Team: Zero^Infinity(DUET)

```
ll a[limit][limit];     /// Common for all BIT
ll BIT[limit][limit];
```

******** 2D BIT :**

```
ll n,m,q;
void update(ll xx,ll yy,ll val){

   for(ll x=xx;x<=n;x+=(x&-x)){
      for(ll y=yy;y<=m;y+=(y&-y)){

         BIT[x][y]+=val;
      }
   }
}

ll query(ll xx,ll yy){
   ll sum=0;
   for(ll x=xx;x>0;x-=(x&-x)){

      for(ll y=yy;y>0;y-=(y&-y)){
         sum+=BIT[x][y];
      }
   }
   return sum;
}

int main(){
   cin>>n>>m>>q;
   for(int i=1;i<=n;i++){
      for(int j=1;j<=m;j++){
         cin>>a[i][j];
         update(i,j,a[i][j]);
      }
   }
   while(q--){
      string s; cin>>s;

      if(s=="sum"){  ///summation of a range

         ll x1,y1,x2,y2;
         cin>>x1>>y1>>x2>>y2;
         ll in1=query(x2,y2);
         ll ex1=query(x2,y1-1);
         ll ex2=query(x1-1,y2);
         ll in2=query(x1-1,y1-1);
         cout<<in1+in2-ex1-ex2<<endl;
      }
      else{
         ll x,y,val;
         cin>>x>>y>>val;
         a[x][y]+=val;   ///adding value
         update(x,y,val);
      }
   }
   return 0;
}
```

****** BIT Implemetation**

```
void update(int index,int val,int n){
   while(index <= n){
      BIT[index] += val;
      index += (index & -index);
   }
}

int answer(int index){
   int sum = 0;
   while(index > 0){
      sum += BIT[index];
      index -= (index & -index);
   }
   return sum;
}

int main(){
   int n,q;   cin>> n >> q;
   for(int i=1;i<=n;i++){
      cin>>a[i];
      update(i,a[i],n);
   }
   while(q--){
      int l,r;
      cin>>l>>r;   /// sum in range
      cout<<answer(r)-answer(l-1)<<endl;
   }
   return 0;
}
```

***** Binary Lifting in BIT**

/// In this problem we need to find minimum Index where query wanted sum bigger.

```
void update(int index,int val,int n){
   while(index <= n){
      BIT[index] += val;
      index += (index & -index);
   }
}

int answer(int index){
   int sum = 0;
   while(index > 0){
      sum += BIT[index];
      index -= (index & -index);
   }
   return sum;
}

int main(){
   int n,q;   cin>>n>>q;
   for(int i=1;i<=n;i++){
      cin>>a[i];
      update(i,a[i],n);
   }
```

Team: Zero^Infinity(DUET)

```
while(q--){
    string s;   cin>>s;
    if(s=="add"){     /// Adding something to this position
      ll pos,add;   cin>> pos >> add;
      a[pos] += add;
      update(pos , add , n);
    }

    else if(s=="assign"){   /// Assign value to this position
      ll pos,val;   cin>>pos >> val;
      update( pos, -a[pos] , n); ///clear first
      update(pos , val , n ); /// assign now
      a[pos]=val;
    }
    else {
      ll qsum;    cin>>qsum;
      ll cur_jump=0 , cursum=0;

      for(int jump=20; jump >= 0 ;jump--){

        ll next_jump = cur_jump + (1 << jump);

        if(next_jump <= n && (cursum +
                              BIT[next_jump]) < qsum){
          cur_jump = next_jump;
          cursum += BIT[next_jump];
        }
      }
      cout<<cur_jump<<endl;
    }
  }
  return 0;
}
```

***** Segment Tree lazy propagation 1**

```
int a[limit];
int st[limit << 2] , lazy[limit << 2];

void segment_tree(int node,int b,int e){
        if(b==e){
                st[node] = a[b];   return;
        }
  int mid=(b+e)/2;
  segment_tree(node << 1, b, mid);
  segment_tree(node << 1 | 1 , mid+1, e);
  st[node] = st[node << 1] + st[node << 1 | 1];
}
void update(int node,int b,int e,int l,int r,int val){
        if(lazy[node] != 0){
                int Update=lazy[node];
                lazy[node]=0;
                st[node] += Update * (e-b+1);
                if(b!=e){
                  lazy[node << 1] += Update;
                  lazy[node << 1 | 1] += Update;
                }
        }
        if(e<l || b>r) return;
        if(b>=l && e<=r){

                int Update = (e-b+1)*val;
                st[node] += Update;
                if(b!=e) {
                  lazy[node << 1] += val;
                  lazy[node << 1 | 1] += val;
                }
                return;
        }
  int mid=(b+e)/2;
  update(node << 1, b, mid, l, r, val);
  update(node << 1 | 1, mid+1, e, l, r, val);
  st[node]=st[node << 1]+st[node << 1 | 1];
}

int query(int node,int b,int e,int l,int r){ /// summation
query
        if(lazy[node]!=0){
                int Update = lazy[node];
                lazy[node]=0;
                st[node]+=Update * (e-b+1);
                if(b!=e){
                  lazy[node << 1] += Update;
                  lazy[node << 1 | 1] += Update;
                }
        }
        if(e<l || b>r) return 0;
        if(b>=l && e<=r) return st[node];
        int mid=(b+e)/2;
        return query(node << 1, b, mid, l, r) + query(node <<
                        1 | 1, mid+1, e, l, r);
}
```

***** Segment Tree lazy propagation 2**

```
int a[limit];
struct Node{
  int prop;  nt sum;
}st[limit << 2];
void segment_tree(int node,int b,int e){
  if(b==e){
    st[node].sum = a[b];
    st[node].prop=0;
    return;
  }
  int mid=(b+e)/2;
  segment_tree(node << 1 , b, mid);
  segment_tree(node << 1 | 1, mid+1, e);
  st[node].sum=st[node << 1].sum + st[node << 1 | 1].sum;
  st[node].prop=0;
}
void update(int node,int b,int e,int l,int r,int val){
  if(e<l || b>r) return;
  if(b>=l && e<=r){
    st[node].sum += ((e-b+1)*val);
    st[node].prop += val;
    return;
  }
```

```
   int mid=(b+e)/2;
   update(node << 1, b, mid, l, r, val);
   update(node << 1 | 1, mid+1, e, l, r, val);
   st[node].sum = st[node << 1].sum + st[node << 1 | 1].sum
                                + (e-b+1 )*st[node].prop;
}
int query(int node,int b,int e,int l,int r,int carry){

   if(e<l || b>r) return 0;
   if(b>=l && e<=r){
      return st[node].sum + carry*(e-b+1);
   }
   int mid=(b+e)/2;
   int q1 = query(node << 1 , b, mid, l, r,
         carry+st[node].prop);

   int q2 = query(node << 1 | 1, mid+1, b, l, r,
         carry+st[node].prop);
   return q1+q2;
}
```

***** Max subarray sum in range + index update**

```
/// update k'th pos with v
/// find the maximum sum sub array from L to R

struct Node{
   ll tsum;   ll pref;  ll suff;   ll maxsub;
   Node(){
      tsum = maxsub = pref = suff =-1e16;
   }
}st[limit << 2];
ll a[limit];

Node Marge(Node left,Node right){
   Node parentnode;
   parentnode.tsum=left.tsum+right.tsum;
   parentnode.pref=max(left.pref, left.tsum + right.pref);
   parentnode.suff=max(right.suff,right.tsum+left.suff);

parentnode.maxsub=max(max(left.maxsub,right.maxsub),
                                    left.suff+right.pref);
   return parentnode;
}
void built_tree(int node,int b,int e){
   if(b==e){
      st[node].tsum= st[node].pref= st[node].suff=
                                st[node].maxsub= a[b];
      return;
   }
   int mid=(b+e)/2;
   built_tree(node << 1,b,mid);
   built_tree(node << 1 | 1,mid+1,e);
   st[node] = Marge(st[node << 1] ,st[node << 1 | 1]);
}
void update(int node,int b,int e,int pos,int val){
   if(b==e){
      st[node].tsum = st[node].pref = st[node].suff =
                          st[node].maxsub = a[b] = val;
      return;
   }

   int mid=(b+e)/2;
   if(pos<=mid)
      update(node << 1,b,mid,pos,val);
   else
      update(node << 1 | 1,mid+1,e,pos,val);
   st[node] = Marge(st[node << 1],st[node << 1 |1]);
}
Node query(int node,int b,int e,int l,int r){
   if(e<l || b>r) {
      Node emptynode;
      return emptynode;
   }
   if(b>=l && e<=r){
      return st[node];
   }
   int mid=(b+e)/2;
   Node left=query(node << 1,b,mid,l,r);
   Node right=query(node << 1 | 1,mid+1,e,l,r);
   Node ans=Marge(left,right);
   return ans;

}
```

output …. query(1,1,n,l,r).maxsub;

****** Maximum prefix in range + index update**

```
/// Find maximum prefix in range between [a, b] if less than
        0 the 0 is the answer
/// update the k th index with v

struct Node{
   ll sum;  ll pref_sum;
   Node(){
      sum = pref_sum =0;
   }
}st[limit << 2];
ll aa[limit];

void built_tree(int node,int b,int e){
   if(b==e){
      st[node].sum = aa[b];
      st[node].pref_sum = aa[b];
      return;
   }
   int mid =(b+ e) / 2;
   built_tree(node << 1 , b ,mid);
   built_tree(node << 1 | 1, mid+1 , e);
   st[node].sum  = st[node << 1 ].sum +
                              st[node << 1 |  1].sum;
   st[node].pref_sum = max(st[node << 1 ].pref_sum ,
        st[node << 1 ].sum + st[node << 1 | 1].pref_sum);
}
void update(int node,int b,int e,int id){
   if(b==e){
      st[node].sum = aa[b];
      st[node].pref_sum = aa[b];
      return;
   }
```

Team: Zero^Infinity(DUET)

```
    int mid =(b + e) / 2;
    if(id <= mid) update(node << 1 , b ,mid,id);
    else update(node << 1 | 1 , mid+1 , e,id);
    st[node].sum = st[node << 1 ].sum +
                                    st[node << 1 | 1].sum;

    st[node].pref_sum = max(st[node << 1 ].pref_sum ,
          st[node << 1 ].sum + st[node << 1 | 1].pref_sum);
}
Node query(int node,int b,int e,int l,int r){
    if(b>r || e<l){
        Node emptynode;
        return emptynode;
    }
    if(b >= l && e <= r){
        return st[node];
    }
    int mid = (b+e)/2;
    Node left = query(node << 1 , b,mid , l, r);
    Node right = query(node << 1 | 1,mid+1,e, l,r);
    Node res;
    res.sum = left.sum + right.sum;
    res.pref_sum = max(left.pref_sum , left.sum +
                                    right.pref_sum);
    return res;
}
```

output … query(1,1,n,a,b).pref_sum

****** Stars & Bars**

```
ll fact[limit];
ll Mod(ll x){
    return ((x%MOD + MOD)%MOD);
}
ll mulmod(ll a,ll b){
    return Mod(Mod(a)*Mod(b));
}
ll Binary_expo(ll a,ll p){
        ll res=1;
        while(p){
                if(p & 1) res=mulmod(res,a);
                p/=2;
                a=mulmod(a,a);
        }
        return res%MOD;
}
ll nCr(ll n,ll r){
    ll numerator= fact[n];
    ll denominator = mulmod(fact[r],fact[n-r]);
    return mulmod(numerator,
Binary_expo(denominator,MOD-2));
}
int main(){
    cin >> n >> m;  // n = no of people & m = no of items
    fact[0] = 1;
    for(ll i=1;i<limit;i++){
        fact[i]=mulmod(fact[i-1],i);
    }
    cout<< nCr(m+n-1,n-1) <<endl;
}
```

***** Derangement**

/// No of ways such that n distict items arrange that none of them occupies it's orginal position.
// Dn = n! * (1/0! - 1/1! + 1/2! - 1/3! ..... (-1)^n*1/n!)

```
int main(){
    ll n; cin >> n;
    ll ans = 0;
    for(int i=0;i<=n;i++){

        ll mod_inv = Binary_expo(fact[i] , MOD-2);

        ll help = mulmod(fact[n] , mod_inv);
        if(i&1){
            ans = Mod(ans-help);
        }
        else {
            ans = Mod(ans+help);
        }
    }
    cout<< ans << endl;
}
```

*****longest increasing subsequence O(nlogn)**

```
int main() {
        cin>>n;
        for(int i=0;i<n;i++) cin>>a[i];
        vector<int>lis;
        lis.push_back(a[0]);
        for(int i=1;i<n;i++)  {
            if(lis.back()<a[i]) lis.push_back(a[i]);
            else   {
                int indx=lower_bound(lis.begin(),lis.end(),a[i])-
                                    lis.begin();
                lis[indx]=a[i];
            }
        }
        cout<<lis.size()<<endl;
        return 0;
}
```

***** longest palindromic subsequence (O(n*n))**

```
int dp[1005][1005];
int solution(int B,int E) {  /// B = 0 & E = n-1
    if(B>E) return 0;
    if(B==E) return 1;
    if(dp[B][E]!=-1) return dp[B][E];
    /// if Begin and End char are equal that means
    /// we got a subsequence of 2 lenght and next we can
          reduce the B and E
    if(s[B]==s[E]) return dp[B][E]=2+solution(B+1,E-1);
    else
    {
        ///here at first we reduce Begin or End and searching
           our maximum solution
        return dp[B][E]=max(solution(B,E-1),solution(B+1,E));
    }
}
```

Team: Zero^Infinity(DUET)

***** Euler Tour //// Sum from root to node s**

```
vector<int>adj[limit];
int in[limit],out[limit];
ll val[limit],tr[8*limit],ft[2*limit];
int timer=1;

void Euler_tour(int node,int par){
   in[node]=timer;
   ft[timer]=val[node];
   timer++;
   for(int ch:adj[node])
      if(ch!=par) Euler_tour(ch,node);

   out[node]=timer;
   ft[timer]=-val[node];
   timer++;
}
void built_tree(int node,int s,int e){
   if(s==e) {
      tr[node]=ft[s];
      return;
   }
   int mid=(s+e)/2;
   built_tree(2*node,s,mid);
   built_tree(2*node+1,mid+1,e);
   tr[node]=tr[2*node]+tr[2*node+1];
}
void update(int node,int s,int e,int idx,ll Val){
   if(s==e && s==idx){
      tr[node]=Val;
      ft[s]=Val;
      return;
   }
   int mid=(s+e)/2;
   if(idx<=mid) update(2*node,s,mid,idx,Val);
   else update(2*node+1,mid+1,e,idx,Val);
   tr[node]=tr[2*node]+tr[2*node+1];
}

ll Subtree_sum(int node,int s,int e,int l,int r){
   if(e<l || s>r)
      return 0;
   if(s>=l && e<=r)
      return tr[node];
   int mid=(s+e)/2;
   ll Left=Subtree_sum(2*node,s,mid,l,r);
   ll Right=Subtree_sum(2*node+1,mid+1,e,l,r);
   return Left+Right;
}

int main(){

   int n,q;
   cin>>n>>q;

   for(int i=1;i<=n;i++){
      cin>>val[i];
   }
   for(int i=1;i<n;i++){
      // make adjacent
   }
   Euler_tour(1,-1);   /// Call euler function
   built_tree(1,1,2*n);   /// Built segment tree
   while(q—){   /// Enter query
      int t;  cin>>t;
      if(t==1){     /// update value of a node
         int node,Val;
         cin>>node>>Val;
         int inIdx=in[node];
         int outIdx=out[node];
         update(1,1,2*n,inIdx,Val);
         update(1,1,2*n,outIdx,-Val);
      }
      else{   /// Find the sum from root to node s
         int node;
         cin>>node;
         int inNode=in[node];
         int inRoot=in[1];
         cout<<Subtree_sum(1,1,2*n,inRoot,inNode)<<endl;
      }
   }
   return 0;
}
```

****** xor from node a to b**

```
ll val[limit],tr[8*limit],ft[2*limit];

int level[limit] , LCA[limit][25] , in[limit],out[limit];

int timer=1;

/// Call Euler Tour function

void Euler_tour(int node,int par,int l){;}
int get_lca(int a,int b){

   if(level[a]>level[b]) swap(a,b);

   int d=level[b]-level[a];

   while(d>0){

      int i=log2(d);  b=LCA[b][i];  d-=(1<<i);

   }

   if(a==b) return a;

  for(int i=MaxN;i>-1;i--) {

      if(LCA[a][i]!=-1 && (LCA[a][i]!=LCA[b][i])) {

         a=LCA[a][i],b=LCA[b][i];

      }

   }

   return LCA[a][0];

}
```

Team: Zero^Infinity(DUET)

```
void built_tree(int node,int s,int e){
   if(s==e) {
      tr[node]=ft[s];   return;
   }
   int mid=(s+e)/2;
   built_tree(2*node,s,mid);
   built_tree(2*node+1,mid+1,e);
   tr[node]=( tr[2*node] ^ tr[2*node+1]);
}
void update(int node,int s,int e,int idx,ll Val){
   if(s==e && s==idx){
      tr[node]=Val;
      ft[s]=Val;
      return;
   }
   int mid=(s+e)/2;
   if(idx<=mid) update(2*node,s,mid,idx,Val);
   else update(2*node+1,mid+1,e,idx,Val);
   tr[node] = (tr[2*node] ^ tr[2*node+1]);
}
ll Subtree_xor(int node,int s,int e,int l,int r){
   if(e<l || s>r)  return 0;
   if(s>=l && e<=r)
      return tr[node];
   int mid=(s+e)/2;
   ll Left=Subtree_xor(2*node,s,mid,l,r);
   ll Right=Subtree_xor(2*node+1,mid+1,e,l,r);
   return Left^Right;
}


int main(){
   cin>>n>>q;
   for(int i=1;i<=n;i++)  cin>>val[i];  ///val of node
   for(int i=1;i<n;i++)  // make adjecent
   memset(LCA,-1,sizeof(LCA));
   Euler_tour(1,-1,0);
   for(int i=1;i<=MaxN;i++) {
      for(int j=1;j<=n;j++) {
         if(LCA[j][i-1]!=-1) {
            int p=LCA[j][i-1];
            LCA[j][i]=LCA[p][i-1];
         }
      }
   }
   built_tree(1,1,2*n);
   while(q--){
      int t;  cin>>t;
      if(t==1)   /// update as above
      else{  int a,b;   cin>>a>>b;
         int inA=in[a] , inB=in[b] , inRoot=in[1];
         int lca=get_lca(a,b); int lcaVal=0;
         if(lca!=-1) lcaVal=val[lca];
         ll root_to_a = Subtree_xor(1,1,2*n,inRoot,inA);
         ll root_to_b = Subtree_xor(1,1,2*n,inRoot,inB);
         ll ans=(root_to_a ^ root_to_b ^ lcaVal);
         cout<<ans<<endl;
      }
   }
   return 0;
}
```

***** calculate the sum of values in the subtree of node s**

```
/// Euler TOUR FUNCTION CALL
void Euler_tour(int node,int par){;}

void built_tree(int node,int s,int e){
   if(s==e) {  tr[node]=ft[s];  return;  }
   int mid=(s+e)/2;
   built_tree(2*node,s,mid);
   built_tree(2*node+1,mid+1,e);
   tr[node]=tr[2*node]+tr[2*node+1];
}
```

Team: Zero^Infinity(DUET)

```
void update(int node,int s,int e,int idx,ll Val){
   if(s==e && s==idx){
      tr[node]=Val;   ft[s]=Val;  return;
   }
   int mid=(s+e)/2;
   if(idx<=mid) update(2*node,s,mid,idx,Val);
   else update(2*node+1,mid+1,e,idx,Val);
   tr[node]=tr[2*node]+tr[2*node+1];
}
ll Subtree_sum(int node,int s,int e,int l,int r){
   if(e<l || s>r)  return 0;
   if(s>=l && e<=r)
      return tr[node];
   int mid=(s+e)/2;
   ll Left=Subtree_sum(2*node,s,mid,l,r);
   ll Right=Subtree_sum(2*node+1,mid+1,e,l,r);
   return Left+Right;
}
int main(){
   int n,q;  cin>>n>>q;
   for(int i=1;i<=n;i++){ cin>>val[i]; }
   for(int i=1;i<n;i++) // Make adjecent
   Euler_tour(1,-1);
   built_tree(1,1,2*n);
   while(q--){
      int t;   cin>>t;
      if(t==1) /// update value as above
      else{
         int node; cin>>node;
         int inIdx=in[node];
         int outIdx=out[node];
         cout<<Subtree_sum(1,1,2*n,inIdx,outIdx)/2<<endl;
      }
   }
   return 0;
}
```

***** Subtree maxVal of a tree**

```
/// Euler TOUR FUNCTION CALL
void Euler_tour(int node,int par){;}
void built_tree(int node,int s,int e){
   if(s==e) {  tr[node]=ft[s]; return;  }
   int mid=(s+e)/2;
   built_tree(2*node,s,mid);
   built_tree(2*node+1,mid+1,e);
   tr[node]=max(tr[2*node],tr[2*node+1]);
}
void update(int node,int s,int e,int idx,ll Val){
   if(s==e && s==idx){
      tr[node]=Val; ft[s]=Val; return;
   }
   int mid=(s+e)/2;
   if(idx<=mid) update(2*node,s,mid,idx,Val);
   else update(2*node+1,mid+1,e,idx,Val);
   tr[node]=max(tr[2*node],tr[2*node+1]);
}
ll Subtree_max(int node,int s,int e,int l,int r){
   if(e<l || s>r)  return 0;
   if(s>=l && e<=r)  return tr[node];
   int mid=(s+e)/2;
   ll Left=Subtree_sum(2*node,s,mid,l,r);
   ll Right=Subtree_sum(2*node+1,mid+1,e,l,r);
   return max(Left,Right);
}
int main(){
   int n,q; cin>>n>>q;
  for(int i=1;i<=n;i++){ cin>>val[i]; }
  for(int i=1;i<n;i++) // Make adjecent
   Euler_tour(1,-1);
   built_tree(1,1,2*n);
   while(q--){
      int t; cin>>t;
      if(t==1) /// update value as above
```

```
    else{
        int node; cin>>node;
        int inIdx=in[node];
        int outIdx=out[node];
        cout<<Subtree_max(1,1,2*n,inIdx,outIdx)<<endl;
    }
  }
  return 0;
}
```

***** Inclusion Exclusion**

```
int main(){
  ll n , k; cin >> n >> k;
  vector<ll>a(k);
  for(int i=0;i<k;i++) cin >> a[i];
  vector<ll>divisors_contribution(k+1,0);
  for(int mask=1;mask<(1<<k);mask++){
    ll N=n,nod=0;   ///nod->number of divisors
    for(int bit=0;bit<k;bit++){
      if((mask & ( 1 << bit ))){
        N/=a[bit];  nod++;
      }
    }
    divisors_contribution[nod]+=N;
  }
  ll ans=0;
  for(int i=1;i<=k;i++){
    if(i&1) ans+=divisors_contribution[i];
    else ans-=divisors_contribution[i];
  }
  cout<<ans<<endl;
  return 0;
}
```

***** Ordered set**

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#define ordered_set tree<int, null_type,less_equal<int>,
rb_tree_tag,tree_order_statistics_node_update>
using namespace __gnu_pbds;

ordered_set s;
s.insert(1);
…
//kth element of the set --- 0 base index;
int x=*(s.find_by_order(k));   /
// position of the val in the set  --- 0 base index;
int x= s.order_of_key(val);  /

/// Erase k'th position value of orginal array from the set
a[k] is the value of k'th position of the orginal array &
pos is the postion of that value in the set
   int pos = s.order_of_key(a[k]);
// Erasing the value from the set.
   s.erase(s.find_by_order(pos));
```

***** 0-1 BFS**

```
vector<pair<ll,ll>>adj[limit];
ll dis[limit];    ///level
ll n,m;

void bfs01(ll node){
   for(int i=1;i<=n;i++) dis[i]=1e9;
   deque<ll>dq;
   dq.push_back(node);
   dis[node]=0;
   while(dq.size()>0){
      ll node=dq.front();
      dq.pop_front();

      for(pair<ll,ll> ch:adj[node]){
         if(dis[ch.first] > dis[node] + ch.second){
            dis[ch.first]= dis[node]+ch.second;
            if(ch.second == 1) dq.push_back(ch.first);
            else dq.push_front(ch.first);

         }
      }
   }
}
int main(){
   cin>>n>>m;
   for(int i=1;i<=m;i++){
      ll x,y>>w;  cin>>x>>y>>w;
      adj[x].push_back({y,w});
      adj[y].push_back({x,w});
   }
   bfs01(1);
   cout<<dis[n]<<endl;
   return 0;
}
```

Team: Zero^Infinity(DUET)

***** Dijkstra sssp**

```
int main() {
   int n,e;  cin>>n>>e;
   vector<pair<int,int>>adj[n+1];
   vector<int>dis(n+1,1e9),vis(n+1,0);
   for(int i=1;i<=e;i++){
      int x,y,w;  cin>>x>>y>>w;
      adj[x].push_back({y,w});
      adj[y].push_back({x,w});
   } priority_queue<pair<int,int>,vector<pair<int,int>>,
                           greater<pair<int,int>>>pq;
   dis[1]=0;
   pq.push({0,1});
   while(!pq.empty()) {
      int weight=pq.top().first;
      int node=pq.top().second;
      pq.pop();
      if(vis[node]) continue;
      vis[node]=1;
      for(pair<int,int>child:adj[node]){
         if(weight+child.second<dis[child.first]) { ///(dis of
src + adjecent dis src to child) < dis of child stored before
            dis[child.first]=weight+child.second;
            pq.push({dis[child.first],child.first});
         }
      }
   }
   for(int i=1;i<=n;i++) cout<<dis[i]<<" ";
   cout<<endl;
   return 0;
}
```

***** Bellmen Ford**

```
struct edges{
   int src,des,w;
};
int main(){
   int n,e;  cin>>n>>e;
   vector<edges>adj(e);
   vector<int>Parent(n+1,-1),dis(n+1,1e9);
   for(int i=0;i<e;i++){
      int a,b,w;   cin>>a>>b>>w;
      adj[i].src=a;
      adj[i].des=b;
      adj[i].w=w;
   }
   dis[1]=0;
   bool isupdated=false;
   for(int i=1;i<n;i++){  /// updating n-1 times
      isupdated=false;
      for(int j=0;j<e;j++){
         int src=adj[j].src;
         int des=adj[j].des;
         int w=adj[j].w;
         if(dis[src]!=1e9 && dis[src]+w<dis[des]){
            isupdated=true;
            dis[des]=dis[src]+w;
            Parent[des]=src;
         }
      }
      if(isupdated==false) break;
   }
   ///one more relaxation for detect cycle
   isupdated=true;
   for(int j=0;j<e;j++){
      int src=adj[j].src;
      int des=adj[j].des;
      int w=adj[j].w;
      if(dis[src]!=1e9 && dis[src]+w<dis[des]){

         ///negative edge cycle
         cout<<"This graph has -ve edge cycle"<<endl;
         return 0;
      }
   }
   for(int i=1;i<=n;i++) cout<<dis[i]<<" ";
   cout<<endl;
   return 0;
}
```

***** Z algorithm O(n)**

```
/// Finding the longest prefix stated from any index s[i]
which is also a proper prefix from index s[0]
vector<int>Z(limit);
void Z_function(){
   for (int i=1, l=0, r=0; i<n ; i++) {
      if (i <= r){
         Z[i] = min (r-i+1, Z[i-l]);  /// already calculated
      }
      while ( (i + Z[i]) < n && ( s[Z[i]] == s[i+Z[i]]) ){
         Z[i]++ ;
      }
      if ( ( i+Z[i]-1 ) > r){
         l = i;
         r = i + Z[i]-1;
      }
   }
   return;
}

int main(){
   cin >> n >> s;
   Z_function();
//    for(int i=0 ;i <=n;i++) cout<<Z[i] <<" ";
//    cout<<endl;
   return 0;
}
```

***** Manacher Algorithm O(n)**

```
/// Manacher's algorithm is used to find the longest
palindromic substring in any given string.
vector<int>lps(2*limit);
void manacher(string s , string p){
   int n=s.size() , maxlen = 0 ,maxindex = 0;
   for(int i=1 , center=0 , radius=0 ; i<n-1 ; i++){
```

```
    int mirror = center - (i -center);
    if( i < radius ){  /// Give an initial lps value
       lps[i] = min(lps[mirror] , radius-i);
    }
    while( s[i - lps[i] - 1] == s[i + lps[i] + 1]){
       lps[i]++;
    }
    if(i+lps[i] > radius){  ///update center & radius
       center = i;
       radius = i+lps[i];
    }
    if(maxlen < lps[i]) {  /// updating the max lps
       maxlen = lps[i];
       maxindex = i;
    }
  }
  /// start index of substring in converted string s
  int id1 = maxindex - maxlen + 1;
  /// start index of substring in actual string p
  int id2 = (id1 - 2)/2;
  cout << p.substr(id2 , maxlen) <<endl;
  return;
}

int main(){
  string s , p;
  cin >> p;
  s.push_back('@');
  for(char c:p){
     s.push_back('#');
     s.push_back(c);
  }
  s.push_back('#');
  s.push_back('$');
  manacher(s, p);
  return 0;
}
```

***** Finding articulation point** O(N+M)

```
int in[100005] , low[100005];
int timer;
set<int> AP;  /// store articulation point
void dfs(int node,int par) {
  vis[node]=true;
  in[node]=low[node]=timer++;

  int children=0;///if root has more than one children than
                          It should be an articulation point
  for(int child:g[node]) {
     if(child==par) continue;
     else if(vis[child]) {
        ///back edge so update kore nicchi anchester er in
        time er maddhome
        low[node]=min(low[node],in[child]);
     }
     else {
        ///forward edge
        dfs(child,node);
        ///dfs call hower por child low time er maddhome
        node er low time update kore
        low[node]=min(low[node],low[child]);

        ///This node is an articulation point and not a root
        because it's parents is not -1
        if(low[child]>=in[node] && par!=-1)  {
           cout<<node<<" is an articulation point"<<endl;
           AP.insert(node);///insert into set
        }
        children++;
     }
  }
  ///This means If root has more than one children and it
        must be an articulation point
  if(par==-1&&children>1){
     cout<<node<<" is an articulation point"<<endl;
     AP.insert(node);///insert into set
  }
}
int main() {
    int n,e;   cin>>n>>e;
    for(int i=1;i<=e;i++)  /// make graph
    for(int i=1;i<=n;i++) {
        if(!vis[i]) dfs(i,-1);///root and his parents-1
    }
    for(auto i:AP) cout<<i<<endl;
    return 0;
}
```

***** Suffix array**

```
vector<int> suffix_array(string &s){
 //s.push('$');
 int n = s.size(), alph = 256;
 vector<int> cnt(max(n, alph)), p(n), c(n);

 for(auto c : s) cnt[c]++;
 for(int i = 1; i < alph; i++) cnt[i] += cnt[i - 1];
 for(int i = 0; i < n; i++) p[--cnt[s[i]]] = i;
 for(int i = 1; i < n; i++)
  c[p[i]] = c[p[i - 1]] + (s[p[i]] != s[p[i - 1]]);

 vector<int> c2(n), p2(n);

 for(int k = 0; (1 << k) < n; k++){
  int classes = c[p[n - 1]] + 1;
  fill(cnt.begin(), cnt.begin() + classes, 0);

  for(int i = 0; i < n; i++) p2[i] = (p[i] - (1 << k) + n)%n;
  for(int i = 0; i < n; i++) cnt[c[i]]++;
  for(int i = 1; i < classes; i++) cnt[i] += cnt[i - 1];
  for(int i = n - 1; i >= 0; i--) p[--cnt[c[p2[i]]]] = p2[i];

  c2[p[0]] = 0;
  for(int i = 1; i < n; i++){
   pair<int, int> b1 = {c[p[i]], c[(p[i] + (1 << k))%n]};
   pair<int, int> b2 = {c[p[i - 1]], c[(p[i - 1] + (1 << k))
%n]};
```

```
      c2[p[i]] = c2[p[i - 1]] + (b1 != b2);
    }

    c.swap(c2);
  }
  return p;
}

// Longest Common Prefix with SA O(n)
vector<int> lcpr(string &s, vector<int> &p){

  int n = s.size();
  vector<int> ans(n - 1), pi(n);
  for(int i = 0; i < n; i++) pi[p[i]] = i;

  int lst = 0;
  for(int i = 0; i < n - 1; i++){
    if(pi[i] == n - 1) continue;
    while(s[i + lst] == s[p[pi[i] + 1] + lst]) lst++;

    ans[pi[i]] = lst;
    lst = max(0, lst - 1);
  }

  return ans;
}

int main(){

    string s;  cin >> s;
    /// suffix array
    vector<int>sa = suffix_array(s);

    for(int i =0 ;i<sa.size();i++) cout<<sa[i] <<" ";
    cout<<endl;

    vector<int>lcp = lcpr(s , sa);
    /// Longest Repeated Substring O(n)

    int lrs = 0;
    for (int i = 0; i < lcp.size(); ++i) {cout<<lcp[i] <<"
";lrs=max(lrs,lcp[i]);}

    cout<< endl << lrs <<endl;
    /// distinct substring

    int ds=s.size() - sa[0];
    for(int i=1;i<sa.size();i++){

        ll len = s.size() - sa[i];
        len -= lcp[i-1];
        ds+=len;
    }
    cout<<ds<<endl;
    return 0;

}
```