1 Template

1.1 template

```
#include <bits/stdc++.h>
using namespace std;
#define INF 1<<30
#define endl '\n'
#define maxn 1000005
#define FASTIO ios_base::sync_with_stdio(false), cin.tie(0), cout.tie(0);
typedef long long ll;
const double PI = acos(-1.0);
const ll mod = 1e9 + 7;

inline void normal(ll &a) { a %= mod; (a < 0) && (a += mod); }
inline ll modMul(ll a, ll b) { a %= mod, b %= mod; normal(a), normal(b);
return (a * b) % mod; }
inline ll modAdd(ll a, ll b) { a %= mod, b %= mod; normal(a), normal(b);
return (a + b) % mod; }
inline ll modSub(ll a, ll b) { a %= mod, b %= mod; normal(a), normal(b); a
-= b; normal(a); return a; }
inline ll modPow(ll b, ll p) { ll r = 1; while (p) { if (p & 1) r =
modMul(r, b); b = modMul(b, b); p >>= 1; } return r; }
inline ll modInverse(ll a) { return modPow(a, mod - 2); }
inline ll modDiv(ll a, ll b) { return modMul(a, modInverse(b)); }


///**
template < typename F, typename S >
ostream& operator << ( ostream& os, const pair< F, S > & p ) {
  return os << "(" << p.first << ", " << p.second << ")";
}

template < typename T >
ostream &operator << ( ostream & os, const vector< T > &v ) {
  os << "{";
  for (auto it = v.begin(); it != v.end(); ++it) {
    if ( it != v.begin() ) os << ", ";
    os << *it;
  }
  return os << "}";
}

template < typename T >
ostream &operator << ( ostream & os, const set< T > &v ) {
  os << "[";
  for (auto it = v.begin(); it != v.end(); ++it) {
    if ( it != v.begin()) os << ", ";
    os << *it;
  }
  return os << "]";
}

template < typename F, typename S >
ostream &operator << ( ostream & os, const map< F, S > &v ) {
  os << "[";
  for (auto it = v.begin(); it != v.end(); ++it) {
    if ( it != v.begin() ) os << ", ";
    os << it -> first << " = " << it -> second ;
  }
  return os << "]";
}

#define dbg(args...) do {cerr << #args << " : "; faltu(args); } while(0)

clock_t tStart = clock();
#define timeStamp dbg("Execution Time: ", (double)(clock() -
tStart)/CLOCKS_PER_SEC)

void faltu () { cerr << endl; }

template <typename T>
void faltu( T a[], int n ) {
  for (int i = 0; i < n; ++i) cerr << a[i] << ' ';
  cerr << endl;
}

template <typename T, typename ... hello>
void faltu( T arg, const hello &... rest) { cerr << arg << ' ';
faltu(rest...); }
// Program showing a policy-based data structure.
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp>
#include <functional> // for less
#include <iostream>
```

```
using namespace __gnu_pbds;
using namespace std;
// GNU link : https://goo.gl/WVDL6g
typedef tree<int, null_type, less_equal<int>, rb_tree_tag,
        tree_order_statistics_node_update>
        new_data_set;

// find_by_order(k) {  kth ordered element         kth    ,    !
// order_of_key(x) {  x

//*//**_____________________________________________________**/
```

2 Graph

2.1 AP

```
#define Max 100000
vector<int> graph[Max];
int parent[Max];
int low[Max];
int d[Max];
int visited[Max];
bool isArticulationPoint[Max];
int Time = 0;

void dfs(int u, int root)
{
    Time = Time + 1;
    visited[u] = Time;
    d[u] = low[u] = Time;
    int noOfChildren = 0;
    for(int i = 0; i <graph[u].size(); i++){
        int v = graph[u][i];
        if(v == parent[u])continue;
        parent[v] = u;
        if(visited[v]) low[u] = min(low[u], d[v]);
        else{
            noOfChildren = noOfChildren + 1;
            dfs(v, root);
            low[u] = min(low[u], low[v]);
            if(low[v] >= d[u] and u != root)isArticulationPoint[u] = true;
        }
    }
    if(u == root and noOfChildren > 1)isArticulationPoint[u] = true;
}//if(isArticulationPoint[i])cout << i<<endl; ans store
```

2.2 Dijkstra

```
const int INF = 1e9 + 7;
vector<pair<int,int> > graph[1005];
vector<int> p,d;
void dijkstra(int s, int n)
{
    d.assign(n,INF);
    //p.assign(n, -1);

    d[s] = 0;
    using pii = pair<int,int>;
    priority_queue<pii, vector<pii>, greater<pii>> PQ;

    PQ.push({0,s});

    while(!PQ.empty())
    {
        int v = PQ.top().second;
        int d_v = PQ.top().first;
        PQ.pop();
       if(d_v != d[v]) continue;

        for(auto x: graph[v])
        {
            int to = x.first;
            int len = x.second;
            int mx = max(len, d[v]);
            if(mx < d[to]){
                d[to] = mx;
                PQ.push({d[to], to});
            }
           // cout << to << " "<<len << " "<<d[to]<<endl;
           //d[to] = min(d[to], max(len, d[v]));
        }
    }
}
```

2.3 Flow

```
const int inf = 1e9,N = 105;
struct Edge
{
  int to, rev; int f, cap;
  Edge();
  Edge(int to, int rev, int f, int cap): to(to), rev(rev), f(f), cap(cap)
  {}
};
vector<Edge> graph[N];
void addEdge(int u, int v, int cap)
{
  Edge a = Edge(v, (int)graph[v].size(), 0, cap);
  Edge b = Edge(u, (int)graph[u].size(), 0, cap);
  graph[u].push_back(a); graph[v].push_back(b);
}

int n, start[N], level[N];
queue<int> Q;

bool dinic_bfs(int s, int t)
{
  fill(level, level + n + 1, -1);
  Q.push(s);
  level[s] = 0;
  while (!Q.empty()) {
    int u = Q.front();
    Q.pop();
    for (int i = 0; i < (int)graph[u].size(); i++) {
      Edge &E = graph[u][i];
      int v = E.to;
      if (level[v] < 0 && E.f < E.cap) {
        Q.push(v);
        level[v] = level[u] + 1;
      }
    }
  }
  return level[t] >= 0;
}

int dinic_dfs(int u, int dst, int flow)
{
  if (u == dst) return flow;
  for (int &i = start[u]; i < (int)graph[u].size(); i++) {
    Edge &E = graph[u][i];
    int v = E.to;
    if (level[v] == level[u] + 1 && E.f < E.cap) {
      int cur_flow = dinic_dfs(v, dst, min(flow, E.cap - E.f));
      if (cur_flow > 0) {
        E.f += cur_flow;
        graph[v][E.rev].f -= flow;
        return cur_flow;
      }
    }
  }
  return 0;
}

int dinic_flow(int s, int t)
{
  int flow = 0;
  while ((dinic_bfs(s, t))) {
    fill(start, start + n + 1, 0);
    int delta;
    while ((delta = dinic_dfs(s, t, INT_MAX))) flow += delta;
  }
  return flow;
}
int main()
{
  int T;
  scanf("%d", &T);
  for (int cs = 1; cs <= T; cs++) {
    scanf("%d", &n);
    int s, t, c;
    scanf("%d %d %d", &s, &t, &c);
    while (c--) {
      int u, v, w;
      scanf("%d %d %d", &u, &v, &w);
      addEdge(u, v, w);
    }
    int ans = dinic_flow(s, t);
    printf("Case %d: %d\n", cs, ans);
    for (int i = 0; i <= n; i++)
```

```
      graph[i].clear();
  }
  return 0;
}
```

2.4 MST

```
const int maxn = (int) 2e5 + 5;
struct edge
{
    int u,v,w;
};
vector<edge>graph, output;
int parent[maxn], mstValue = 0;

bool cmp (edge a, edge b)
{
    return a.w < b.w;
}

int Find(int r)
{
    if(parent[r] == r)
        return r;
    return parent[r] = Find(parent[r]);
}

void initPar(int r)
{
    for(int i = 0; i <= r; i++)parent[i] = i;
}

void kruskals_Algorithm(int n)
{
    sort(graph.begin(), graph.end(), cmp);
    for(int i = 0; i < (int)graph.size(); i++){
        cout << graph[i].u << " "<<graph[i].v << " "<< graph[i].w<<endl;
    }
    initPar(n);
    int cnt = 0;

    for(int i = 0; i < (int)graph.size(); i++){
        int uPr = Find(graph[i].u);
        int vPr = Find(graph[i].v);

        if(uPr != vPr){
            if(cnt == n-1) break;

            output.push_back(graph[i]);
            mstValue += graph[i].w;
            parent[uPr] = vPr;
            cnt++;
        }
    }
}
```

2.5 SCC

```
const int maxn = 10005;
vector<int> g[maxn], gr[maxn];
vector<bool > used;
vector<int> order, component;
void dfs1(int u)
{
    used[u] = true;
    for(int i = 0; i < g[u].size(); i++){
        int v = g[u][i];
        if(!used[v])
            dfs1(v);
    }
    order.push_back(u);
}
void dfs2(int u)
{
    used[u] = true;
    component.push_back(u);
    for(int i = 0; i < gr[u].size(); i++){
        int v = gr[u][i];
        if(!used[v])
            dfs2(v);
    }
}
int main()
{
   // freopen("in.txt", "r", stdin);
    ios_base::sync_with_stdio(false);
```

```
    cin.tie(0);
    int n , m;
    cin >> n >> m;
    for(int i = 0; i < m; i++){
        int u,v;
        cin >> u >> v;
        g[u].push_back(v);
        gr[v].push_back(u);
    }
    used.assign(n+1, false);
    for(int i = 1; i <= n; i++){
        if(!used[i])
            dfs1(i);
    }
    used.assign(n+1, false);
    for(int i = 1; i <= n; i++){
        int v = order[n - i];
        if(!used[v]){
            dfs2(v);
            for(int k = 0; k < component.size(); k++) cout << component[k]
            << " ";
            cout << endl;
            component.clear();
        }
    }
    return 0;
}
```

3 Data Structure

3.1 Segment Tree

```
int arr[100001];
int tree[3*100001];

void Init( int node, int b, int e)
{
    if(b>e)
        return;
    if( b == e)
    {
        tree[node] = 0;
        return ;
    }
    int Left = node*2;
    int Right = node*2+1;
    int mid  = (b+e)/2;
    Init( Left, b, mid);
    Init( Right, mid+1, e);

    tree[node] = tree[Left] + tree[Right];
}

int Query( int node, int b, int e, int i, int j)
{
    if( b >= i && e <= j)
        return tree[node];
    if( j<b || i>e )
        return 0;

    int Left = node*2;
    int Right = node*2+1;
    int mid  = (b+e)/2;
    int p1 = Query( Left, b, mid, i,j);
    int p2 = Query( Right, mid+1, e, i,j);

    return p1+p2;
}

void Update( int node, int b, int e, int i, int j, int newvalue)
{
    if( b >= i && e <= j)
    {
        tree[node] = newvalue;
        return ;
    }
    if( j<b || i>e )
        return ;

    int Left = node*2;
    int Right = node*2+1;
    int mid  = (b+e)/2;
    Update( Left, b, mid, i,j,newvalue);
    Update( Right, mid+1, e, i,j,newvalue);
```

```
 tree[node] = tree[Left] + tree[Right];
}

/// Lazy
ll arr[mx];
struct info {
    ll prop, sum;
} tree[mx * 4];

void init(int node, int b, int e)
{
    if (b == e) {
        tree[node].sum = arr[b];
        return;
    }
    int Left = node * 2;
    int Right = node * 2 + 1;
    int mid = (b + e) / 2;
    init(Left, b, mid);
    init(Right, mid + 1, e);
    tree[node].sum = tree[Left].sum + tree[Right].sum;
}

void update(int node, int b, int e, int i, int j, ll x)
{
    if (i > e || j < b)
        return;
    if (b >= i && e <= j)
    {
        tree[node].sum += ((e - b + 1) * x);
        tree[node].prop += x;
        return;
    }
    int Left = node * 2;
    int Right = (node * 2) + 1;
    int mid = (b + e) / 2;
    update(Left, b, mid, i, j, x);
    update(Right, mid + 1, e, i, j, x);
    tree[node].sum = tree[Left].sum + tree[Right].sum + (e - b + 1) *
    tree[node].prop;
}
```

```
ll query(int node, int b, int e, int i, int j, ll carry = 0)
{
    if (i > e || j < b)
        return 0;
    if (b >= i and e <= j)
        return tree[node].sum + carry * (e - b + 1);
    int Left = node << 1;
    int Right = (node << 1) + 1;
    int mid = (b + e) >> 1;
    ll p1 = query(Left, b, mid, i, j, carry + tree[node].prop);
    ll p2 = query(Right, mid + 1, e, i, j, carry + tree[node].prop);
    return p1 + p2;
}
```

3.2 Wavelet Tree

```
const int N = 3e6,M = 1e6;vector<int> g[N];int a[N];
struct wavelet_tree
{
int lo, hi;wavelet_tree *l, *r;
vector<int> b;vector<int> c;// c holds the prefix sum of elements
// nos are in range [x,y]// array indices ar [from, to]
wavelet_tree(int *from, int *to, int x, int y)
{lo = x;hi = y;
    if (from >= to) return;
    if (hi == lo) {b.reserve(to - from + 1);
b.push_back(0);c.push_back(to - from + 1);c.push_back(0);
      for (auto it = from; it != to; it++) {
        b.push_back(b.back() + 1);c.push_back(c.back() + *it);
      }
      return;
    }
    int mid = (lo + hi) / 2;
    auto f = [mid](int x) {
      return x <= mid;
    };
    b.reserve(to - from + 1);b.push_back(0);c.reserve(to - from + 1);
    c.push_back(0);
    for (auto it = from; it != to; it++) {
    b.push_back(b.back() + f(*it));c.push_back(c.back() + *it);
    }
    // see how lamda function is used here
auto pivot  = stable_partition(from, to, f);
```

```
l = new wavelet_tree(from, pivot, lo, mid);
r = new wavelet_tree(pivot, to, mid + 1, hi);
  }
  // swap a[i] with a[i+1]  , if a[i]!=a[i+1] call swapadjacent(i)
  void swapadjacent(int i)
  {
    if (lo == hi)return ;
b[i] = b[i - 1] + b[i + 1] - b[i];c[i] = c[i - 1] + c[i + 1] - c[i];
    if ( b[i + 1] - b[i] == b[i] - b[i - 1])
    {
if (b[i] - b[i - 1]) return this->l->swapadjacent(b[i]);
else return this->r->swapadjacent(i - b[i]);
    }
    else return ;
  }
  //kth smallest element in [l, r]
  int kth(int l, int r, int k)
  {
    if (l > r) return 0;
    if (lo == hi) return lo;
    int inleft = b[r] - b[l - 1];
    int lb = b[l - 1];//amt of nos in first (l-1) nos that go in left
    int rb = b[r]; //amt of nos in first (r) nos that go in left
    if (k <= inleft) return this->l->kth(lb + 1, rb, k);
    return this->r->kth(l - lb, r - rb, k - inleft);
  }
  //count of nos in [l,r] less than or equal to k
  int LTE(int l, int r, int k)
  {
    if (l > r or k < lo) return 0;
    if (hi <= k) return r - l + 1;
    int lb = b[l - 1]; int rb = b[r];
    return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l - lb, r - rb, k);
  }
  // count of nos in [l,r] equal to k
  int count(int l, int r, int k)
  {
    if (l > r or k < lo or k > hi) return 0;
    if (lo == hi) return r - l + 1;
    int lb = b[l - 1];int rb = b[r];int mid = (lo + hi) / 2;
    if (k <= mid) return this->l->count(lb + 1, rb, k);
    return this->r->count(l - lb, r - rb, k);
  }
  // sum of nos in [l,r] less than or eqaul to k
  int sumk(int l, int r, int k)
  {
    if (l > r or k < lo) return 0;
    if (hi <= k) return c[r] - c[l - 1];
    int lb = b[l - 1];int rb = b[r];
    return this->l->sumk(lb + 1, rb, k) + this->r->sumk(l - lb, r - rb,
    k);
  }
  ~wavelet_tree() {
    delete l;delete r;
  }
};
int main()
{
  int q,x,n,l,r,k;cin >> n;
  for (int i = 1; i <= n; i++) cin >> a[i];
  wavelet_tree T(a + 1, a + n + 1, 1, M);
  cin >> q;
while (q--) { cin >> x;cin >> l >> r >> k;
    if (x == 0) { // kth smallest
      cout << "Kth smallest: ";
      cout << T.kth(l, r, k) << endl;
    }
    else if (x == 1) { // lss than or equal to k
      cout << "LTE: ";
      cout << T.LTE(l, r, k) << endl;
    }
    else if (x == 2) { // count occurence of K in [l, r]
      cout << "Occurence of K: ";
      cout << T.count(l, r, k) << endl;
    }
    else if (x == 3) {//sum of elements less than or equal to K in [l, r]
      cout << "Sum: ";
      cout << T.sumk(l, r, k) << endl;
}}}
```

3.3 SQRT

```
#define nx 10000 int blk_sz,ar[nx],block[nx];
///O(1)
void update(int idx, int val)
```

```
{
int blockNumber = idx/blk_sz;block[blockNumber] += val - ar[idx];ar[idx] =
val;
}
/// O(sqrt(n))
int query(int l, int r)
{ int sum = 0;
while(l < r && l%blk_sz != 0 && l != 0){sum += ar[l];l++;}
while(l+blk_sz <= r) {sum += block[l/blk_sz];l += blk_sz;}
while(l <= r) { sum += ar[l]; l++;}
//   cout << sum << " ";
return sum;
}
void preprocess(int a[], int n)
{ int blk_idx = -1;blk_sz = sqrt(n);
    for(int i = 0; i < n; i++)
    { ar[i] = a[i];
        //   cout << ar[i] << " ";
if(i%blk_sz == 0){blk_idx++;} block[blk_idx] += ar[i];
        //    cout << block[blk_idx] <<" ";
    }
}
int main()
{ cin >> n; int a[n];
for(int i = 0; i < n; i++) cin >> a[i];
    preprocess(a, n);
cout << query(0,9)<<endl;cout << query(3,8)<<endl;cout <<
query(1,6)<<endl;
    update(8,0);
cout << query(8,8)<<endl;
}
```

3.4 LCA

```
/// lca using sparse table - O(nlogn).
int n, u, v;
int dp[maxn][18], depth[maxn];
vector<int> graph[maxn];

void dfs(int u, int parent)
{
    dp[u][0] = parent;
    for (auto v : graph[u]) {
        if (v == parent) continue;
        depth[v] = depth[u] + 1;
        dfs(v, u);
    }
}

int lca(int u, int v)
{
    if (depth[u] < depth[v]) swap(u, v);
    for (int k = 17; k >= 0; k--) {
        if (depth[u] - (1 << k) >= depth[v]) {
            u = dp[u][k];
        }
    }

    if (u == v) return u;
    for (int k = 17; k >= 0; k--) {
        if (dp[u][k] != dp[v][k]) {
            u = dp[u][k];
            v = dp[v][k];
        }
    }
    return dp[u][0];
}
int main()
{
    int T;
    //cin >> T;
    T = 1;
    for (int cs = 1; cs <= T; cs++) {
        scanf("%d", &n);
        for (int i = 1; i < n; i++) {
            scanf("%d %d", &u, &v);
            graph[u].push_back(v);
            graph[v].push_back(u);
        }

        memset(dp, -1, sizeof dp);
        dfs(1, -1);

        for (int k = 1; k <= 17; k++) {
            for (int u = 1; u <= n; u++) {
```

```
                if (dp[u][k - 1] == -1)continue;
                dp[u][k] = dp[dp[u][k - 1]][k - 1];
            }
        }
    }
    int q;
    scanf("%d", &q);
    while (q--) {
        int u, v;
        scanf("%d %d", &u, &v);
        printf("lca (%d,%d) = %d\n", u, v, lca(u, v));
    }
}
```

3.5 Trie

```
///O(N) #define INF 1<<30,MAX 10005
struct node {
  bool endmark;node* next[27];
  node(){ endmark = false;
    for(int i = 0; i < 26; i++) next[i] = NULL;
  }
} *root;
void insert(char* str, int len)
{ node* curr = root;
  for(int i = 0; i < len; i++){int id = str[i] - 'a';
    if(curr->next[id] == NULL) curr->next[id] = new node();
    curr = curr->next[id];
  }curr->endmark = true;
}
bool search(char* str, int len)
{ node* curr = root;
  for(int i = 0; i < len; i++){ int id = str[i] - 'a';
    if(curr->next[id] == NULL) return false;
    curr = curr->next[id];
  }return curr->endmark;
}
void del(node* cur)
{ for(int i = 0; i < 26; i++)del(cur->next[i]);delete(cur);
}
int main()
{ root = new node();int num_word;cin >> num_word;
for(int i = 1; i<= num_word; i++){
char str[50];scanf("%s", str);insert(str, strlen(str));
}
    int query;cin >> query;
    for(int i = 1; i <= query; i++){
      char str[50];scanf("%s", str);
      if(search(str, strlen(str))) puts("FOUND");
      else puts("NoT FOUND"); }
    del(root);
}
```

3.6 SPLAY

```
const string EMPTY = "";
struct SplayTree {
  int v; // Value of node
  SplayTree *child[2]; // Left child -> [0], right child -> [1]
  SplayTree *parent; // Parent of node
  SplayTree(int _v) {
    v = _v; child[0] = child[1] = parent = NULL;
  }
  void Rotate() {
    SplayTree *g = parent->parent;
    bool isLeft = (parent->child[0] == this);
    // isLeft == True -> rightRotate
    // isLeft == False -> leftRotate
    parent->child[isLeft ^ 1] = child[isLeft];
    if (child[isLeft] != NULL) child[isLeft]->parent = parent;
    child[isLeft] = parent;
    parent->parent = this;
    if (g != NULL) {
      bool parentIsLeft = (g->child[0] == parent);
      g->child[parentIsLeft ^ 1] = this;
    }
    parent = g;
  }
  void Splay() {
    while (parent != NULL) {
      if (parent->parent != NULL) {
        bool parentIsLeft = parent->parent->child[0] == parent;
        bool isLeft = parent->child[0] == this;
        if (parentIsLeft == isLeft) parent->Rotate();
        else Rotate();
      }
```

```
      Rotate();
    }
  }
  void Destroy() {
    SplayTree* par = parent;
if (par != NULL) { bool isRight = (par->child[1] == this);
par->child[isRight] = NULL;}
    parent = NULL;
}
SplayTree* FindNode(int v) { SplayTree *x = this; SplayTree *xx = NULL;
    while (x != NULL) { xx = x;
      if (x->v > v) {x = x->child[0];}
      else if (x->v < v) {x = x->child[1];}
      else return x; }
    return xx;
}

SplayTree* Search(int v){ SplayTree* x = FindNode(v);x->Splay();return x;}

  SplayTree* Insert(int v) {
SplayTree* par = FindNode(v);
if (par->v == v) { par->Splay();return par;}

    SplayTree* x = new SplayTree(v);
    if (par->v < v) {par->child[1] = x;}
    else if (par->v > v) {par->child[0] = x;}
    x->parent = par;x->Splay();
    return x;
  }

  SplayTree* FindMax(SplayTree* root) {
    SplayTree* x = NULL;
    while (root != NULL) { x = root;root = root->child[1];}
    return x;
  }
  SplayTree* Delete(int v) {
    SplayTree* x = FindNode(v);x->Splay();

    if (x->v != v) { return x;}

    SplayTree* leftSubTree = x->child[0];
    SplayTree* rightSubTree = x->child[1];

    x->child[0] = x->child[1] = NULL;

    if (leftSubTree != NULL)  leftSubTree->parent = NULL;
    if (rightSubTree != NULL) rightSubTree->parent = NULL;

if (leftSubTree != NULL) {SplayTree* maxNode = FindMax(leftSubTree);
      maxNode->child[1] = rightSubTree;
      if (rightSubTree != NULL) rightSubTree->parent = maxNode;
      maxNode->Splay();
      return maxNode;
    }
    return rightSubTree;
  }

  pair<SplayTree*, SplayTree*> SplitByValue(int v) {
    SplayTree* x = FindNode(v); x->Splay();
    SplayTree *leftTree, *rightTree;
    if (x->v <= v) {
      // Destroy right edge
      leftTree = x; rightTree = x->child[1];

      if (x->child[1] != NULL) x->child[1]->Destroy();
    }

    else {
      // Destroy left edge
      leftTree = x->child[0]; rightTree = x;

      if (x->child[0] != NULL)  x->child[0]->Destroy();
    }
    return make_pair(leftTree, rightTree);
  }

  void Print(string prefix = EMPTY, bool isRight = false, bool isRoot =
  true) {
    if (child[1])
      child[1]->Print(prefix + (!isRight && !isRoot ? "|   " : "    "),
      true, false);

    cout << prefix;
    cout << (isRoot ? "---" : (isRight ? ".--" : "`--"));
```

```
    cout << v << endl;

    if (child[0])
      child[0]->Print(prefix + (isRight ? "|   " : "    "), false, false);
  }
};

int main() {
SplayTree* w = new SplayTree(1);SplayTree* x = new SplayTree(2);
SplayTree* y = new SplayTree(3);SplayTree* z = new SplayTree(4);
SplayTree* a = new SplayTree(5);SplayTree* b = new SplayTree(6);
SplayTree* c = new SplayTree(7);SplayTree* d = new SplayTree(8);
SplayTree* e = new SplayTree(9);SplayTree* f = new SplayTree(10);
SplayTree* g = new SplayTree(11);
w->child[1] = x;x->parent = w;x->child[1] = y;y->parent = x;y->child[1] =
z;
z->parent = y;z->child[1] = a;a->parent = z;a->child[1] = b;b->parent = a;
b->child[1] = c;c->parent = b;c->child[1] = d;d->parent = c;
d->child[1] = e;e->parent = d;e->child[1] = f;f->parent = e;f->child[1] =
g;
g->parent = f;
SplayTree* root = w;
root->Print();g->Splay();
root = g;root->Print();
c->Splay();root = c;root->Print();
root = root->Search(6);root->Print();
if (root->v == 6)cout << "Yes" << endl;
else cout << "No" << endl;
root = root->Search(0);root->Print();
if (root->v == 0)cout << "Yes" << endl;
else cout << "No" << endl;
root = root->Insert(13);root->Print();
root = root->Insert(12);root->Print();
root = root->Delete(8);root->Print();
root = root->Delete(9);root->Print();
root = root->Delete(5);root->Print();
pair<SplayTree*, SplayTree*> roots = root->SplitByValue(5);
if (roots.first != NULL) {
cerr << "First Tree\n";
roots.first->Print();
}if (roots.second != NULL) {
    cerr << "Second Tree\n";
    roots.second->Print();
    cerr << "#########################\n";
  }
}
```

4 Number Theory

4.1

```
#define NN 10000005 long total[1000005]; bool Isprime[NN];
int prime[NN]; int totalPrime;
void EXTENDED_EUCLID(int64 a, int64 b) {
if (b == 0) { x = 1; y = 0; d = a; return; }
EXTENDED_EUCLID(b, a % b);x = x - (a / b) * y;
  swap(x, y);}
void SIEVE()
{
int t = sqrt(NN);Isprime[0]=Isprime[1]=1;
for(int i = 4; i <= NN; i+= 2) Isprime[i] = true;
for( int i=3; i<=t; i += 2 ){
if( !Isprime[i] ){
for( int j=i*i; j<NN; j+= i+i)Isprime[j] = true; } }
totalPrime = 0;prime[totalPrime++] = 2;
for(int i=3; i<NN; i+=2) if(!Isprime[i])prime[totalPrime++] = i;
}
/// BITWISE SIEVE
int flag[M/32];int cnt;int prime[5761482];
unsigned ans;unsigned store[5761482];

void prime_gen()
{
int add,x=0;prime[x++]=2;
for(int i = 4; i<M; i+=2)flag[i/32]=_set(flag[i/32],i%32);
int sq = sqrt(M);for(int i = 3; i<M; i+=2){
if(check(flag[i/32],i%32)==0){ prime[x++]=i;
            if(sq>=i)
            { add = i*2;
for(int j = i*i; j<M; j+=add) flag[j/32]=_set(flag[j/32],j%32);
            }}} cnt=x; }

void CountDiv()
{
```

```
for( int i=2;i<NN;i++ ){
int N = i;int t = sqrt(N);int res = 1;
    for( int j=0;;j++ ){
    if( t<prime[j] ) break;int cnt = 1;
    while( N%prime[j]==0 )
    { N/=prime[j] ;cnt++;}
            t = sqrt(N); res*=cnt;
        }
        if( N>1 ) res*=2; d[i] = res;}
}

void primeFactor()
{
for( int i=2;i<NN;i++ ){
int N = i;int t = sqrt(N);
for( int j=0;;j++ ){
if( t < prime[j] ) break;bool hasFactor = false;
while( N % prime[j] == 0 ) {
N /= prime[j] ;hasFactor = true;}t = sqrt(N);
if( hasFactor ) factor[i].push_back(prime[j]);
        }
if( N>1 ) factor[i].push_back(N);
    }
}

int arr[SIZE];
int segmentedSieve ( int a, int b ) {
if ( a == 1 ) a++;int sqrtn = sqrt ( b );
memset ( arr, 0, sizeof arr );
    for ( int i = 0; i < prime.size() && prime[i] <= sqrtn; i++ ) {
int p = prime[i];int j = p * p;
if ( j < a ) j = ( ( a + p - 1 ) / p ) * p;
for ( ; j <= b; j += p ) {arr[j-a] = 1;}
    } int res = 0;
for (i=a;i<=b;i++){if(arr[i-a]==0)res++;
    } return res;
}

int SOD( int n ) {int res = 1;
int sqrtn = sqrt ( n );
for ( int i = 0; i < prime.size() && prime[i] <= sqrtn; i++ ) {
if ( n % prime[i] == 0 ) {int tempSum = 1;int p = 1;
while ( n % prime[i] == 0 ) {n /= prime[i];p *= prime[i];
tempSum += p;}sqrtn = sqrt ( n );res *= tempSum;}}
if ( n != 1 ) {res *= ( n + 1 );}return res;
}
int catalan[MAX];
void init() {catalan[0] = catalan[1] = 1;
for (int i=2; i<=n; i++) {catalan[i] = 0;
for (int j=0; j < i; j++) {
            catalan[i] += (catalan[j] * catalan[i-j-1]) % MOD;
if (catalan[i] >= MOD) {catalan[i] -= MOD;}}}}

/// BELLMAN FORD
struct edge
{ int u, v, w; };
edge data[MAX];
int key[MAX];
int main() {
 int n, m, i, j, cost;
 scanf("%d %d", &n, &m);
 for(i=1; i<=m; i++) scanf("%d %d %d", &data[i].u, &data[i].v,
 &data[i].w);
 for(i=1; i<=n; i++) key[i]=INF;
 key[1]=0;
 for(i=1; i<n; i++)
 {
 for(j=1; j<=m; j++) {
 cost=key[data[j].u]+data[j].w;
 if(key[data[j].v]>cost) key[data[j].v]=cost;
 }
 }
 for(j=1; j<=m; j++) {
 cost=key[data[j].u]+data[j].w;
 if(key[data[j].v]>cost) break;
 }
 if(j>m)printf("no negative cycle\n");
 else printf("negative cycle\n");
 return 0;
}
```

4.2

```
/// seive phi..
ll phi[MAX];
```

```
void seivePHI(){
for(i = 2; i < MAX; i++){ if(phi[i] == 0){ phi[i] = i - 1;
for(j = i*2; j < MAX; j += i){ if(phi[j] == 0)phi[j] = j;phi[j] /=
i;phi[j] *= (i-1);
}}}}

ll po(ll x, ll y){ans = 1;while(y--) ans *= x;return ans;}
ll prime(ll a){
for(ll i = 1; i*i <= a; i++){if(a%i == 0)return 1;}
}
ll phi(ll n)
{ ll i,mul = 1, holder, fre = 0;
  if(prime(n) == 0) mul = n - 1;
  else{
for(i = 2; i*i <= n; i++){if(n%i == 0){
while(n%i == 0){ n = n/i;holder = i;fre++;}
mul *= (po(holder, fre-1)*(holder - 1));fre = 0; }}
    if(n != 1){ mul *= (n-1);}
  }return mul;
}
```

4.3

```
typedef long long ll;
using u64 = uint64_t;
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod)
{
    u64 result = 1;
    base %= mod;
    while(e){
        if(e & 1)
            result = (u128) result * base % mod;
        base = (u128) base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_compsite(u64 n, u64 a,u64 d,  int s)
{
    u64 x = binpower(a, d, n);
    if(x == 1 || x == n - 1)
        return false;
    for(int r = 1; r < s; r++){
        x = (u128)x * x % n;
        if(x == n - 1)
            return false;
    }
    return true;
};

bool MillerRabin(u64 n) // returns true if n is probably prime, else
returns false.
{
    if(n < 2)
        return false;
    int s = 0;
    u64 d = n - 1;
    while((d & 1) == 0){
        d >>= 1;
        s++;
    }
    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (n == a)
            return true;
        if (check_compsite(n, a, d, s))
            return false;
    }
    return true;
}///if(MillerRabin(n)) cout << "YES\n";
```

5 Geometry

5.1

Usage: Use `solve` with `vector<Point>`

```
const double EPS=1e-9,PI=acos(-1);
/// POINT
struct point{
     double x, y; // check the data type
     point(){ x=y=0.0; }
     point(double _x, double _y){ x=_x; y=_y; } // user defined
     bool operator< (point p) const{    // sorting
```

```
          if(fabs(x-p.x)>EPS) return x<p.x;
          else return y<p.y;
     }
};
// Euclidean distance
double dist(point p1, point p2){
     // return hypot(p1.x-p2.x, p1.y-p2.y);    hypot(dx,
     dy)=sqrt(dx*dx+dy*dy) :v
     return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}

// rotate p by rad(in radian) CCW w.r.t origin (0, 0)
point pointRotate(point p, double rad){ return
point(p.x*cos(rad)-p.y*sin(rad), p.x*sin(rad)+p.y*cos(rad)); }

// rotate p1 by rad(in radian) CCW w.r.t origin p2
point pointRotatePoint(point p1, point p2, double rad){
     point p=pointRotate(point(p1.x-p2.x, p1.y-p2.y), rad);
     p.x+=p2.x; p.y+=p2.y;
     return p;
}
//Returns true if _x1==_x2
bool floatEqual(double _x1, double _x2){ return fabs(_x1-_x2)<EPS; }
//Returns true if _x1>_x2
bool floatGreater(double _x1, double _x2){ return _x1-EPS>_x2; }
//Returns true if _x1<_x2
bool floatLesser(double _x1, double _x2){ return _x1+EPS<_x2; }

/// LINE
// Equation of a line: ax+by+c=0
struct line{
     double a, b, c;
};
// Returns equation of a line passing through p1 and p2
line pointsToLine(point p1, point p2){
     line l;
     if(fabs(p1.x-p2.x)<EPS){      // Vertical line
          l.a=1; l.b=0.0; l.c=-p1.x;
     }
     else{
          l.a=-(p1.y-p2.y)/(p1.x-p2.x);
          l.b=1;
          l.c=-(l.a*p1.x)-p1.y;
     }
     return l;
}
// Returns true if two lines are parallel
bool areParallel(line l1, line l2){ return fabs(l1.a-l2.a)<EPS &&
fabs(l1.b-l2.b)<EPS; }
// returns true if a point p is on the segment whose end points are a, b
///NOTE: Point P must on the line made from point a and b
bool onSegment(point p, point a, point b){
     return p.x>=min(a.x, b.x) && p.x<=max(a.x, b.x) && p.y>=min(a.y, b.y)
     && p.y<=max(a.y, b.y);
}
// returns true(also the point) if two lines intersect
// intersected point is stored in p
bool doIntersect(point p1, point p2, point p3, point p4, point &p){
line l1=pointsToLine(p1, p2);line l2=pointsToLine(p3, p4);
     if(areParallel(l1, l2)){
          if(onSegment(p1, p3, p4)) p.x=p1.x, p.y=p1.y;
          else if(onSegment(p2, p3, p4)) p.x=p2.x, p.y=p2.y;
          else return 0;
          return 1;
     }
     p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a * l2.b); //
     by solving two line equation
     if(fabs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);  // to avoid divide
     by zero
     else p.y = -(l2.a * p.x + l2.c);
     return 1;
}
/// VECTOR
struct vec{
     double x, y;
     vec(){x=y=0.0;}
     vec(double _x, double _y){x=_x; y=_y;}
};

// convert 2 points to vector p1->p2
vec toVector(point p1, point p2){ return vec(p2.x-p1.x, p2.y-p1.y); }

// nonnegative s = [<1 .. 1 .. >1]
//                        shorter.same.longer
```

```
vec scale(vec v, double x){ return vec(v.x*x, v.y*x); }

// translate p according to v
// i.e: p is transferred (from its current position) |v| unit in the
direction of v
point translate(point p, vec v){ return point(p.x+v.x, p.y+v.y); }

// Dot product of vector a, b: (axi+ayi).(bxi+byi)=ax*bx+ay*by
double dotProduct(vec a, vec b){ return a.x*b.x+a.y*b.y; }

// Using determinant rule
double crossProduct(vec a, vec b) { return a.x * b.y - a.y * b.x; }

// |v|^2=v.x*v.x+v.y*v.y
double norm_sq(vec v){ return v.x*v.x+v.y*v.y; }

// returns the distance from p to the line(_segment=0)/segment(_segment=1)
defined by -
// two points a and b (a and b must be different for line)
// the closest point (from p to line) is stored in c
double distToLineOrSeg(point p, point a, point b, point &c, bool
_segment){
     vec ap=toVector(a, p), ab=toVector(a, b);
     double u=dotProduct(ap, ab)/norm_sq(ab);
     if(_segment){
          if(u<0.0) c=a; // Closer to a
          else if(u>1.0) c=b; // Closer to b
          else c=translate(a, scale(ab, u)); // Similar to line
     }
     else c=translate(a, scale(ab, u));
     return dist(c, p);
}

// Given a point p and a line l (described by two points a and b) -
// returns the location of a reflection point r of point p when mirrored
against line l
point mirrorPoint(point p, point a, point b){
     point c, r; distToLineOrSeg(p, a, b, c, 0);
     r=translate(p, scale(toVector(p, c), 2));
     return r;
}
```

```
// returns angle aob in rad
double angle(point a, point o, point b){
     vec oa=toVector(o, a), ob=toVector(o, b);
     return acos(dotProduct(oa, ob)/(sqrt(norm_sq(oa)*norm_sq(ob))));
}

// returns c.c.w. angle from x axis to the vector in rad
double angleWRTx(vec v){
     double rad=atan2(v.y, v.x);
     if(rad<0) rad=2.0*PI+rad;
     return rad;
}

// note: to accept collinear points as CCW, change '> 0' to '>=0'
// returns true if point r is on the left side of line pq
bool ccw(point p, point q, point r){ return crossProduct(toVector(p, q),
toVector(p, r))>0; }

// returns true if point r is on the same line as the line pq
bool collinear(point p, point q, point r) { return
fabs(crossProduct(toVector(p, q), toVector(p, r))) < EPS; }

/// TRIANGLE

// returns angle between edges a and b (length of the edges of triangles
are a, b, c)
double angleFromLength(double a, double b, double c){ return
acos((a*a+b*b-c*c)/(2.0*a*b)); }

// area of the triangle with edge lengths a, b, c
double areaFromLength(double a, double b, double c){
     double s=(a+b+c)/2.0;
     return sqrt(s*(s-a)*(s-b)*(s-c));
}
double areafrompoint(point p1,point p2,point p3)
{
double a = dist(p1,p2);double b = dist(p2,p3);double c = dist(p3,p1);
return areaFromLength(a,b,c);
}

/// CIRCLE
```

```
// returns the radius of the circle surrounding the triangle
double rCircumCircle(double ab, double bc, double ca){
     return ab * bc * ca / (4.0 * areaFromLength(ab, bc, ca));
}
double rCircumCircle(point a, point b, point c){
     return rCircumCircle(dist(a, b), dist(b, c), dist(c, a));
}

// returns the radius of the circle surrounded by the triangle
double rInCircle(double ab, double bc, double ca){
     return areaFromLength(ab, bc, ca) / (0.5 * (ab+bc+ca));
}
double rInCircle(point a, point b, point c){
     return rInCircle(dist(a, b), dist(b, c), dist(c, a));
}

// returns 1 if there is an inCircle(circle surrounded by the triangle)
center
// if this function returns 1, ctr will be the inCircle center
// and r is the same as rInCircle
int inCircle(point p1, point p2, point p3, point &ctr, double &r){
     r=rInCircle(p1, p2, p3);
     if(fabs(r)<EPS) return 0; // no inCircle center
     double ration = dist(p1, p2) / dist(p1, p3);
     point p = translate(p2, scale(toVector(p2, p3), ration / (1 +
     ration)));
     ration = dist(p2, p1) / dist(p2, p3);
     p = translate(p1, scale(toVector(p1, p3), ration / (1 + ration)));
     doIntersect(p1, p, p2, p, ctr); // get their intersection point
     return 1;
}

// returns the overlapped area(union) of two circles
// first circle center c1 and radius r1
// second circle center c2 and radius r2
double overlapCircleArea(point c1, double r1, point c2, double r2){
     double d, rad1, rad2, area1, area2, chord;
     if(r1>r2){
          swap(c1, c2);swap(r1, r2);
     }
     d=dist(c1, c2);
     if(d>=r1+r2) return 0;
     if(d<=r2-r1) return PI*r1*r1;
     rad1=angleFromLength(r1, d, r2), rad2=angleFromLength(r2, d, r1);
     chord=2.0*r1*sin(rad1);
     area1=(r1*r1*rad1)-((2*rad1>PI)?-1:1)*areaFromLength(r1, r1, chord);
     area2=(r2*r2*rad2)-areaFromLength(r2, r2, chord);
     return area1+area2;
}

/// POLYGON

//returns true if the point p is inside polygon(first point=last point)
bool inPolygon(point poly[], int n, point p){
     int i, j, k;
     double totAngle=0;
     for(i=0; i<n; ++i) if(poly[i].x==p.x && poly[i].y==p.y) return true;
     //if it overlaps with some point
     for(i=1; i<n; ++i) if(collinear(p, poly[i-1], poly[i]) &&
     onSegment(p, poly[i-1], poly[i])) return true; //if it is in some
     edges of the polygon
     for(i=1; i<n; ++i){
          if(ccw(poly[i-1], p, poly[i])) totAngle-=angle(poly[i-1], p,
          poly[i]);
          else totAngle+=angle(poly[i-1], p, poly[i]);
     }
     return fabs(totAngle-2.*PI)<EPS;
}

/// CONVEX HULL
double cross(point p1, point p2, point p3){ return
(p2.x-p1.x)*(p3.y-p1.y)-(p2.y-p1.y)*(p3.x-p1.x); }
// Returns the Hull created by the n points of ara[]
// Does not take linear points in the hull
vector< point > ConvexHull(int n, point ara[]){
     int i, j, k;
     vector< point > cnvx(2*n);
     sort(ara, ara+n);
     for(i=0, k=0; i<n; ++i){
          while(k>=2 && cross(cnvx[k-2], cnvx[k-1], ara[i])<=0) k--;
          cnvx[k++]=ara[i];
     }
     for(i=n-2, j=k+1; i>=0; --i){
          while(k>=j && cross(cnvx[k-2], cnvx[k-1], ara[i])<=0) k--;
```

```
            cnvx[k++]=ara[i];
      }
      cnvx.resize(k-1);   // Not taking the last point as first point
      return cnvx;
}

// returns the maximum area of a triangle created by three points on the
convex hull
double maxTriangleArea(vector< point > cnvx){
      int sz=cnvx.size();
      if(sz<3) return 0;
      int a=0, b=(a+1)%sz, c=(b+1)%sz;
      double area, narea, ans=0;
      while(a<sz){
            area=areafrompoint(cnvx[a], cnvx[b], cnvx[c]);
            while(1){
                  while(1){
                        c=(c+1)%sz;
                        narea=areafrompoint(cnvx[a], cnvx[b], cnvx[c]);
                        if(narea<area){
                              c=(c-1+sz)%sz;
                              break;
                        }
                        area=narea;
                  }
                  b=(b+1)%sz;
                  narea=areafrompoint(cnvx[a], cnvx[b], cnvx[c]);
                  if(narea<area){
                        b=(b-1+sz)%sz;
                        break;
                  }
                  area=narea;
            }
            ans=max(ans, area);
            a++;
            if(a==b) b=(b+1)%sz;
            if(b==c) c=(c+1)%sz;
      }
      return ans;
}

int main(){
}

/// Pick's Theorem
/*

struct Point {
        int x, y;
        Point(){}
};
struct Vector{
        LL x, y;
        Vector(){}
        Vector(Point a, Point b) { x = b.x-a.x, y = a.y-b.y; }
        LL cross(Vector &B) {
                return x * B.y - y * B.x;
        }
};
LL parallelogramArea(Point a, Point b, Point c) {
        Vector A(b,a), B(b,c);
        return A.cross(B);
}
int Case;
Point P[10000+7];

int latticePoints(Point a, Point b)
{
        b.x -= a.x; b.y -= a.y;
        if( b.x < 0 ) b.x = -b.x;
        if( b.y < 0 ) b.y = -b.y;
        return __gcd(b.x,b.y);
}
int main()
{
            LL n ;
      while(  cin >> n ){
        if(n==0)break;
        LL twoA = 0; /// twice of polygon area
        LL B = n;
        scanf("%d %d",&P[0].x,&P[0].y);
        for(int i=1; i<n; i++) {
                scanf("%d %d",&P[i].x,&P[i].y);
```

```
            twoA += parallelogramArea( P[0],P[i-1],P[i] );
            B += latticePoints( P[i-1], P[i] ) - 1;
        }
        B += latticePoints( P[n-1], P[0] ) - 1;
        twoA = abs(twoA);

        cout << (( twoA - B + 2 ) >> 1) << endl;
      }
    //    printf("Case %d: %lld\n", Case, ( twoA - B + 2 ) >> 1 );
}
*/
```

6 String

6.1 kmp + plaindromic Tree

```
/// performs O(n) actions.
vector<int> prefix_function(string s)   /// Longest Length of a
{
    int n = (int)s.length();           /// prefix in a string
    vector<int> pi(n);
    for (int i = 1; i < n; i++)
    {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
void computeLPSArray(char* pat, int M, int* lps)
{
    int len = 0;
    lps[0] = 0; /// lps[0] is always 0
    int i = 1;
    while (i < M)
    {
        if (pat[i] == pat[len])
        {
            len++;
            lps[i] = len;
            i++;
        }
        else /// (pat[i] != pat[len])
        {
            if (len != 0)
            {
                len = lps[len - 1];
            }
            else /// if (len == 0)
            {
                lps[i] = 0;
                i++;
            }
        }
    }
}
void KMPSearch(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);
    int lps[M];
    computeLPSArray(pat, M, lps);

    int i = 0; /// index for txt[]
    int j = 0; /// index for pat[]
    while (i < N)
    {
        if (pat[j] == txt[i])
        {
            j++;
            i++;
        }
        if (j == M)   /// printf("Found pattern at index %d ", i - j);
        {
            j = lps[j - 1];
            ///      counT++; count for number of pattern match
        }
        else if (i < N && pat[j] != txt[i])   /// mismatch after j matches
        {
            if (j != 0)
                j = lps[j - 1];
```

```
            else
                i = i + 1;
        }
    }
}
void cnounT_number_of_occurrences_of_each_preffix()
{
    int ans = s.length();
    for(int i = 1; i < s.length() ; i ++ )
    {
        int j = pi[i];
        while(j > 0 )
        {
            ans ++;
            j = pi[j-1];
            ans %= MOD;
        }
    }
}

int kmp_process(string text,string pattern) ///  search for the longest
prefix of pattern in text.
{
    int j=0;
    int n = text.size();
    int m = pattern.size();
    vector<int>preffix = prefix_function(pattern);
    for(int i=0;i<n;i++)
    {
        if(text[i]==pattern[j])
            j++;
        else
        {
            while(j>0)
            {
                j = preffix[j-1];
                if(text[i]==pattern[j])
                {
                    j++;
                    break;
                }
            }
        }
    }
    return j;
}
const int N = 1e5+10;
int ans[N];
int tree[N][26], idx ;
int len[N], link[N], t ;
int occurrence[N];
/// char s[N] ; /// 1-indexed
string s ;

void Init()
{
    memset(tree,0,sizeof tree);
    memset(link,0,sizeof link);
    memset(len,0,sizeof len);

    len[1] = -1, link[1] = 1 ;
    len[2] = 0, link[2] = 1 ;
    idx = t = 2 ;
}

void extend(int p)
{
    while(s[p-len[t]-1] != s[p] )
        t = link[t] ;
    int x = link[t], c = s[p] - 'a' ;
    while(s[p-len[x]-1]!=s[p])
        x = link[x] ;

    if(!tree[t][c])
    {
        tree[t][c] = ++idx;
        len[idx] = len[t] + 2 ;
        link[idx] = len[idx] == 1 ? 2 : tree[x][c];
        ans[idx] = 1 + ans[link[idx]];
    }
    t = tree[t][c];
    occurrence[t]++;
}
int main()
```

```
{
    int tt ;
    cin >> tt;
    for(int i=1;i<=tt;i++)
    {
        cin >> s ;
        Init();
        int counT = 0 ;
        s = '#'+s;
        for(int j=1;j<s.size();j++)
        {
            extend(j);
          //  counT += ans[t] ;
        }
        printf("Case #%d: %d\n",i,idx-2); /// Distinct palindrome
  ///  cout << counT << endl ; /// Not Distinct

    for(int i=idx;i>2;i--)
    {
        occurrence[link[i]] += occurrence[i];
    }
    for(int i=3;i<=idx;i++)
        cout << occurrence[i] << " ";
    }
}
```

6.2 Aho

```
const int N = 250004;
const int M = 505;
int n, name_of_node, cnt;
int res[N];
int node[N][27];
int fail[N];
int path[N];
int end_node[N];
char txt[1000006], pat[M];
void init()
{
  name_of_node = 0;
  cnt = 0;
  memset(path, 0, sizeof path);
  memset(fail, 0, sizeof fail);
  memset(res, 0, sizeof res);
  memset(node, -1, sizeof node);
}

void Insert(char s[], int pos)
{
  int now = 0;
  int len = strlen(s);
  for (int i = 0; i < len; i++) {
    if (node[now][s[i] - 'a'] == -1) {
      node[now][s[i] - 'a'] = ++name_of_node;
    }
    now = node[now][s[i] - 'a'];
  }
  end_node[pos] = now;
}

void failure()
{
  queue<int> Q;
  for (int i = 0; i < 26; i++) {
    if (~node[0][i])
      Q.push(node[0][i]);
    else node[0][i] = 0;
  }

  while (!Q.empty()) {
    int u = Q.front();
    Q.pop();
    for (int i = 0; i < 26;  i++) {
      int v = node[u][i];
      if (~v) {
        Q.push(v);
        fail[v] = node[fail[u]][i];
        path[++cnt] = v;
      }
      else {
        node[u][i] = node[fail[u]][i];
      }
    }

  }
```

```
  return;
}

void aho_corasick(char s[])
{
  int now = 0;
  int len = strlen(s);
  for (int i = 0; i < len; i++) {
    now = node[now][s[i] - 'a'];
    res[now]++;
  }
  for (int i = cnt; i >= 1; i--) {
    res[fail[path[i]]] += res[path[i]];
  }
}

int main()
{
  int T;
  scanf("%d", &T);
  //T = 1;
  for (int cs = 1; cs <= T; cs++) {
    init();
    scanf("%d", &n);
    scanf("%s", txt);
    for (int i = 0; i < n; i++) {
      scanf("%s", pat);
      Insert(pat, i);
    }
    failure();
    aho_corasick(txt);
    printf("Case %d:\n", cs);
    for (int i = 0; i < n; i++) {
      printf("%d\n", res[end_node[i]]);
    }
  }
  return 0;
}
```

6.3 Z-Algorithm

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

vector<int> z_algo(string s) // from tushar roy video
{
    int n = s.length();
    vector <int> z(n);

    int l = 0, r = 0;
    for (int k = 1; k < n; k++) {
        if (k > r) {
            l = r = k;
            while (r < n && s[r] == s[r - l]) r++;

            z[k] = r - l;
            r--;
        }
        else {
            // inside box
            int i = k - l;
            if (z[i] < r - k + 1) z[k] = z[i];
            else {
                l = k;
                while (r < n && s[r] == s[r - l]) r++;
                z[k] = r - l;
                r--;
            }
        }
    }
    return z;
```

```
}
```

7 Matrix Expo

7.1 Matrix Expo

```
struct matrix
{
ll x,y;ll tb[23][23];
void clear(){MEM(tb,0);}
}aa,ee;
matrix mul(matrix A,matrix B)
{ matrix C;
f1(i,d) f1(j,d) C.tb[i][j] = 0 ;
f1(i,d)f1(j,d)f1(k,d)
C.tb[i][j] = (C.tb[i][j] + (A.tb[i][k] * B.tb[k][j]) % M)% M;
return C;}
matrix Pow(matrix A,int p)
{ matrix res = ee ;
while(p){ if(p&1)res = mul(res,A);
A = mul(A,A);p >>=1;}return res;
}
```

8 miscellaneous

8.1 miscellaneous

```
#define MSET(x) memset(x, 0x3f, sizeof(x));
const int Max = 1001 * 105;
ll n, W, value, weight, ans;
int main()
{ cin >> n >> W; vector<ll>knapSack(Max, INT_MAX); knapSack[0] = 0;
  ans = 0; for (int i = 1; i <= n; i++) {
    cin >> weight >> value;
    for (int j = ans; j >= 0; j--) {
      if (knapSack[j] + weight <= W && knapSack[j + value] > knapSack[j] +
      weight) {
        knapSack[j + value] = knapSack[j] + weight; ans = max(ans, j +
        value);
      }
    }
  }
  cout <<  ans << endl;
}
////////////////////////////////////////////////
const int NX = 1000; int input[ NX + 5] , n ;
void LIS_with_set() {
  multiset < int > lis ;
  multiset < int > :: iterator  it ; scanf("%d", &n);
  for ( int i = 0 ; i < n ; i++ ) {
    scanf("%d", &input[i]); lis.insert( input[i]);
    it = lis.upper_bound( input[i]); if ( it != lis.end()) lis.erase(it);
  }
  cout << lis.size() << endl ;
}
//------------------------------------------------
/*Given sum find n */int main() {
  int n, m, i, sqr, a, b;
  scanf("%d %d", &n, &m); m %= n * (n + 1) / 2;
  sqr = (sqrt(m * 8.0 + 1) - 1) / 2; sqr = sqr * (sqr + 1) / 2;
  printf("%d\n", m - sqr);
}
/*----------Edit Distance---------------------------------------*/
int main() {
  dp[0][0] = 0; for (int i = 1; i <= 100 ++i)
  {dp[i][0] = dp[0][i] = i;} scanf("%s%s", a, b); int n = strlen(a);
  int m = strlen(b); for (int i = 1; i <= n; ++i)
    for (int j = 1; j <= m; ++j)dp[i][j] = min(min(dp[i - 1][j],
                                               dp[i][j - 1]) + 1, dp[i -
                                               1][j - 1] + (a[i - 1] != b[j
                                               - 1]));
}
/*----------TSP-------------------------*/
int VISITED_ALL = (1 << n) - 1;
// mask = friends I already visited,At = last visited friend
int tsp(int mask, int pos) {
  if (mask == VISITED_ALL) return dist[pos][0];
  if (dp[mask][pos] != -1) return dp[mask][pos];
//  Now from current node, we will try to go to every other node and take
the min ans
  int ans = INT_MAX; for (int city = 0; city < n; city++) {
    if ( (mask & (1 << city ) ) == 0) {
      int newAns = dist[pos][city] + tsp(mask | (1 << city), city);
      ans = min(ans, newAns);
```

```
    }
  } return dp[mask][pos] = ans;
}
/*----------------------------Kinght Distance---------------*/
ll dist(ll x1, ll y1, ll x2, ll y2) {
ll dx = abs(x2 - x1);ll dy = abs(y2 - y1);ll lb = (dx + 1) / 2;
lb = max(lb, (dy + 1) / 2);lb = max(lb, (dx + dy + 2) / 3);
while ((lb & 1) != ((dx + dy) & 1)) lb++;if (dx == 1 && dy == 0) return 3;
if (dy == 1 && dx == 0) return 3;if (dx == 2 && dy == 2) return 4;return
lb;}
int n;ll dp[(1 << 15) + 2], d[20][20];pair <ll, ll> a[20], b[20];
ll f(int idx, int mask) {if (idx == n) return 0LL;ll &ret = dp[mask];
if (ret != -1) return ret;ret = 10000000000000000LL;
for (int i = 0; i < n; ++i) if (checkBit(mask, i) == 0) ret = min(ret,
d[idx][i] + f(idx + 1, setBit(mask, i)));return ret;}int main () {
int cs = 0;while (scanf("%d", &n) && n) {if (n == 0) break;
for (int i = 0; i < n; ++i) scanf("%lld %lld", &a[i].first, &a[i].second);
for (int i = 0; i < n; ++i) scanf("%lld %lld", &b[i].first,&b[i].second);
for (int i = 0; i < n; ++i) for (int j = 0; j < n; ++j) d[i][j]
=dist(a[i].first, a[i].second, b[j].first, b[j].second);
for (int i = 0, j = 1 << n; i < j; ++i) dp[i] = -1;printf("%d. %lld\n",
++cs, f(0, 0));}}
/*-cyclic shif kore minimum string output-*/
vector<string> duval(string const& s){int n = s.size();int i = 0;
vector<string> factorization;while (i < n) {
int j = i + 1, k = i;
while (j < n && s[k] <= s[j]) {
if (s[k] < s[j]) k = i;else k++;j++;}while (i <= k) {
factorization.push_back(s.substr(i, j - k));i += j - k;}}return
factorization;}
int min_cyclic_string(string s)
{s += s;int n = s.size();int i = 0, ans = 0;while (i < n / 2) {ans = i;
 int j = i + 1, k = i;while (j < n && s[k] <= s[j]) {
if (s[k] < s[j]) k = i;else k++;j++;} while (i <= k) {i += j - k;}}return
ans + 1;
//dbg(ans);//return s.substr(ans, n / 2);}
/*every colum and every thekeektai value nibo*/
int n;int a[22][22];int dp[20][1 << 16]; int Set(int N, int pos) {
return N = N | (1 << pos);}bool check(int N, int pos) {
return (bool) (N & (1 << pos));}
int solve(int groom, int mask){
if (groom >= n) return 0;int &ret = dp[groom][mask];
 if (ret != -1) return ret;int mx = 0;
for (int i = 0; i < n; i++) {if (!check(mask, i)) { // if not married
int ans = a[groom][i] + solve(groom + 1, Set(mask, i)); mx = max(mx, ans);
}}
return ret = mx; }

/*rod cutting*/
/// O(n^2)
vector<int> rods;
int cutRodDp(int price[], int n){int dp[n + 1];
int lastRod[n + 1];dp[0] = 0;
for (int i = 1; i <= n; i++) {int mx = INT_MIN;
int best_rod_len = -1;for (int j = 0; j < i; j++) {
if (mx < price[j] + dp[i - j - 1]) {mx = price[j] + dp[i - j - 1];
best_rod_len = j; }//mx = max(mx, price[j] + dp[i - j - 1]);}
dp[i] = mx;lastRod[i] = best_rod_len + 1;}
for (int i = n; i > 0; i -= lastRod[i]) {rods.push_back(lastRod[i]);}
return dp[n];}
int main{ cout << cutRodDp(a, n);cout << " { ";for (auto x : rods)
 cout << x << " ";cout << "}\n";}
 /*-----------------CRT----------*/
int primes[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47};
ll binomial_coefficient(int n, int k)
{
    if(n < k) return 0;
    ll ans = 1;
    k = k < (n - k) ? k : (n - k);
    for(int i = 1; i <= k; i++,n--)
    {
        if(n % i == 0)
        {
            ans *= n/i;
        }
        else if(ans % i == 0)
        {
            ans = (ans / i) * n;
        }
        else
        {
            ans = (ans * n)/i;
        }
    }
```

```
    return ans;
}

LL lucas_theorem(int n, int k, int p)
{ LL ans = 1;
    while(k > 0)
    {
int tmp_n = n % p;int tmp_k = k % p;
ans *= binomial_coefficient(tmp_n, tmp_k) % p;ans %= p;n /= p;k /= p;
    }
    return ans;
}

LL get_reminder_squarefree(int n, int k, int m)
{
    LL ans = -1; LL last = 1;
    int primes_length = sizeof(primes)/sizeof(int);
    if(m == 1) return 0;
    for(int i = 0; i < primes_length && m > 1; i++)
    {
        if(m % primes[i] != 0) continue;

        LL rem = lucas_theorem(n, k, primes[i]);

        if(ans == -1) ans = rem;
        else
        {
            for(int j = 0; j < 50; j++)
            {
                if( (ans + (last * j)) % primes[i] == rem)
                {
                    ans = (ans + (last * j));
                    break;
                }
            }
        }
        last *= primes[i];
        m /= primes[i];
 }
return ans;}int main(){
int t;cin >> t;while(t--){int n,m,r;cin >> n >> r >> m;
cout << get_reminder_squarefree(n, r, m) << endl;}}
```

```
/*
2. Numbers are divisible by 3 if the sum of all the individual digits is
evenly divisible by 3. For
example, the sum of the digits for the number 3627 is 18, which is evenly
divisible by 3 so the
number 3627 is evenly divisible by 3.
3. Whole numbers are divisible by 4 if the number formed by the last two
individual digits is
evenly divisible by 4. For example, the number formed by the last two
digits of the number
3628 is 28, which is evenly divisible by 4 so the number 3628 is evenly
divisible by 4.
4. Numbers are evenly divisible by 5 if the last digit of the number is 0
or 5.
5. Numbers are evenly divisible by 6 if they are evenly divisible by both
2 AND 3. Even numbers
are always evenly divisible by 2. Numbers are evenly divisible by 3 if the
sum of all the
individual digits is evenly divisible by 3. For example, the sum of the
digits for the number 3627
is 18, which is evenly divisible by 3 but 3627 is an odd number so the
number 3627 is not evenly
divisible by 6.
6. To determine if a number is divisible by 7, take the last digit off the
number, double it and
subtract the doubled number from the remaining number. If the result is
evenly divisible by 7
(e.g. 14, 7, 0, -7, etc.), then the number is divisible by seven. This may
need to be repeated
several times.
Example: Is 3101 evenly divisible by 7?
310 - take off the last digit of the number which was 1
-2 - double the removed digit and subtract it
---------
308 - repeat the process by taking off the 8
-16 - and doubling it to get 16 which is subtracted
---------
14 - the result is 14 which is a multiple of 7
7. Numbers are divisible by 8 if the number formed by the last three
individual digits is evenly
```

```
divisible by 8. For example, the last three digits of the number 3624 is
624, which is evenly
divisible by 8 so 3624 is evenly divisible by 8.
8.  Numbers are divisible by 9 if the sum of
all the individual digits is evenly divisible by 9. For example, the last
sum of the digits of the
number 3627 is 18, which is evenly divisible by 9 so 3627 is evenly
divisible by 9.

11        ,       11       11
 145816
1{4+5{8+1{6= -11   11    (-11%11=0)     11
12        3  4       12

(area of trapezium)
Area =(a+b)/(a-b)×√(s-a)(s-b)(s-b-c)(s-b-d)
S =( a+b+c+d)/2
a = long parallel side  b = short parallel side   c = non parallel side
d = non parallel side
*/
```

8.2 Bitmask

```
/*
set    N=N | (1<<pos)
reset  (N & (1<<pos))
clear  number &= ~(1UL << n)
toggle number ^= 1UL << n
changing nth bit to x(0,1) number ^= (-x ^ number) & (1UL << n)
is power of 2 : x && (!(x&(x-1)))
int bitmask(int r,int mask)
{
    if(r>=n)    return 0;
    if(dp[r][mask]!=-1)    return dp[r][mask];
    int mx=0;
    for(int i=0;i<n;i++)
    {
        if(check(mask,i)==0)
        {
           int ans=a[r][i]+bitmask(r+1,Set(mask,i));
           mx=max(mx,ans);
        }
    }
    return dp[r][mask]=mx;
}
*/
unsigned int nextPowerOf2(unsigned int n)
{
    n--;n |= n >> 1;n |= n >> 2;
    n |= n >> 4;n |= n >> 8;n |= n >> 16;n++;
    return n;
}
int highestPowerof2(int n)
{
   int p = (int)log2(n);
   return (int)pow(2, p);
}
int highestOneBit(int i) {
i |= (i >>  1);i |= (i >>  2);i |= (i >>  4);
    i |= (i >>  8);i |= (i >> 16);
    return i - ((unsigned)i >> 1);
}
```