নোড। অন্য কোনো বাহুর ক্ষেত্রে এই শর্ত সত্য না, তাই আর কোনো আর্টিকুলেশন নোড নেই।

গুরুত্বপূর্ণ পৃষ্ঠা) এবং অর্থলাইন সাইকেল (euler cycle)

একটি ফ্রেমে যদি একটি ভাট্টেজ থেকে যায়া ডক করে, প্রতিটি বাহ্যিক একবার করে ঘুরে কোনো একটি ভাট্টেজে যায়া শেষ করা যায় তাহলে তাকে অয়লার পথ বলে। আর যদি ডক ও শেষের ভাট্টেজ একই হয় তাহলে তাকে অয়লার সাইকেল (euler cycle) বা অয়লার সার্কিট (euler circuit) বলে। আর একটি ফ্রেমে অয়লার পথ বা সাইকেল আছে কিনা তা দেখে করা খুবই সহজ। প্রথম শর্ত হলো প্রাক্ষেপ কানেক্টড হত হবে। এখন যদি সামগ্রে নোডের ডিগ্রী (degree) জোড় হয় তাহলে ফ্রেমে অয়লার সাইকেল আছে (অয়লার সাইকেল থাকা মানে কিন্তু অয়লার পথও থাকা, কিন্তু উল্লেখ সত্য নয়)। আর যদি এই ফ্রেমের শুধুমাত্র দুটি নোড বেজেড় ডিগ্রী (odd

卷之三

(degree) ওয়ালা হয় তাহলেও গ্রাফটিতে অয়লার পাথ থাকবে তবে সেক্ষেত্রে আমাদের অবশ্যই ই-ডুটি নোডের কোনো একটি থেকে যাতা শুরু করতে হবে। এরকম হিউয়ার কারণ হলো, শুরু থেকের নোড বাদে বাকি সব নোডের ফ্রেন্টে আমরা কিন্তু একবার চূকলে নেব হতে হয় সুতরাং আমাদের বাছগুলো জোড়ায় জোড়ায় থাকে বা বলতে পারি মাঝের সব ভার্টেক্সগুলোর ডিপ্রী হবে জাড়। এখন যদি সাইকেল হয় তাহলে যেখান থেকে শুরু করেছি সেখানেই শেষ করেছি সুতরাং সব ইন নোডের ডিপ্রীও জোড় হবে। কিন্তু যদি এটি সাইকেল না হয়ে পাথ হয় তাহলে দেখ শুরু আর শেষ ভার্টেক্স আলাদা এবং তাদের ডিপ্রী হবে বিজোড়। এটি তো আমরা প্রমাণ করলাম যে অয়লার পাথ বা সাইকেল হলে এরকম বৈশিষ্ট্য থাকবে। কিন্তু এরকম বৈশিষ্ট্য থাকলেই যে অয়লার পাথ সাইকেল হবে তা কিন্তু প্রমাণ করিনি। সেটি প্রমাণ করাও কিন্তু খুব একটা কঠিন না। তোমরা প্রায়তিক আরোহ বা ইনডাকশন (induction) ব্যবহার করে খুব সহজেই প্রমাণ করতে পারবে।

এখন কেতো দিন যাবত স্টেট পর্যবেক্ষণ করা যাবে? এটাকে শুধু হজার, DFS এর মতো তুমি কোনো একটি ভার্টেক্স থেকে যাত্রা শুরু কর। তারে এখনে আমাদের ভার্টেক্সের জন্য কোনো *visited* থাকবে না, থাকবে বাছুর জন্য। খেয়াল রেখো কোনো একটি বাছুর দুই দিকের কোনো একদিক থেকেই *visit* করা যায়। এখন কোনো একটি ভার্টেক্সে আমরা প্রতিয়ে দেখব যে এর থেকে নের হওয়া কোন কোন বাছ এখনো *visited* হয়নি, যদি এমন কোনো বাছ থাকে তাহলে সেটি নিয়ে বের হয়ে যাব এবং আগের মতোই ঘুরতে থাকব। আর যদি দুই এই ভার্টেক্সের সঙ্গে লাগানো সব বাছই *visited* হয়ে গেছে তাহলে এই ভার্টেক্সে প্রিন্ট করে দেব। খেয়াল কর এই পদ্ধতিতে কিন্তু একটি ভার্টেক্স কিন্তু অনেকবার *visit* হতে পারে।

“সাধাৰণত ইন্ডিয়া মানে তো জানোই? একটি আনড়িরেষ্টেড গ্রাফে একটি নোডের সঙ্গে কঢ়তেলো বাছ লাগানো আছে।  
হলেও মনে নেই।” সুত্রাং আমি একটি ইন্টারনেট ষেঁটুঁটুঁটুঁ যা দেখলাম তা হলো ডিৱেৱেষ্টেড গ্রাফের মতো। আমরা কোনো  
অ্যালগোরিদম পাখ বা সাইকেল বের কৰা প্ৰয়োজনীয় আনড়িরেষ্টেড গ্রাফের মতো। আমরা কোনো  
একটি নোড থেকে শুরু কৰব, এৰ বহিৰ্ভূমী বা আউটগোয়িং (outgoing) কোনো বাছ দিয়ে বেৰ  
হয় যতক্ষণ কোনো না কোনো আউটগোয়িং বাছ (outgoing edge) বাকি থাকে। যখনই শেষ  
হয়ে যাবে আমরা প্ৰিন্ট কৰে দিব এবং আগেৰ নোডে ফিৰে যাব, ঠিক আগেৰ DFS এৰ মতো।  
আশা কৰি বুৰুবুৰে পাৰছ যে আমাদেৰ অ্যালগোরিদম পাখ বা সাইকেল এৰ ঠিক উল্লে ক্ৰমে প্ৰিন্ট হোৱেছ।  
আৱেকটি জিনিস, তা হলো পাখ প্ৰিন্ট কৰার আগে প্ৰতিটি নোডেৰ আউটডিগ্ৰী (outdegree) আৱ  
ইনডিগ্ৰী (indegree)<sup>3</sup> একটি দেখে নিতে হবে। প্ৰতিটি নোডেৰ ইনডিগ্ৰী আৰ আউটডিগ্ৰী সমান  
হতে হবে, অথবা কেবল একটি নোডেৰ ইনডিগ্ৰী, আউটডিগ্ৰী হতে এক বেশি হতে পাৰবে এবং  
একটি নোডেৰ আউটডিগ্ৰী, ইনডিগ্ৰী এৰ থেকে এক বেশি হতে পাৰব। তাৰে তাৰা যথাক্ষেত্ৰে  
পাখেৰ শ্ৰেণি ও শুল্ক হৈব। কিন্তু সবাৰ যদি ইনডিগ্ৰী আৰ আউটডিগ্ৰী সমান হয় তাহে যেকোনো  
নোড থেকে শুৰু কৰে সেখানে ফেৰত আসা যাবে। তবে আগেৰ মতোই কানেক্টেড বাপৰাটি একোৱাৰ

মোডেক কঙ্কলো ডিরেক্টেড বাহ (directed edge) রয়েছে, আর আওতাজো  
কঙ্কলো ডিরেক্টেড বাহ দ্বের হয়েছে।

একটি ডিরেক্ট গ্লাফে নোভগ্লোকে এমনক্রমে সাজাতে হবে যেন, যা

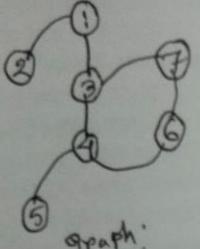
ই ফ্রাফে কোনো ডিরেক্টেড সাইকেল (directed cycle) থাকবে না। সাইকেল থাকলে তো ওই সাইকেল এর নোডগুলোকে তুমি কোনোভাবেই অমনক্রমে সাজাতে পারবে না তাই না? আমরা এই গ্রাফটিতে ব্যবহার করে কোনো ডিরেক্টেড ফ্রাফে সাইকেল আছে কিনা তাও বের করে ফেলতে চাব।

আমরা দূর্ভাবে টপোলজিক্যাল সর্ট (topological sort) করতে পারি। একটি হলো BFS আরেকটি DFS দিয়ে। আমরা কাছে BFS দিয়ে কল্পনামূলক সমস্যা যাবে কী?

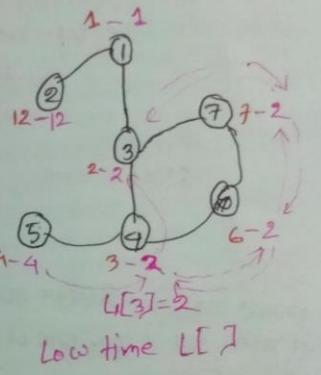
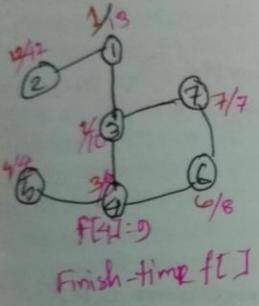
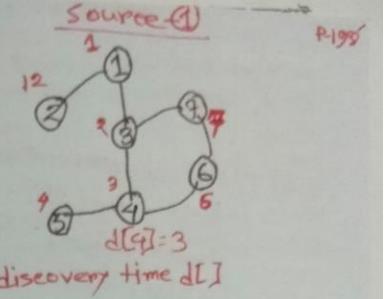
ପ୍ରଥମେ ଆମାଦେର ଏକଟି ଇନଡିଗ୍ରୀ ଏର ଆୟରେ ଲାଗିବେ ସେଥାନେ ପ୍ରତିତି ଲୋଡ଼େର ଇନଡିଗ୍ରୀ ଲେଖା  
କିମ୍ବା କିମ୍ବା କିମ୍ବା କିମ୍ବା କିମ୍ବା

করবে। এখন একটি কিউটে সেসব নোট রাখতে হবে যাদের ইনডিগ্রী শূন্য। এবার কিউ থেকে এক কে নোট তোলার পালা। একটি করে নেওত তুলব আর তার থেকে যেসব বাহু বের হয়ে তাদের দেখে দেখে অন্য প্রাণের নোডের ইনডিগ্রী কমিয়ে দিব। যদি অন্য প্রাণের ইনডিগ্রী হয়ে যায় তাহলে তাকে কিউটে ছুকিয়ে দিব। এভাবে যতক্ষণ না কিউ ফাঁকা হয়ে যায় ততক্ষণ থেকে থাকবে। কিউটে নোট দেই ক্রমে ছুকেছে সেটিই কিউ টপোলজিকাল সর্টের ক্রম। তবে একটি জিনিস, যদি কিউ ফাঁকা হয়ে যাওয়ার পরও দেখ যে কেনো নোডের ইনডিগ্রী এখনো শূন্য ননি তার মানে শ্বাসটিতে সাইকেল আছে। কেন কিউ খুবই খুজলাক একটি আলগরিদম। কেন করে তা খুব সহজেই বোা যায়। যেমন শুরু আমরা ইনডিগ্রী শূন্য ওয়ালা নোডগুলোকে উভে ছুকিয়েছি কারণ এসব নোডকে কাঠো উপর নির্ভর করেন।। এখন এসব নোডকে খুলুন আসলেই যে ফেলব এর মানে এর উপর যারা নির্ভর করে তাদের নির্ভরতা এক করে করে থাবে। যদি তাদের নির্ভরতা শূন্য হয়ে যাব তার মানে আমরা চাইলে এই নোডকে এখন নিতে পারব, সেজন্য আমরা কে কিউটে ছুকাই। সহজ না? ~

জাল্টের একটি যারে, আর *visited* অবেকচিটি যারে যাই বা *কি* হলো আবশ্যিক পদ্ধতি। যদি এখন ক্ষমা দাওয়া এবং এরা বাকি ১ হলো আবশ্যিক পদ্ধতি তবে একটি নোড দেখের আর যদি সেই নোডের *visited* এর মান এখনো ২ না হয়ে থাকে তাহলে এটি *DFS* কর করব। *DFS* এর প্রথমেই আমরা যা করব তা হলো *visited* এর মান ১ করে। এর পর এখন থেকে যেই মেই ডিরেক্টেড বাহু বের হয়েছে তাদের দেখব। যদি অনেক প্রাণীটি *visited* এর মান ১ হয়ে থাকে এর মানে হলো আমরা একটি সাইকেল পেষে গেছি সুতরাং এটিকে কেনো টপোলজিক্যাল ক্রম (topological order) নেই। যদি *visited* এর মান ২



graph:



\* If  $d[u] < \text{Low}[v]$  then it is Articulation point.

\* If  $\text{Low}[u] < \text{Low}[v]$  then Edge is Articulation edge.

time  $\leftarrow 0$   
 $d[u] \leftarrow 0$   
 $low[u] \leftarrow 0$

Find Articulation point ( $G_1, u$ ):

time = time + 1;  
 $low[u] = d[u] = time$   
 $vis[u] = 1$ ;

for each edge  $u$  to  $v$  in  $G_1$ , adjacent( $u$ )

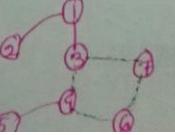
if ( $v \neq \text{parent}[u]$ )  
 Find Articulation point ( $G_1, v$ )  
 $low[u] = \min(\text{low}[v], \text{low}[u])$

$f[u] = time$

+ তুমার একটি আলগরিদম মাঝে এতে যাই হবে না? কিন্তু প্রশ্ন  
 করা প্রতিটি প্রশ্ন "যারে মের প্রথমে V ও দ্বিতীয়ে পথ ইন্ডিকেট কোন তিনিক পথ আছে?

বিষ ব্যবহার করে উই সমস্যাটি সমাধান করা যায়। নথি ১০  
 দোষ ব্যবহার করে মের V ও আমার মন্তব্যটি অন্ত দ্বিতীয় ২৫,  
 দোষ u থেকে মের V ও আমার মন্তব্যটি অন্ত দ্বিতীয় ২৫,  
 সঙ্গে, মন্তব্য প্রতিটি দুটি অবশ্যই কেটে ফেলে রিষ হবে।  
 অবশ্য অন্যান্য দুটি শুরু হবে মাঝে বিল মুলে হেলে  
 আক মুলে আছে পিছ। দোষ যদি প্রথমে V কেই আবমান  
 এ প্রয়োগে আর তারে ভাস্কে পথ অবশ্যই  
 ইচ্ছিক।

{1,2,3}, {4,5,6}, {7,8}



কর না কেন আমরা একে প্রাচে ডিরেক্টেড বাহু দিয়ে প্রকাশ করব। ( $-x \vee y$ ) এর ফলে আমাদের বাহু হবে  $x$  থেকে  $y$  এর দিকে আর  $-y$  থেকে  $-x$  এর দিকে ইমপ্লিকেশন চিহ্নের দিক অনুসূচে।) অন্যভাবে বলতে: যদি  $x = true$  হয় তাহলে  $y = true$  হবে, আর যদি  $-y = true$  হয় তাহলে  $-x = true$  হবে এ জিনিসটা তুম ইমপ্লিকেশন চিহ্ন দিয়ে যেভাবে প্রকাশ করেছো সেভাবে ডিরেক্টেড বাহু দেওয়া হবে। এখন আমরা এভাবে প্রত্যেকটি ক্লজের জন্য দুটি করে বাহু দিব। সব বাহু আঁকা থেকে আমাদের দেখতে হবে যে,  $x$  আর  $-x$  (এখনে  $x$  হলো যেকোনো ভ্যারিয়েবল অর্থাৎ তোমাকে  $n$  টি ভ্যারিয়েবল এর জন্য যাচাই করে দেখতে হবে) এর জন্য  $x$  থেকে  $-x$  এ আর  $-x$  থেকে  $x$  এ দুটিকেই পাখ আছে কিনা। যদি থাকে তাহলে আমাদের 2-sat সমাধান করা যাবে না। কারণ  $x$  হতে  $-x$  এ পাখ থাকা মানে  $x = true$  হলে  $-x = true$  হবে। সুতরাং একেজেনে আমরা বলতে পারি  $x = true$  হতে পারে না,  $x = false$  হবে। কিন্তু যদি অন্যদিকেও পাখ থাকে মানে  $-x$  হতে  $x$  এ তাহলে তো আবার একইভাবে আমরা বলব  $-x = false$  হতেই হবে। কিন্তু  $x$  আর  $-x$  তো একই সঙ্গে  $false$  হতে পারে না তাই না? এজন্য যদি  $x$  আর  $-x$  একটি থেকে উপরটিতে যাওয়া যায় এর মানে হবে সমাধান নেই। আর যদি এমন কোনো ভ্যারিয়েবল খুজ পাওয়া না যায় তাহলে সমাধান করা যাবে। আমরা দুটি নোড থেকে একে অপরের দিকে যাওয়া যায় কিনা কীভাবে সহজে বের করতে পারি? SCC! যদি আমাদের ঘাফকে SCC তে ভাঙ্গ পরে দেখি  $x$  ও  $-x$  একই ক্লজেনেটে তাহলে বুঝব একে উপরের দিকে যাওয়া যায়, আর না হলে যাবে না।

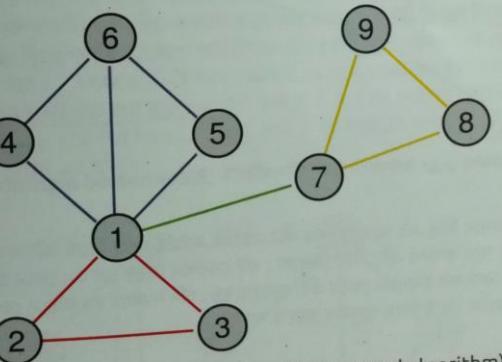
এতক্ষণ আমরা বের করলাম কীভাবে 2-sat সমাধান যোগ্য কিনা তা বের করা যায়। কিন্তু এর সমাধান কীভাবে বের করা যায় (অর্থাৎ কোন ভ্যারিয়েবল  $false$  আর কোন ভ্যারিয়েবল  $true$ )? এর উপায় হলো, তুম SCC এর প্রত্যেক ক্লজেনেটকে কন্ট্রাক্ট (contract) বা সংকৃতিত করে একটি নোড বানাও। এই পরিবর্তিত গ্রাফ অবশ্যই একটি DAG হবে (DAG = Directed Acyclic Graph) অর্থাৎ এই ডিরেক্টেড গ্রাফে কোনো সাইকেল নেই। যেহেতু কোনো সাইকেল নেই তাই এর টপোলজিক্যাল ক্রম আছে। আমাদের যা করতে হবে, এই অর্ডারের শেষ থেকে আসতে হবে এর প্রত্যেক ক্লজেনেটকে true দেওয়ার চেষ্টা করতে হবে (কোনো ক্লজেনেটকে true দেবার মানে হলো এতে থাকা সব নোড true)। যদি দেখ তোমার এই ক্লজেনেটে এমন একটি নোড আছে যার মান আগে থেকেই বাসনে করা হয়ে গেছে আর তোমার এই এখন true করতে চাওয়া মানের সঙ্গে মিলছে না তাহলে  $false$  দিবে। তুম চাইলে চিন্তা করে দেখতে পার বা প্রমাণও করতে পার কেন এই গ্রাফ (greedy) পদ্ধতিটি ঠিকভাবে মান বাসাচ্ছে।

## ৮.১২ বাইকানেটেড কম্পোনেন্ট (Biconnected component)

সত্যি কথা বলতে আমি এই অ্যালগরিদম নিজে থেকে কখনও ইমপ্লিমেন্ট করিনি। আমার কাছে বেশ কঠিন লাগে বা বলতে পার সময়সাপেক্ষ লাগে। বাইকানেটেড গ্রাফ (Biconnected graph) মানে হলো সেই প্রাফের কোনো ভ্যারিয়েবল থেকে আমরা মুছে ফেলি তাহলে সেই প্রাফ ডিস্কানেটেড হবে না। যেমন ধৰা যাক আমাদের তিন নোডের একটি প্রাফ আছে এবং এদের

১০৫

মাঝের বাহু গুলো হলো 1-2, 2-3, 1-3। এটি কিন্তু বাইকানেটেড গ্রাফ না কারণ এই প্রাফ থেকে আমরা যদি 2 মুছে ফেলি তাহলে বাকি নোড দুটি ডিস্কানেটেড হয়ে যায়। কিন্তু এই গ্রাফটিতে যদি আরও একটি বাহু 1-3 থাকত তাহলে কিন্তু এটি বাইকানেটেড গ্রাফ হতো। আমরা যেকোনো গ্রাফকে কিছু সংখ্যক বাইকানেটেড সাবগ্রাফ (biconnected subgraph) বা বাইকানেটেড কম্পোনেন্ট (biconnected component) এ ভাগ করতে পারি যেন সব বাহু কোনো না কোনো ভাগে পরে। খেয়াল কর, একটি বাহু একটি BCC হতে পারে কারণ এর এক মাথার নোড মুছে ফেললে কিন্তু কেউ ডিস্কানেটেড হয় না। আমরা চিত্র ৮.৩ তে একটি গ্রাফকে BCC তে ভাড়িয়ে দেখালাম। প্রতিটি রঙ একেকটি কম্পোনেন্ট। এখানে খেয়াল কর একটি নোড কিন্তু একাধিক কম্পোনেন্টের অংশ হতে পারে। কেন? কারণ একই নোড প্রতিটি বাহু একটি ক্লজেনেটেড হয়ে থাকে। আবার নোডগুলো কানেক্টেড থাকে। আবার লাল অংশেও একই কথা সত্য। তবে তুম যদি 1 নোডটি মুছে ফেলি তাহলে বাকি নোডগুলো কানেক্টেড থাকে। আবার লাল অংশেও একই কথা সত্য। তবে কিন্তু যদি 2 মুছে ফেললে গ্রাফটি ডিস্কানেটেড হয়ে যেত। এর মানে একটি নোড একাধিক BCC এর অংশ হতে পারে কিন্তু একটি বাহু কেবলমাত্র একটি BCC এরই অংশ। আশা করি এটি বলার অপেক্ষা রাখে না যে আমরা প্রতিটি কম্পোনেন্টকে যত বড় করা সম্ভব তত বড় করতে চেষ্টা করি। তোমরা চিত্র ৮.৩ এ যদি বলতে প্রতিটি বাহু আলাদা আলাদা কম্পোনেন্ট, হাঁ কথা ঠিক কিন্তু এই যে বললাম প্রতিটি কম্পোনেন্টকে আমরা বড় করার চেষ্টা করি, সে জন্য আমাদের BCC হবে চিরের মতো।

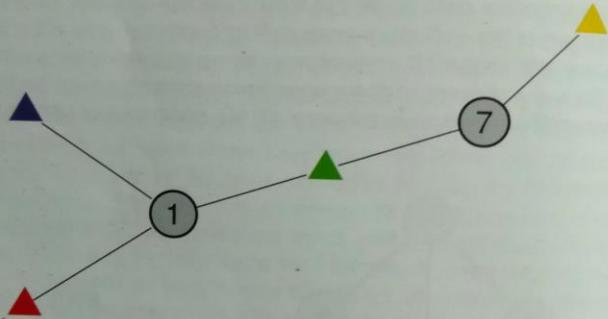


নকশা ৮.৩: বাইকানেটেড অ্যালগরিদম (Biconnected algorithm)

অনেক সময় সমস্যা ভেদে তোমাকে হয়তো গ্রাফকে কম্পোনেন্টে ভাগ করতে হয় যেন একই কম্পোনেন্টের কোনো বাহু মুছে ফেললে গ্রাফটি ডিস্কানেটেড না হয়ে যায়। সেকেতে আমাদের এত কষ্ট করতে হবে না, শুধু আর্টিকুলেশন রিপ বা বাহু বের করে একটি BFS বা DFS চালিয়ে

দিলেই আমাদের সব কম্পোনেন্ট বের হয়ে যাবে। আমরা প্রতিটি unvisited নোডের জন্য BFS বা DFS চালাব এবং আর্টিকুলেশন রিজ ব্যবহীত অন্য সব বাছ দিয়ে আমরা ট্রাইভার্সেল করব।

BCC এর সঙ্গে সম্পর্কিত আরেকটি জিনিস আছে আর তা হলো ব্লক কার ভাট্টেটি (Block cut vertex tree). প্রতিটি আনডিরেস্টেড গ্রাফকে যেমন আমরা BCC তে ভাগ করতে পরি ঠিক তেমনই সেই BCC কে আমরা একটি ট্রি আকারে সাজাতে পারি যেখানে ট্রি এর নোডগুলো হলো BCC এর কাট ভার্টেক্স (আর্টিকুলেশন নোড) সমূহ এবং ব্লক (block) বা কম্পোনেন্টগুলো। যদি কোনো একটি কাট ভার্টেক্স একটি ব্লকের অংশ হয় তাহলে তাদের মধ্যে বাছ থাকবে। তাহলে যেকোনো কানেক্টেড আনডিরেস্টেড গ্রাফ (connected undirected graph) এর জন্য আমরা একটি ট্রি পাব। যেমন চিত্র ৮.৩ এর ব্লক কাট ভার্টেক্স ট্রি হবে চিত্র ৮.৪ এর মতো। বেশ কিছু সমস্যায় আমরা দেখব যে এই ট্রি বেশ কাজে লাগে।



নকশা ৮.৪: বাইকানেক্টেড অ্যালগরিদম (Biconnected algorithm)

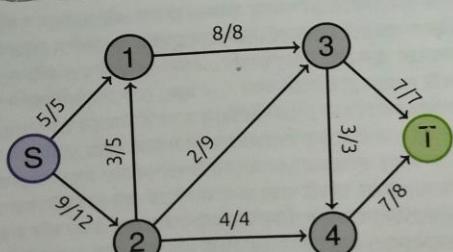
সবই বুবলাম কিন্তু এর অ্যালগরিদম কী? আগেই বলেছি আমি নিজে এটা কখনও কোড করি নাই। হয়তো পরে কখনও এটা নিয়ে লিখবো। এই সেকশন থেকে তোমরা জেনে রাখলে BCC কী জিনিস এবং কোন সব সমস্যার ক্ষেত্রে এটা ব্যবহার হয়। যদি দরকার হয় তাহলে তোমরা ইন্টারনেট থেকে এর কোনো কোড নিয়ে ব্যবহার করতে পারো।

## ৮.১৩ ফ্লো (Flow) সম্পর্কিত অ্যালগরিদম

নিঃসন্দেহে ফ্লো (flow) একটি কঠিন টপিক। এর কোড বেশ সহজ কিন্তু এটি ঠিক মতো বোঝা বা একে রঙ করা খুবই কঠিন। দেখা যাক তোমাদের এর মূল ধারনাটি বোঝাতে পারি কিনা।

### ৮.১৩.১ ম্যাক্সিমাম ফ্লো (Maximum flow)

এই সমস্যায় কিছু নোড এবং কিছু ডিরেস্টেড বাছ থাকবে। নোডসমূহের মাঝে দুটি বিশেষ নোড থাকবে। একটি হলো উৎস বা সোর্স (source) যা সাধারণত আমরা  $S$  দিয়ে প্রকাশ করি আর আরেকটি হলো গন্তব্য বা সিঙ্ক (sink) যা আমরা সাধারণত  $T$  দিয়ে প্রকাশ করি। ডিরেস্টেড বাছসমূহ এর সঙ্গে একটি সংখ্যা থাকবে, একে আমরা ওজন (weight) বলব না, একে আমরা বলব ধারনক্ষমতা বা ক্যাপাসিটি (capacity). এখন মনে কর সোর্সে অপরিসীম তরল পদার্থ আছে, আর সিঙ্কে যেকোনো সময়ে অপরিসীম তরল প্রবেশ করতে পারে। অন্যান্য যেসব বাহুর কথা বললাম তাদের একেকটি পাইপ মনে কর যাদের ক্যাপাসিটি দেওয়া আছে অর্থাৎ কোনো একক সময়ে সর্বোচ্চ কত তরল ওই পাইপ দিয়ে প্রবাহিত হতে পারে তা দেওয়া আছে। তোমাদের বলতে হবে কোনো একক সময়ে কী পরিমাণ তরল সোর্স হতে সিঙ্কে প্রবাহিত হতে পারে? তোমরা ধরে নিতে পার যে, কোনো পাইপ দিয়ে তরল যেতে কোনো সময় লাগে না তবে একক সময়ে তার ক্যাপাসিটির থেকে বেশি তরল প্রবাহিত হবে না হয়। এই সমস্যাটি হলো ম্যাক্সিমাম ফ্লো (maximum flow) কিছু উদাহরণ দেখা যাক। ধরা যাক,  $S$  হতে  $A$  তে একটি বাছ আছে যার ক্যাপাসিটি 10, আর  $A$  হতে  $T$  তে একটি বাছ আছে যার ক্যাপাসিটি 20। তাহলে এই গ্রাফে ম্যাক্সফ্লো হবে 10। যদি আগের গ্রাফে ক্যাপাসিটি ঠিক উল্টো হতো তাহলেও কিন্তু ম্যাক্সফ্লো এর থেকে বেশি তরল প্রবাহিত হবে না হয়। এই সমস্যাটি হলো ম্যাক্সিমাম ফ্লো (maximum flow) সর্বোচ্চ করাই ম্যাক্সফ্লো এর লক্ষ্য।



নকশা ৮.৫: ম্যাক্সিমাম ফ্লো (Maximum flow)

- পরিবৃত্তের ব্যাসার্ধ বের কর (radius of circumcircle)
- অন্তর্বৃত্তের ব্যাসার্ধ বের কর (radius of innercircle)
- তিনটি মধ্যমা (median) এর দৈর্ঘ্য বের কর
- তিনটি উচ্চতা (height) বের কর
- তিনটি কোণ সরাসরি বের না করে তুমি কি বলতে পারবে কোনো ত্রিভুজ সূক্ষ্মকোণ (acute), সমকোণী (right) বা স্ফূলকোণী (obtuse) কিন? এটি প্রাণী দরকার হয়। কারণ আমরা যতটা সন্তুষ্ট ফ্লোটিং পয়েন্ট সংখ্যার হিসাবনিকাশ কর করার চেষ্টা করি। যদি ত্রিভুজের তিন বাহু পূর্ণ সংখ্যায় দেওয়া থাকে তাহলে আমরা ফ্লোটিং পয়েন্ট সংখ্যার হিসাবনিকাশ না করে বলে দিতে পারি ত্রিভুজটি সূক্ষ্মকোণী/সমকোণী/স্ফূলকোণ কিন। বা আসলে কোনো একটি কোণ সূক্ষ্মকোণ/সমকোণ/স্ফূলকোণ কিন। উপায় হলো, আমরা জানি পিথাগোরাসের সূর্য হলো  $a^2 = b^2 + c^2$  যেখানে  $a$  হলো সমকোণের বিপরীত বাহু বা অতিভুজ। এবন তুমি যদি = এর পরিবর্তে  $<$  বা  $>$  বসাও তাহলেই তুমি বলতে পারবে যে তাৰ সূক্ষ্মকোণ/স্ফূলকোণ কিন।

∴ ধর একটি  $r$  ব্যাসার্ধের বৃত্ত আছে। এখন এর কেন্দ্র হতে  $d$  দূরত্ব দূরে একটি সরলরেখা টেনে বৃত্তের একটি অংশ কেটে ফেলে দেওয়া হলো। বলতে হবে বাকি অংশের ক্ষেত্রফল কত? বাকি অংশের পরিধিই বা কত? এটি যদি বৃত্ত না হয়ে একটি গোলক (sphere) হতে তাহলে সুতৰঙ্গো কেমন হতো?

## ১০.২ স্থানাঙ্কভিত্তিক জ্যামিতি (Coordinate Geometry) এবং ভেক্টর (Vector)

আমরা আগের সেকশনে জ্যামিতির খুবই মৌলিক কিছু হিসাবনিকাশ দেখলাম। এখন আমরা কিছু স্থানাঙ্কভিত্তিক জ্যামিতি (coordinate geometry) আর ভেক্টর (vector) দেখব। আমরা হয়তো এক ফাঁকে জটিল সংখ্যা (complex number) ব্যবহার করে কীভাবে ভেক্টর এবং স্থানাঙ্কভিত্তিক জ্যামিতির কিছু কিছু হিসাবনিকাশ আরও সহজে করা যায় তা ও দেখব। তোমরা যদি উচ্চ মাধ্যমিকের বই দেখে স্থানাঙ্কভিত্তিক জ্যামিতি আর ভেক্টর নিয়ে একটু জ্ঞান নিয়ে ফেল। আমরা এখানে প্রোগ্রামিং প্রতিযোগিতার দিক থেকে এই দুটি জিনিস দেখবো।

বিমাত্রিক বা  $2 - dimensional (2D)$  স্থানাঙ্কভিত্তিক জ্যামিতিতে আমরা একটি বিন্দুক যে  $(x, y)$  দিয়ে প্রকাশ করি। অর্থাৎ আমরা একটি স্ট্রাকচার (structure) ব্যবহার করে দু'ব সহজেই বিন্দু (point) বানিয়ে ফেলতে পারি। একইভাবে সেই একই স্ট্রাকচার ব্যবহার করে আমরা ১D ভেক্টর এর কাজও করে ফেলতে পারি। সুতরাং দেখা যায় যে, point structure এর জন্ম দেখ

(addition), বিয়োগ (subtraction), ক্লেইর / ডট গুণন (scalar/dot product), ক্রস গুণন (cross product) ইত্যাদি ফাংশন বা অপারেটর ওভারলোডিং (operator overloading) করার প্রয়োজন হয়।

রেখাকে উপস্থাপন করার আরেকটা উপায় হলো "parametric representation"। এবং এটি বেশ কাজের। ধর তোমাকে দুটি বিন্দু  $A(x_a, y_a)$  এবং  $B(x_b, y_b)$  দেওয়া আছে। তুমি এদের ভেতর দিয়ে যায় এরকম একটি সরলরেখার প্যারামিট্রিক রূপে উপস্থাপন (parametric representation) চাও। এটি হবে:  $A + t(B - A)$ । এখানে  $B - A$  কে একটি ভেতরের মতো ভাবতে পার। একটি উন্নাহরণ দেওয়া যাব, মনে কর  $A(1, 1)$  আর  $B(4, 6)$  এর ভেতর দিয়ে যায় এরকম একটি রেখার প্যারামিট্রিক রূপ বের করতে চাই। এটি হবে  $(1, 1) + t[(4, 6) - (1, 1)]$  বা  $(1, 1) + t(3, 5)$  বা  $(1 + 3t, 1 + 5t)$ । একটু অভিজ্ঞ তাই না? সমস্যা নাই, এই অনন্তেরের কিছু সুবিধা দেখাক। এখনে তোমার ক্লেইটিং প্যেস্ট সংখ্যার দরকার হয় না- যদি  $A$  এবং  $B$  দুটি পূর্ণ সংখ্যার হ্যানকড ওয়ালা বিন্দু হয়। আবার খেয়াল কর  $t$  এর মান যদি  $[0, 1]$  এ সীমাবদ্ধ হয় তাহলে এটি একটি রেখাখণ্ড উপস্থাপন করে। যদি  $[0, \infty)$  এ সীমাবদ্ধ হয় তাহলে এটি হবে রশ্মি, আবার যদি পুরো  $t$  রেখাকে উপস্থাপন করতে চাও তাহলে হবে  $[-\infty, \infty)$ । জিনিসটি এমন যে তুমি  $A$  থেকে ভর্ত করে রেখাকে উপস্থাপন করতে চাও তাহলে তিলে তিলে যাবে।  $t = 0$  এর মানে তুমি  $A$  তেই থাকবে (যুক্তে  $t = 0$  বসিয়েই দেখে  $B$  এর দিকে তিলে তিলে যাবে)।  $t = 1$  এর মানে তুমি  $B$  তে, এর মধ্যের মান মানে তুমি  $A$  আর  $B$  এর মধ্যে আছ। এমনবি-