KUET_Effervescent Team Notebook

June 1, 2022

Md. Mehrab Hossain Opi, Arnob Sarker,
Sharif Minhazul Islam

1 Data Structure

1.1 MO with Update [43 lines]

```
//1 indexed
//Complexity:O(S × Q + Q × N^2/S^2)
//S = (2*n^2)^(1/3)
const int block_size = 2720;  // 4310 for 2e5
const int mx = 1e5 + 5;
struct Query {
  int L, R, T, id;
  Query() {}
  Query(int _L, int _R, int _T, int _id) : L(_L),
    R(_R), T(_T), id(_id) {}
  bool operator<(const Query &x) const {
    if (L / block_size == x.L / block_size) {
      if (R / block_size == x.R / block_size)
    return T < x.T;
      return R / block_size < x.R / block_size;
    }
    return L / block_size < x.L / block_size;
  }
} Q[mx];
struct Update {
  int pos;
  int old, cur;
  Update(){};
  Update(int _p, int _o, int _c) : pos(_p),
    old(_o), cur(_c){};
} U[mx];
int ans[mx];
inline void add(int id) {}
inline void remove(int id) {}
inline void update(int id, int L, int R) {}
inline void undo(int id, int L, int R) {}
inline int get() {}
void MO(int nq, int nu) {
  sort(Q + 1, Q + nq + 1);
  int L = 1, R = 0, T = nu;
  for (int i = 1; i <= nq; i++) {
    Query q = Q[i];
    while (T < q.T) update(++T, L, R);
    while (T > q.T) undo(T--, L, R);
    while (L > q.L) add(--L);
    while (R < q.R) add(++R);
    while (L < q.L) remove(L++);
    while (R > q.R) remove(R--);
    ans[q.id] = get();
  }
}
```

1.2 MO [28 lines]

```
const int N = 2e5 + 5;
const int Q = 2e5 + 5;
const int SZ = sqrt(N) + 1;
struct qry {
  int l, r, id, blk;
  bool operator<(const qry& p) const {
    return blk == p.blk ? r < p.r : blk < p.blk;
  }
};
qry query[Q];
ll ans[Q];
void add(int id) {}
void remove(int id) {}
ll get() {}
int n, q;
void MO() {
  sort(query, query + q);
  int cur_l = 0, cur_r = -1;
  for (int i = 0; i < q; i++) {
    qry q = query[i];
    while (cur_l > q.l) add(--cur_l);
    while (cur_r < q.r) add(++cur_r);
    while (cur_l < q.l) remove(cur_l++);
    while (cur_r > q.r) remove(cur_r--);
    ans[q.id] = get();
  }
}
/* 0 indexed. */
```

1.3 SegmentTree [74 lines]

```
/*edit:data,combine,build
  check datatype*/
template<typename T>
struct SegmentTree {
#define lc (C << 1)
#define rc (C << 1 | 1)
#define M ((L+R)>>1)
  struct data {
    T sum;
    data() :sum(0) {};
  };
  vector<data>st;
  vector<bool>isLazy;
  vector<T>lazy;
  int N;
  SegmentTree(int _N) :N(_N) {
    st.resize(4 * N);
    isLazy.resize(4 * N);
    lazy.resize(4 * N);
  }
  void combine(data& cur, data& l, data& r) {
    cur.sum = l.sum + r.sum;
  }
  void push(int C, int L, int R) {
    if (!isLazy[C]) return;
    if (L != R) {
      isLazy[lc] = 1;
      isLazy[rc] = 1;
      lazy[lc] += lazy[C];
      lazy[rc] += lazy[C];
    }
    st[C].sum = (R - L + 1) * lazy[C];
    lazy[C] = 0;
    isLazy[C] = false;
  }
  void build(int C, int L, int R) {
    if (L == R) {
      st[C].sum = 0;
      return;
    }
    build(lc, L, M);
    build(rc, M + 1, R);
    combine(st[C], st[lc], st[rc]);
  }
  data Query(int i, int j, int C, int L, int R) {
    push(C, L, R);
    if (j < L || i > R || L > R) return data();
    // default val 0/INF
    if (i <= L && R <= j) return st[C];
    data ret;
    data d1 = Query(i, j, lc, L, M);
    data d2 = Query(i, j, rc, M + 1, R);
    combine(ret, d1, d2);
    return ret;
  }
  void Update(int i, int j, T val, int C, int L,
    int R) {
    push(C, L, R);
    if (j < L || i > R || L > R) return;
    if (i <= L && R <= j) {
      isLazy[C] = 1;
      lazy[C] = val;
      push(C, L, R);
      return;
    }
    Update(i, j, val, lc, L, M);
    Update(i, j, val, rc, M + 1, R);
    combine(st[C], st[lc], st[rc]);
  }
  void Update(int i, int j, T val) {
    Update(i, j, val, 1, 1, N);
  }
  T Query(int i, int j) {
    return Query(i, j, 1, 1, N).sum;
  }
};
```

2 Dynamic Programming

3 Flow

3.1 Dinic [72 lines]

```
/*.Complexity: O(V^2 E)
  .Call Dinic with total number of nodes.
  .Nodes start from 0.
  .Capacity is long long data.
  .make graph with create edge(u,v,capacity).
  .Get max flow with maxFlow(src,des).*/
#define eb emplace_back
struct Dinic {
  struct Edge {
```

```
  int u, v;
  ll cap, flow = 0;
  Edge() {}
  Edge(int u, int v, ll cap) :u(u), v(v),
  cap(cap) {}
};
int N;
vector<Edge>edge;
vector<vector<int>>adj;
vector<int>d, pt;
Dinic(int N) :N(N), edge(0), adj(N), d(N), pt(N)
  {}
void addEdge(int u, int v, ll cap) {
  if (u == v) return;
  edge.eb(u, v, cap);
  adj[u].eb(edge.size() - 1);
  edge.eb(v, u, 0);
  adj[v].eb(edge.size() - 1);
}
bool bfs(int s, int t) {
  queue<int>q({ s });
  fill(d.begin(), d.end(), N + 1);
  d[s] = 0;
  while (!q.empty()) {
    int u = q.front();q.pop();
    if (u == t) break;
    for (int k : adj[u]) {
      Edge& e = edge[k];
      if (e.flow<e.cap && d[e.v]>d[e.u] + 1) {
        d[e.v] = d[e.u] + 1;
        q.emplace(e.v);
      }
    }
  }
  return d[t] != N + 1;
}
ll dfs(int u, int T, ll flow = -1) {
  if (u == T || flow == 0) return flow;
  for (int& i = pt[u];i < adj[u].size();i++) {
    Edge& e = edge[adj[u][i]];
    Edge& oe = edge[adj[u][i] ^ 1];
    if (d[e.v] == d[e.u] + 1) {
      ll amt = e.cap - e.flow;
      if (flow != -1 && amt > flow) amt = flow;
      if (ll pushed = dfs(e.v, T, amt)) {
        e.flow += pushed;
        oe.flow -= pushed;
        return pushed;
      }
    }
  }
  return 0;
}
ll maxFlow(int s, int t) {
  ll total = 0;
  while (bfs(s, t)) {
    fill(pt.begin(), pt.end(), 0);
    while (ll flow = dfs(s, t)) {
      total += flow;
    }
  }
  return total;
}

};
```

3.2 HopCroftKarp [67 lines]

```
/*.Finds Maximum Matching In a bipartite graph
  .Complexity O(E√V)
  .1-indexed
  .No default constructor
  .add single edge for (u, v)*/
struct HK {
  static const int inf = 1e9;
  int n;
  vector<int>matchL, matchR, dist;
  //matchL contains value of matched node for L
    part.
  vector<vector<int>>adj;
  HK(int n) :n(n), matchL(n + 1),
  matchR(n + 1), dist(n + 1), adj(n + 1) {
  }

  void addEdge(int u, int v) {
    adj[u].push_back(v);
  }
  bool bfs() {
    queue<int>q;
    for (int u = 1;u <= n;u++) {
      if (!matchL[u]) {
        dist[u] = 0;
        q.push(u);
      }
      else dist[u] = inf;
    }
    dist[0] = inf;///unmatched node matches with
    0.
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (auto v : adj[u]) {
        if (dist[matchR[v]] == inf) {
          dist[matchR[v]] = dist[u] + 1;
          q.push(matchR[v]);
        }
      }
    }
    return dist[0] != inf;
  }

  bool dfs(int u) {
    if (!u) return true;
    for (auto v : adj[u]) {
      if (dist[matchR[v]] == dist[u] + 1
          && dfs(matchR[v])) {
        matchL[u] = v;
        matchR[v] = u;
        return true;
      }
    }
    dist[u] = inf;
    return false;
  }
  int max_match() {
    int matching = 0;
    while (bfs()) {
      for (int u = 1;u <= n;u++) {
        if (!matchL[u])
          if (dfs(u))
            matching++;
      }
    }
    return matching;
  }
};
```

3.3 Hungarian [116 lines]

```
/* Complexity: O(n^3) but optimized
   It finds minimum cost maximum matching.
   For finding maximum cost maximum matching
   add -cost and return -matching()
   1-indexed */
struct Hungarian {
  long long c[N][N], fx[N], fy[N], d[N];
  int l[N], r[N], arg[N], trace[N];
  queue<int> q;
  int start, finish, n;
  const long long inf = 1e18;
  Hungarian() {}
  Hungarian(int n1, int n2) : n(max(n1, n2)) {
    for (int i = 1; i <= n; ++i) {
      fy[i] = l[i] = r[i] = 0;
      for (int j = 1; j <= n; ++j) c[i][j] = inf;
    }
  }
  void add_edge(int u, int v, long long cost) {
    c[u][v] = min(c[u][v], cost);
  }
  inline long long getC(int u, int v) {
    return c[u][v] - fx[u] - fy[v];
  }
  void initBFS() {
    while (!q.empty()) q.pop();
    q.push(start);
    for (int i = 0; i <= n; ++i) trace[i] = 0;
    for (int v = 1; v <= n; ++v) {
      d[v] = getC(start, v);
      arg[v] = start;
    }
    finish = 0;
  }
  void findAugPath() {
```

```
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    for (int v = 1; v <= n; ++v) if (!trace[v])
  {
      long long w = getC(u, v);
      if (!w) {
        trace[v] = u;
        if (!r[v]) {
          finish = v;
          return;
        }
        q.push(r[v]);
      }
      if (d[v] > w) {
        d[v] = w;
        arg[v] = u;
      }
    }
  }
}
void subX_addY() {
  long long delta = inf;
  for (int v = 1; v <= n; ++v) if (trace[v] == 0
  && d[v] < delta) {
    delta = d[v];
  }
  // Rotate
  fx[start] += delta;
  for (int v = 1; v <= n; ++v) if (trace[v]) {
    int u = r[v];
    fy[v] -= delta;
    fx[u] += delta;
  }
  else d[v] -= delta;
  for (int v = 1; v <= n; ++v) if (!trace[v] &&
  !d[v]) {
    trace[v] = arg[v];
    if (!r[v]) {
      finish = v;
      return;
    }
    q.push(r[v]);
  }
}
void Enlarge() {
  do {
    int u = trace[finish];
    int nxt = l[u];
    l[u] = finish;
    r[finish] = u;
    finish = nxt;
  } while (finish);
}
long long maximum_matching() {
  for (int u = 1; u <= n; ++u) {
    fx[u] = c[u][1];
    for (int v = 1; v <= n; ++v) {
      fx[u] = min(fx[u], c[u][v]);
    }
  }
  for (int v = 1; v <= n; ++v) {
    fy[v] = c[1][v] - fx[1];
    for (int u = 1; u <= n; ++u) {
      fy[v] = min(fy[v], c[u][v] - fx[u]);
    }
  }
  for (int u = 1; u <= n; ++u) {
    start = u;
    initBFS();
    while (!finish) {
      findAugPath();
      if (!finish) subX_addY();
    }
    Enlarge();
  }
  long long ans = 0;
  for (int i = 1; i <= n; ++i) {
    if (c[i][l[i]] != inf) ans += c[i][l[i]];
    else l[i] = 0;
  }
  return ans;
 }
};
```

3.4 MCMF [116 lines]

```
/*Credit: ShahjalalShohag
  .Works for both directed, undirected and with
    negative cost too
  .doesn't work for negative cycles
  .for undirected edges just make the directed
    flag false
  .Complexity: O(min(E^2 *V log V, E logV *
    flow))*/
using T = long long;
const T inf = 1LL << 61;
struct MCMF {
  struct edge {
    int u, v;
    T cap, cost;
    int id;
    edge(int _u, int _v, T _cap, T _cost, int _id)
    {
      u = _u;
      v = _v;
      cap = _cap;
      cost = _cost;
      id = _id;
    }
  };
  int n, s, t, mxid;
  T flow, cost;
  vector<vector<int>> g;
  vector<edge> e;
  vector<T> d, potential, flow_through;
  vector<int> par;
  bool neg;
  MCMF() {}
  MCMF(int _n) { // 0-based indexing
    n = _n + 10;
    g.assign(n, vector<int>());
    neg = false;
    mxid = 0;
  }
  void add_edge(int u, int v, T cap, T cost, int
    id = -1, bool directed = true) {
    if (cost < 0) neg = true;
    g[u].push_back(e.size());
    e.push_back(edge(u, v, cap, cost, id));
    g[v].push_back(e.size());
    e.push_back(edge(v, u, 0, -cost, -1));
    mxid = max(mxid, id);
    if (!directed) add_edge(v, u, cap, cost, -1,
    true);
  }
  bool dijkstra() {
    par.assign(n, -1);
    d.assign(n, inf);
    priority_queue<pair<T, T>, vector<pair<T,
    T>>, greater<pair<T, T>> > q;
    d[s] = 0;
    q.push(pair<T, T>(0, s));
    while (!q.empty()) {
      int u = q.top().second;
      T nw = q.top().first;
      q.pop();
      if (nw != d[u]) continue;
      for (int i = 0; i < (int)g[u].size(); i++) {
        int id = g[u][i];
        int v = e[id].v;
        T cap = e[id].cap;
        T w = e[id].cost + potential[u] -
    potential[v];
        if (d[u] + w < d[v] && cap > 0) {
          d[v] = d[u] + w;
          par[v] = id;
          q.push(pair<T, T>(d[v], v));
        }
      }
    }
    for (int i = 0; i < n; i++) { // update
    potential
      if (d[i] < inf) potential[i] += d[i];
    }
    return d[t] != inf;
  }
  T send_flow(int v, T cur) {
    if (par[v] == -1) return cur;
    int id = par[v];
    int u = e[id].u;
    T w = e[id].cost;
    T f = send_flow(u, min(cur, e[id].cap));
    cost += f * w;
```

```
    e[id].cap -= f;
    e[id ^ 1].cap += f;
    return f;
  }
  //returns {maxflow, mincost}
  pair<T, T> solve(int _s, int _t, T goal = inf) {
    s = _s;
    t = _t;
    flow = 0, cost = 0;
    potential.assign(n, 0);
    if (neg) {
      // run Bellman-Ford to find starting
    potential
      d.assign(n, inf);
      for (int i = 0, relax = true; i < n &&
    relax; i++) {
        for (int u = 0; u < n; u++) {
          for (int k = 0; k < (int)g[u].size();
    k++) {
            int id = g[u][k];
            int v = e[id].v;
            T cap = e[id].cap, w = e[id].cost;
            if (d[v] > d[u] + w && cap > 0) {
              d[v] = d[u] + w;
              relax = true;
            }
          }
        }
      }
      for (int i = 0; i < n; i++) if (d[i] < inf)
    potential[i] = d[i];
    }
    while (flow < goal && dijkstra()) flow +=
    send_flow(t, goal - flow);
    flow_through.assign(mxid + 10, 0);
    for (int u = 0; u < n; u++) {
      for (auto v : g[u]) {
        if (e[v].id >= 0) flow_through[e[v].id] =
    e[v ^ 1].cap;
      }
    }
    return make_pair(flow, cost);
  }
};
```

3.5 blossom [58 lines]

```
// Finds Maximum matching in General Graph
// Complexity O(NM)
// mate[i] = j means i is paired with j
// source: https://codeforces.com/blog/entry
    /92339?#comment-810242
vector<int> Blossom(vector<vector<int>>& graph) {
  //mate contains matched edge.
  int n = graph.size(), timer = -1;
  vector<int> mate(n, -1), label(n), parent(n),
    orig(n), aux(n, -1), q;
  auto lca = [&](int x, int y) {
    for (timer++; ; swap(x, y)) {
      if (x == -1) continue;
      if (aux[x] == timer) return x;
      aux[x] = timer;
      x = (mate[x] == -1 ? -1 :
    orig[parent[mate[x]]]);
    }
  };
  auto blossom = [&](int v, int w, int a) {
    while (orig[v] != a) {
      parent[v] = w; w = mate[v];
      if (label[w] == 1) label[w] = 0,
    q.push_back(w);
      orig[v] = orig[w] = a; v = parent[w];
    }
  };
  auto augment = [&](int v) {
    while (v != -1) {
      int pv = parent[v], nv = mate[pv];
      mate[v] = pv; mate[pv] = v; v = nv;
    }
  };
  auto bfs = [&](int root) {
    fill(label.begin(), label.end(), -1);
    iota(orig.begin(), orig.end(), 0);
    q.clear();
    label[root] = 0; q.push_back(root);
    for (int i = 0; i < (int)q.size(); ++i) {
      int v = q[i];
      for (auto x : graph[v]) {
        if (label[x] == -1) {
          label[x] = 1; parent[x] = v;
          if (mate[x] == -1)
            return augment(x), 1;
          label[mate[x]] = 0;
    q.push_back(mate[x]);
        }
        else if (label[x] == 0 && orig[v] !=
    orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a); blossom(v, x, a);
        }
      }
    }
    return 0;
  };
  // Time halves if you start with (any) maximal
    matching.
  for (int i = 0; i < n; i++)
    if (mate[i] == -1)
      bfs(i);
  return mate;
}
```

4 Geometry

5 Graph

5.1 2SAT [92 lines]

```
struct TwoSat {
  vector<bool>vis;
  vector<vector<int>>adj, radj;
  vector<int>dfs_t, ord, par;
  int n, intime;//For n node there will be 2*n
    node in SAT.
  void init(int N) {
    n = N;
    intime = 0;
    vis.assign(N * 2 + 1, false);
    adj.assign(N * 2 + 1, vector<int>());
    radj.assign(N * 2 + 1, vector<int>());
    dfs_t.resize(N * 2 + 1);
    ord.resize(N * 2 + 1);
    par.resize(N * 2 + 1);
  }
  inline int neg(int x) {
    return x <= n ? x + n : x - n;
  }
  inline void add_implication(int a, int b) {
    if (a < 0) a = n - a;
    if (b < 0) b = n - b;
    adj[a].push_back(b);
    radj[b].push_back(a);
  }
  inline void add_or(int a, int b) {
    add_implication(-a, b);
    add_implication(-b, a);
  }
  inline void add_xor(int a, int b) {
    add_or(a, b);
    add_or(-a, -b);
  }
  inline void add_and(int a, int b) {
    add_or(a, b);
    add_or(a, -b);
    add_or(-a, b);
  }
  inline void force_true(int x) {
    if (x < 0) x = n - x;
    add_implication(neg(x), x);
  }
  inline void add_xnor(int a, int b) {
    add_or(a, -b);
    add_or(-a, b);
  }
  inline void add_nand(int a, int b) {
    add_or(-a, -b);
  }
  inline void add_nor(int a, int b) {
    add_and(-a, -b);
  }
  inline void force_false(int x) {
    if (x < 0) x = n - x;
    add_implication(x, neg(x));
  }
  inline void topsort(int u) {
    vis[u] = 1;
    for (int v : radj[u]) if (!vis[v]) topsort(v);
```

```
    dfs_t[u] = ++intime;
  }
  inline void dfs(int u, int p) {
    par[u] = p, vis[u] = 1;
    for (int v : adj[u]) if (!vis[v]) dfs(v, p);
  }
  void build() {
    int i, x;
    for (i = n * 2, intime = 0;i >= 1;i--) {
      if (!vis[i]) topsort(i);
      ord[dfs_t[i]] = i;
    }
    vis.assign(n * 2 + 1, 0);
    for (i = n * 2;i > 0;i--) {
      x = ord[i];
      if (!vis[x]) dfs(x, x);
    }
  }
  bool satisfy(vector<int>& ret)//ret contains the
    value that are true if the graph is
    satisfiable.
  {
    build();
    vis.assign(n * 2 + 1, 0);
    for (int i = 1; i <= n * 2; i++) {
      int x = ord[i];
      if (par[x] == par[neg(x)]) return 0;
      if (!vis[par[x]]) {
        vis[par[x]] = 1;
        vis[par[neg(x)]] = 0;
      }
    }
    for (int i = 1;i <= n;i++) if (vis[par[i]])
    ret.push_back(i);
    return 1;
  }
};
```

5.2 BellmanFord [57 lines]

```
#include<bits/stdc++.h>
using namespace std;
const int mx = 1e5+6;
const int INF = 0x3f3f3f3f;
struct edge{
  int u,v;
  int cost;
};
vector<edge>e;
vector<int>path(mx);
int dist[mx];
/*
   Time-complexity: O(|V|*|E|)
   Space-complexity: O(|V|)
   To find any negative cycle assign dist[i] = 0
    for all.
   We can use floyd-warshall algorithm to find
    negative cycle too.
 *Handle LL carefully.
 */
void bellmanford(int s,int n){
  int m = e.size();
  memset(dist,0x3f3f3f3f,sizeof dist);
  dist[s] = 0;
  int x;
  for (int i = 0; i < n; ++i) {
    x = -1;
    for (int j = 0; j < m; ++j)
      if (dist[e[j].u] < INF)
        if (dist[e[j].v] > dist[e[j].u] +
    e[j].cost) {
          dist[e[j].v] = max(-INF, dist[e[j].u] +
    e[j].cost);
          path[e[j].v] = e[j].u;
          x = e[j].v;
        }
  }

  if (x == -1)
    cout << "No negative cycle from " << s;
  else {
    int y = x;
    for (int i = 0; i < n; ++i)
      y = path[y];

    vector<int> path;
    for (int cur = y; ; cur = path[cur]) {
      path.push_back(cur);
      if (cur == y && path.size() > 1)
        break;
    }
    reverse(path.begin(), path.end());

    cout << "Negative cycle: ";
    for (size_t i = 0; i < path.size(); ++i)
      cout << path[i] << ' ';
  }
}
int main(){

}
```

5.3 BridgeTree [66 lines]

```
int N, M, timer, compid;
vector<pair<int, int>> g[mx];
bool used[mx], isBridge[mx];
int comp[mx], tin[mx], minAncestor[mx];
vector<int> Tree[mx]; // Store 2-edge-connected
    component tree.(Bridge tree).
void markBridge(int v, int p) {
  tin[v] = minAncestor[v] = ++timer;
  used[v] = 1;
  for (auto& e : g[v]) {
    int to, id;
    tie(to, id) = e;
    if (to == p) continue;
    if (used[to]) minAncestor[v] =
    min(minAncestor[v], tin[to]);
    else {
      markBridge(to, v);
      minAncestor[v] = min(minAncestor[v],
    minAncestor[to]);
      if (minAncestor[to] > tin[v]) isBridge[id]
    = true;
      // if (tin[u] <= minAncestor[v]) ap[u] = 1;
    }
  }
}
void markComp(int v, int p) {
  used[v] = 1;
  comp[v] = compid;
  for (auto& e : g[v]) {
    int to, id;
    tie(to, id) = e;
    if (isBridge[id]) continue;
    if (used[to]) continue;
    markComp(to, v);
  }
}
vector<pair<int, int>> edges;
void addEdge(int from, int to, int id) {
  g[from].push_back({ to, id });
  g[to].push_back({ from, id });
  edges[id] = { from, to };
}
void initB() {
  for (int i = 0; i <= compid; ++i)
    Tree[i].clear();
  for (int i = 1; i <= N; ++i) used[i] = false;
  for (int i = 1; i <= M; ++i) isBridge[i] =
    false;
  timer = compid = 0;
}
void bridge_tree() {
  initB();
  markBridge(1, -1); //Assuming graph is
    connected.
  for (int i = 1; i <= N; ++i) used[i] = 0;
  for (int i = 1; i <= N; ++i) {
    if (!used[i]) {
      markComp(i, -1);
      ++compid;
    }
  }
  for (int i = 1; i <= M; ++i) {
    if (isBridge[i]) {
      int u, v;
      tie(u, v) = edges[i];
      // connect two componets using edge.
      Tree[comp[u]].push_back(comp[v]);
      Tree[comp[v]].push_back(comp[u]);
      int x = comp[u];
      int y = comp[v];
    }
```

```
  }
}
```

5.4 Dijkstra [33 lines]

```
#include<bits/stdc++.h>
#define ff first
#define ss second
using namespace std;
const int mx = 1e5 + 5;
using ll = long long;
using pll = pair<ll, ll>;
vector<pll>adj[mx];
int dis[mx];
bool vis[mx];
//Complexity O(V+ElogV)
void Dijkstra(int src) {
  priority_queue<pll, vector<pll>, greater<pll>
    >pq;
  pq.push({ 0,src });
  memset(dis, 0x3f3f3f3f, sizeof dis);
  memset(vis, 0, sizeof vis);
  dis[src] = 0;
  while (!pq.empty()) {
    int u = pq.top().ss;
    pq.pop();
    if (vis[u]) continue;
    vis[u] = true;
    for (auto v : adj[u]) {
      if (dis[v.ss] > dis[u] + v.ff) {
        dis[v.ss] = dis[u] + v.ff;
        pq.push({ dis[v.ss],v.ss });
      }
    }
  }
}
int main() {

}
```

5.5 LCA [46 lines]

```
const int Lg = 22;
vector<int>adj[mx];
int level[mx];
int dp[Lg][mx];
void dfs(int u) {
  for (int i = 1;i < Lg;i++)
    dp[i][u] = dp[i - 1][dp[i - 1][u]];
  for (int v : adj[u]) {
    if (dp[0][u] == v)continue;
    level[v] = level[u] + 1;
    dp[0][v] = u;
    dfs(v);
  }
}
int lca(int u, int v) {
  if (level[v] < level[u])swap(u, v);
  int diff = level[v] - level[u];
  for (int i = 0;i < Lg;i++)
    if (diff & (1 << i))
      v = dp[i][v];
  for (int i = Lg - 1;i >= 0;i--)
    if (dp[i][u] != dp[i][v])
      u = dp[i][u], v = dp[i][v];
  return u == v ? u : dp[0][u];
}
int kth(int u, int k) {
  for (int i = Lg - 1;i >= 0;i--)
    if (k & (1 << i))
      u = dp[i][u];
  return u;
}
//kth node from u to v. 0th is u.
int go(int u, int v, int k) {
  int l = lca(u, v);
  int d = level[u] + level[v] - (level[l] << 1);
  assert(k <= d);
  if (level[l] + k <= level[u]) return kth(u, k);
  k -= level[u] - level[l];
  return kth(v, level[v] - level[l] - k);
}
/*
   LCA(u,v) with root r:
   lca(u,v)^lca(u,r)^lca(v,r)
   Distance between u,v:
   level(u) + level(v) - 2*level(lca(u,v))
   */
```

5.6 SCC [43 lines]

```
/*components: number of SCC.
sz: size of each SCC.
comp: component number of each node.
Create reverse graph.
Run find_scc() to find SCC.
Might need to create condensation graph by
    create_condensed().
Think about indeg/outdeg
for multiple test cases- clear
    adj/radj/comp/vis/sz/topo/condensed.*/
vector<int>adj[mx], radj[mx];

int comp[mx], vis[mx], sz[mx], components;
vector<int>topo;
void dfs(int u) {
  vis[u] = 1;
  for (int v : adj[u])
    if (!vis[v]) dfs(v);
  topo.push_back(u);
}
void dfs2(int u, int val) {
  comp[u] = val;
  sz[val]++;
  for (int v : radj[u])
    if (comp[v] == -1)
      dfs2(v, val);
}
void find_scc(int n) {
  memset(vis, 0, sizeof vis);
  memset(comp, -1, sizeof comp);
  for (int i = 1;i <= n;i++)
    if (!vis[i])
      dfs(i);
  reverse(topo.begin(), topo.end());
  for (int u : topo)
    if (comp[u] == -1)
      dfs2(u, ++components);
}
vector<int>condensed[mx];
void create_condensed(int n) {
  for (int i = 1;i <= n;i++)
    for (int v : adj[i])
      if (comp[i] != comp[v])
        condensed[comp[i]].push_back(comp[v]);
}
```

6 Math

6.1 FFT [85 lines]

```
template<typename float_t>
struct mycomplex {
  float_t x, y;
  mycomplex<float_t>(float_t _x = 0, float_t _y =
    0) : x(_x), y(_y) {}
  float_t real() const { return x; }
  float_t imag() const { return y; }
  void real(float_t _x) { x = _x; }
  void imag(float_t _y) { y = _y; }
  mycomplex<float_t>& operator+=(const
    mycomplex<float_t> &other) { x += other.x; y
    += other.y; return *this; }
  mycomplex<float_t>& operator-=(const
    mycomplex<float_t> &other) { x -= other.x; y
    -= other.y; return *this; }
  mycomplex<float_t> operator+(const
    mycomplex<float_t> &other) const { return
    mycomplex<float_t>(*this) += other; }
  mycomplex<float_t> operator-(const
    mycomplex<float_t> &other) const { return
    mycomplex<float_t>(*this) -= other; }
  mycomplex<float_t> operator*(const
    mycomplex<float_t> &other) const {
    return {x * other.x - y * other.y, x * other.y
    + other.x * y};
  }
  mycomplex<float_t> operator*(float_t mult) const
    {
    return {x * mult, y * mult};
  }
  friend mycomplex<float_t> conj(const
    mycomplex<float_t> &c) {
    return {c.x, -c.y};
  }
  friend ostream& operator<<(ostream &stream,
    const mycomplex<float_t> &c) {
    return stream << '(' << c.x << ", " << c.y <<
    ')';
```

```
  }
};
using cd = mycomplex<double>;
void fft(vector<cd> & a, bool invert) {
  int n = a.size();
  for (int i = 1, j = 0; i < n; i++) {
    int bit = n >> 1;
    for (; j & bit; bit >>= 1)
      j ^= bit;
    j ^= bit;
    if (i < j)
      swap(a[i], a[j]);
  }
  for (int len = 2; len <= n; len <<= 1) {
    double ang = 2 * PI / len * (invert ? -1 : 1);
    cd wlen(cos(ang), sin(ang));
    for (int i = 0; i < n; i += len) {
      cd w(1);
      for (int j = 0; j < len / 2; j++) {
        cd u = a[i+j], v = a[i+j+len/2] * w;
        a[i+j] = u + v;
        a[i+j+len/2] = u - v;
        w = w*wlen;
      }
    }
  }
  if (invert) {
    for (cd & x : a){
      double z = n;
      z=1/z;
      x = x*z;
    }
    // x /= n;
  }
}
void multiply (const vector<bool> & a, const
    vector<bool> & b, vector<bool> & res)
    {//change all the bool to your type needed
  vector<cd> fa (a.begin(), a.end()),  fb
    (b.begin(), b.end());
  size_t n = 1;
  while (n < max (a.size(), b.size()))  n <<= 1;
  n <<= 1;
  fa.resize (n),  fb.resize (n);
  fft (fa, false),  fft (fb, false);
  for (size_t i=0; i<n; ++i)
    fa[i] =fa[i] * fb[i];
  fft (fa, true);
  res.resize (n);
  for (size_t i=0; i<n; ++i)
    res[i] = round(fa[i].real());
  while(res.back()==0) res.pop_back();
}
void pow(const vector<bool> &a, vector<bool>
    &res, long long int k){
  vector<bool> po=a;
  res.resize(1);
  res[0] = 1;
  while(k){
    if(k&1){
      multiply(po, res, res);
    }
    multiply(po, po, po);
    k/=2;
  }
}
```

6.2 Karatsuba Idea [5 lines]

Three subproblems:
a = xH yH
d = xL yL
e = (xH + xL)(yH + yL) - a - d
Then xy = a rn + e rn/2 + d

6.3 NTT [96 lines]

```
ll power(ll a, ll p, ll mod) {
  if (p==0)   return 1;
  ll ans = power(a, p/2, mod);
  ans = (ans * ans)%mod;
  if(p%2)    ans = (ans * a)%mod;
  return ans;
}
int primitive_root(int p) {
  vector<int> factor;
  int phi = p-1, n = phi;
  for (int i=2; i*i<=n; i++) {
    if (n%i)    continue;
    factor.push_back(i);
    while (n%i==0)  n/=i;
  }
  if (n>1)    factor.push_back(n);
  for (int res =2; res<=p; res++) {
    bool ok = true;
    for (int i=0; i<factor.size() && ok; i++)
      ok &= power(res, phi/factor[i], p) != 1;
    if (ok) return res;
  }
  return -1;
}
int nttdata(int mod, int &root, int &inv, int &pw)
    {
  int c = 0, n = mod-1;
  while (n%2==0)  c++, n/=2;
  pw = (mod-1)/n;
  int g = primitive_root(mod);
  root = power(g, n, mod);
  inv = power(root, mod-2, mod);
  return c;
}
const int M = 786433;
struct NTT {
  int N;
  vector<int> perm;
  int mod, root, inv, pw;
  NTT(){}
  NTT(int mod, int root, int inv, int pw) :
    mod(mod), root(root), inv(inv), pw(pw) {}
  void precalculate() {
    perm.resize(N);
    perm[0] = 0;
    for (int k=1; k<N; k<<=1) {
      for (int i=0; i<k; i++) {
        perm[i] <<= 1;
        perm[i+k] = 1 + perm[i];
      }
    }
  }
  void fft(vector<ll> &v, bool invert = false) {
    if (v.size() != perm.size()) {
      N = v.size();
      assert(N && (N&(N-1)) == 0);
      precalculate();
    }
    for (int i=0; i<N; i++)
      if (i < perm[i])
        swap(v[i], v[perm[i]]);
    for (int len = 2; len <= N; len <<=1) {
      ll factor = invert ? inv: root;
      for (int i=len; i<pw; i<<=1)
        factor = (factor * factor) % mod;
      for (int i=0; i<N; i+=len) {
        ll w = 1;
        for (int j=0; j<len/2; j++) {
          ll x = v[i+j], y = (w*v[i+j+len/2])%mod;
          v[i+j] = (x+y)%mod;
          v[i+j+len/2] = (x-y+mod)%mod;
          w = (w*factor)%mod;
        }
      }
    }
    if (invert) {
      ll n1 = power(N, mod-2, mod);
      for (ll &x: v)  x = (x*n1)%mod;
    }
  }
  vector<ll> multiply(vector<ll> a, vector<ll> &b)
    {
    while (a.size() && a.back() == 0)
    a.pop_back();
    while (b.size() && b.back() == 0)
    b.pop_back();
    int n = 1;
    while (n < a.size() + b.size()) n<<=1;
    a.resize(n);
    b.resize(n);
    fft(a);
    fft(b);
    for (int i=0; i<n; i++) a[i] = (a[i] *
    b[i])%M;
    fft(a, true);
    while (a.size() && a.back() == 0)
    a.pop_back();
    return a;
  }
```

```
  //      int mod=786433, root, inv, pw;
  //      nttdata(mod, root, inv, pw);
  //      NTT nn = NTT(mod, root, inv, pw);
};
```

7 Misc

7.1 template [30 lines]

```
// #pragma GCC optimize("O3,unroll-loops")
// #pragma GCC
    target("avx2,bmi,bmi2,lzcnt,popcnt")
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
template <typename A, typename B> ostream&
    operator<<(ostream& os, const pair<A, B>& p) {
    return os << '(' << p.first << ", " <<
    p.second << ')'; }
template <typename T_container, typename T =
    typename enable_if<!is_same<T_container,
    string>::value, typename
    T_container::value_type>::type> ostream&
    operator<<(ostream& os, const T_container& v)
    { os << '{'; string sep; for (const T& x : v)
    os << sep << x, sep = ", "; return os << '}';
    }
void dbg_out() { cerr << endl; }
template <typename Head, typename... Tail> void
    dbg_out(Head H, Tail... T) { cerr << " " <<
    H; dbg_out(T...); }
#ifdef SMIE
#define debug(args...) cerr << "(" << #args <<
    "):", dbg_out(args)
#else
#define debug(args...)
#endif

template <typename T> inline T gcd(T a, T b) { T
    c;while (b) { c = b;b = a % b;a = c; }return
    a; } // better than __gcd
ll powmod(ll a, ll b, ll MOD) { ll res = 1;a %=
    MOD;assert(b >= 0);for (; b; b >>= 1) { if (b
    & 1)res = res * a % MOD;a = a * a % MOD;
    }return res; }
template <typename T>using orderedSet = tree<T,
    null_type, less_equal<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
//order_of_key(k) - number of element strictly
    less than k
//find_by_order(k) - k'th element in set.(0
    indexed)(iterator)
using ll = long long;
mt19937
rng(chrono::steady_clock::now().time_since_epoch()
    .count());
//uniform_int_distribution<int>(0, i)(rng)
int main() {
  ios_base::sync_with_stdio(false);//DON'T CC++
  cin.tie(NULL);//DON'T use for interactive
}
```

8 String

8.1 Aho-Corasick [124 lines]

```
const int NODE=3000500;///Maximum Nodes
const int LGN=30;      ///Maximum Number of Tries
const int MXCHR=53;    ///Maximum Characters
const int MXP=5005;    ///
struct node {
  int val;
  int child[MXCHR];
  vector<int>graph;
  void clear(){
    CLR(child,0);
    val=0;
    graph.clear();
  }
}Trie[NODE+10];
int maxNodeId,fail[NODE+10],par[NODE+10];
int nodeSt[NODE+10],nodeEd[NODE+10];
vlong csum[NODE+10],pLoc[MXP];
void resetTrie(){
  maxNodeId=0;
}
int getNode(){
  int curNodeId=++maxNodeId;
  Trie[curNodeId].clear();
  return curNodeId;
}
inline void upd(vlong pos){
  csum[pos]++;
}
inline vlong qry(vlong pos){
  vlong res=csum[pos];
  return res;
}
struct AhoCorasick {
  int root,size,euler;
  void clear(){
    root=getNode();
    size=euler=0;
  }
  inline int getname(char ch){
    if(ch=='-')return 52;
    else if(ch>='A' && ch<='Z')return 26+(ch-'A');
    else return(ch-'a');
  }
  void addToTrie(string &s,int id){
  //Add string s to the Trie in general way
    int len=SZ(s),cur=root;
    FOR(i,0,len-1){
      int c=getname(s[i]);
      if(Trie[cur].child[c]==0){
        int curNodeId=getNode();
        Trie[curNodeId].val=c;
        Trie[cur].child[c]=curNodeId;
      }
      cur=Trie[cur].child[c];
    }
    pLoc[id]=cur;
    size++;
  }
  void calcFailFunction(){
    queue<int>Q;
    Q.push(root);
    while(!Q.empty()){
      int s=Q.front();
      Q.pop();
    //Add all the children to the queue:
      FOR(i,0,MXCHR-1){
        int t=Trie[s].child[i];
        if(t!=0){
          Q.push(t);
          par[t]=s;
        }
      }
      if(s==root){/*Handle special case when s is
    root*/
        fail[s]=par[s]=root;
        continue;
      }
  //Find fall back of s:
      int p=par[s],f=fail[p];;
      int val=Trie[s].val;
  /*Fall back till you found a node who has got val
    as a child*/
      while(f!=root && Trie[f].child[val]==0){
        f=fail[f];
      }
      fail[s]=(Trie[f].child[val]==0)? root :
    Trie[f].child[val];
  //Self fall back not allowed
      if(s==fail[s]){
        fail[s]=root;
      }
      Trie[fail[s]].graph.push_back(s);
    }
  }
  void dfs(int pos){
    ++euler;
    nodeSt[pos]=euler;
    for(auto x: Trie[pos].graph){
      dfs(x);
    }
    nodeEd[pos]=euler;
  }
  //Returns the next state
  int goTo(int state,int c){
    if(Trie[state].child[c]!=0){/*No need to fall
    back*/
      return Trie[state].child[c];
    }
  //Fall back now:
```

```
    int f=fail[state];
    while(f!=root && Trie[f].child[c]==0){
      f=fail[f];
    }
    int res=(Trie[f].child[c]==0)?
    root:Trie[f].child[c];
    return res;
  }
 /*Iterate through the whole text and find all the
    matchings*/
  void findmatching(string &s){
    int cur=root,idx=0;
    int len=SZ(s);
    while(idx<len){
      int c=getname(s[idx]);
      cur=goTo(cur,c);
      upd(nodeSt[cur]);
      idx++;
    }
  }
}acorasick;
```

8.2 Double Hasing [50 lines]

```
struct SimpleHash {
  int len;
  long long base, mod;
  vector<int> P, H, R;
  SimpleHash() {}
  SimpleHash(const char* str, long long b, long
    long m) {
    base = b, mod = m, len = strlen(str);
    P.resize(len + 4, 1), H.resize(len + 3, 0),
    R.resize(len + 3, 0);
    for (int i = 1; i <= len + 3; i++)
      P[i] = (P[i - 1] * base) % mod;
    for (int i = 1; i <= len; i++)
      H[i] = (H[i - 1] * base + str[i - 1] + 1007)
    % mod;
    for (int i = len; i >= 1; i--)
      R[i] = (R[i + 1] * base + str[i - 1] + 1007)
    % mod;
  }
  inline int range_hash(int l, int r) {
    int hashval = H[r + 1] - ((long long)P[r - l
    + 1] * H[l] % mod);
    return (hashval < 0 ? hashval + mod :
    hashval);
  }
  inline int reverse_hash(int l, int r) {
    int hashval = R[l + 1] - ((long long)P[r - l
    + 1] * R[r + 2] % mod);
    return (hashval < 0 ? hashval + mod :
    hashval);
  }
};
struct DoubleHash {
  SimpleHash sh1, sh2;
  DoubleHash() {}
  DoubleHash(const char* str) {
    sh1 = SimpleHash(str, 1949313259, 2091573227);
    sh2 = SimpleHash(str, 1997293877, 2117566807);
  }
  long long concate(DoubleHash& B , int l1 , int
    r1 , int l2 , int r2) {
    int len1 = r1 - l1+1 , len2 = r2 - l2+1;
    long long x1 = sh1.range_hash(l1, r1) ,
         x2 = B.sh1.range_hash(l2, r2);
    x1 = (x1 * B.sh1.P[len2]) % 2091573227;
    long long newx1 = (x1 + x2) % 2091573227;
    x1 = sh2.range_hash(l1, r1);
    x2 = B.sh2.range_hash(l2, r2);
    x1 = (x1 * B.sh2.P[len2]) % 2117566807;
    long long newx2 = (x1 + x2) % 2117566807;
    return (newx1 << 32) ^ newx2;
  }
  inline long long range_hash(int l, int r) {
    return ((long long)sh1.range_hash(l, r) << 32)
    ^ sh2.range_hash(l, r);
  }
  inline long long reverse_hash(int l, int r) {
    return ((long long)sh1.reverse_hash(l, r) <<
    32) ^ sh2.reverse_hash(l, r);
  }
};
```

8.3 KMP [23 lines]

```
char P[maxn],T[maxn];
int b[maxn],n,m;
void kmpPreprocess(){
  int i=0,j=-1;
  b[0]=-1;
  while(i<m){
    while(j>=0 and P[i]!=P[j])
      j=b[j];
      i++;j++;
      b[i]=j;
  }
}
void kmpSearch(){
  int i=0,j=0;
  while(i<n){
    while(j>=0 and T[i]!=P[j])
      j=b[j];
      i++;j++;
    if(j==m){
      //pattern found at index i-j
    }
  }
}
```

8.4 Palindromic Tree [30 lines]

```
struct PalindromicTree{
  int n,idx,t;
  vector<vector<int>> tree;
  vector<int> len,link;
  string s; // 1-indexed
  PalindromicTree(string str){
    s="$"+str;
    n=s.size();
    len.assign(n+5,0);
    link.assign(n+5,0);
    tree.assign(n+5,vector<int>(26,0));
  }
  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];
    }
    t=tree[t][c];
  }
  void build(){
    len[1]=-1,link[1]=1;
    len[2]=0,link[2]=1;
    idx=t=2;
    for(int i=1;i<n;i++) extend(i);
  }
};
```

8.5 Suffix Array [78 lines]

```
struct SuffixArray {
vector<int> p, c, rank, lcp;
vector<vector<int>> st;
SuffixArray(string const& s) {
  build_suffix(s + char(1));
  p.erase(p.begin());
  build_rank(p.size());
  build_lcp(s);
  build_sparse_table(lcp.size());
}
void build_suffix(string const& s) {
  int n = s.size();
  const int MX_ASCII = 256;
  vector<int> cnt(max(MX_ASCII, n), 0);
  p.resize(n); c.resize(n);
  for (int i = 0; i < n; i++) cnt[s[i]]++;
  for (int i=1; i<MX_ASCII; i++) cnt[i]+=cnt[i-1];
  for (int i = 0; i < n; i++) p[--cnt[s[i]]] = i;
  c[p[0]] = 0;
  int classes = 1;
  for (int i = 1; i < n; i++) {
    if (s[p[i]] != s[p[i-1]]) classes++;
    c[p[i]] = classes - 1;
  }
  vector<int> pn(n), cn(n);
  for (int h = 0; (1 << h) < n; ++h) {
    for (int i = 0; i < n; i++) {
      pn[i] = p[i] - (1 << h);
      if (pn[i] < 0) pn[i] += n;
    }
    fill(cnt.begin(), cnt.begin() + classes, 0);
```

```
    for (int i = 0; i < n; i++) cnt[c[pn[i]]]++;
    for (int i=1; i<classes; i++)
    cnt[i]+=cnt[i-1];
    for (int i=n-1;i>=0;i--)
    p[--cnt[c[pn[i]]]]=pn[i];
    cn[p[0]] = 0; classes = 1;
    for (int i = 1; i < n; i++) {
      pair<int, int> cur = {c[p[i]], c[(p[i] + (1
    << h)) % n]};
      pair<int, int> prev = {c[p[i-1]], c[(p[i-1]
    + (1 << h)) % n]};
      if (cur != prev) ++classes;
      cn[p[i]] = classes - 1;
    }
    c.swap(cn);
  }
}
void build_rank(int n) {
  rank.resize(n, 0);
  for (int i = 0; i < n; i++) rank[p[i]] = i;
}
void build_lcp(string const& s) {
  int n = s.size(), k = 0;
  lcp.resize(n - 1, 0);
  for (int i = 0; i < n; i++) {
    if (rank[i] == n - 1) {
      k = 0;
      continue;
    }
    int j = p[rank[i] + 1];
    while (i + k < n && j + k < n && s[i+k] ==
    s[j+k])
      k++;
    lcp[rank[i]] = k;
    if (k) k--;
  }
}
void build_sparse_table(int n) {
  int lim = __lg(n);
  st.resize(lim + 1, vector<int>(n)); st[0] = lcp;
  for (int k = 1; k <= lim; k++)
    for (int i = 0; i + (1 << k) <= n; i++)
      st[k][i] = min(st[k - 1][i], st[k - 1][i +
    (1 << (k - 1))]);
}
int get_lcp(int i) { return lcp[i]; }
int get_lcp(int i, int j) {
  if (j < i) swap(i, j);
  j--; /*for lcp from i to j we don't need last
    lcp*/
  int K = __lg(j - i + 1);
  return min(st[K][i], st[K][j - (1 << K) + 1]);
}
};
```

8.6 Trie [28 lines]

```
const int maxn=100005;
struct Trie{
  int next[27][maxn];
  int endmark[maxn],sz;
  bool created[maxn];
  void insertTrie(string& s){
    int v=0;
    for(int i=0;i<(int)s.size();i++){
      int c=s[i]-'a';
      if(!created[next[c][v]]){
        next[c][v]=++sz;
        created[sz]=true;
      }
      v=next[c][v];
    }
    endmark[v]++;
  }
  bool searchTrie(string& s){
    int v=0;
    for(int i=0;i<(int)s.size();i++){
      int c=s[i]-'a';
      if(!created[next[c][v]])
        return false;
      v=next[c][v];
    }
    return(endmark[v]>0);
  }
};
```

8.7 Z-Algorithm [19 lines]

```
void compute_z_function(const char*S,int N){
  int L=0,R=0;
  for(int i=1;i<N;++i){
    if(i>R){
      L=R=i;
      while(R<N && S[R-L]==S[R])++R;
      Z[i]=R-L,--R;
    }
    else{
      int k=i-L;
      if(Z[k]<R-i+1)Z[i]=Z[k];
      else{
        L=i;
        while(R<N && S[R-k]==S[R])++R;
        Z[i]=R-L,--R;
      }
    }
  }
}
```