Team Note of Joao

Joao Paulo Fontes Gonçalves

Compiled on May 24, 2019

Contents

1 Graph

1.1 Dinic Maxflow

Usage: Use `add_edge` to add edges
Time Complexity: $\mathcal{O}(EV^2)$

```
typedef long long int ll;

struct FlowEdge{
  int v, u;
  ll cap, flow = 0;
  FlowEdge(int v, int u, int cap): v(v), u(u), cap(cap) {}
};
```

```
struct Dinic{
  const ll flow_inf = 1e9;
  vector<FlowEdge> edges;
  vector<vector<int>> adj;
  int n, m = 0;
  int s, t;
  vector<int> level, ptr;
  queue<int> q;

  Dinic(int n, int s, int t): n(n), s(s), t(t) {
    adj.resize(n);
    level.resize(n);
    ptr.resize(n);
  }

  void add_edge(int v, int u, ll cap){
    edges.emplace_back(v, u, cap);
    edges.emplace_back(u, v, 0);
    adj[v].push_back(m);
    adj[u].push_back(m+1);
    m += 2;
  }

  bool bfs(){
    while(!q.empty()){
      int v = q.front();
      q.pop();

      for(int id: adj[v]){
        if(edges[id].cap - edges[id].flow < 1) continue;
        if(level[edges[id].u] != -1) continue;

        level[edges[id].u] = level[v] + 1;
        q.push(edges[id].u);
      }
    }

    return level[t] != -1;
  }

  ll dfs(int v, int pushed){
    if(pushed == 0) return 0;
    if(v == t) return pushed;

    for(int& cid = ptr[v]; cid < (int)adj[v].size(); cid++){
      int id = adj[v][cid];
      int u = edges[id].u;

      if(level[v] + 1 != level[u] || edges[id].cap - edges[id].flow
      < 1) continue;

      int tr = dfs(u,min(pushed,edges[id].cap - edges[id].flow));

      if(tr == 0) continue;

      edges[id].flow += tr;
      edges[id ^ 1].flow -= tr;
      return tr;
    }

    return 0;
  }

  ll flow(){
    ll f = 0;

    while(true){
      fill(level.begin(),level.end(),-1);
      level[s] = 0;
      q.push(s);

      if(!bfs()) break;

      fill(ptr.begin(), ptr.end(),0);

      while(ll pushed = dfs(s, flow_inf)) f += pushed;
    }

    return f;
  }
```

```
};
```

1.2 Hungarian algorithm

Usage: Maximum weighted bipartite matching
Time Complexity: $\mathcal{O}(V^3)$

```
#define MAXN 101

const int INF = 1000000000;
int cases, n, max_match, c1, c2;
int cost[MAXN][MAXN];
int lx[MAXN], ly[MAXN], xy[MAXN], yx[MAXN], slack[MAXN],
slackx[MAXN], ant[MAXN];
bool S[MAXN], T[MAXN];

void init_labels(){
  memset(slack,0,sizeof(slack));
  memset(slackx,0,sizeof(slackx));
  memset(lx, 0, sizeof(lx));
  memset(ly, 0, sizeof(ly));
  for (int x = 0; x < n; x++){
    for (int y = 0; y < n; y++) lx[x] = max(lx[x], cost[x][y]);
  }
}

void update_labels(){
  int x, y, delta = INF;

  for (y = 0; y < n; y++){
    if (!T[y]) delta = min(delta, slack[y]);
  }

  for (x = 0; x < n; x++){
    if (S[x]) lx[x] -= delta;
  }

  for (y = 0; y < n; y++){
    if (T[y]) ly[y] += delta;
  }
  for (y = 0; y < n; y++){
    if(!T[y]) slack[y] -= delta;
  }
}

void add_to_tree(int x, int prevx){
  S[x] = true;
  ant[x] = prevx;

  for (int y = 0; y < n; y++){
    if (lx[x] + ly[y] - cost[x][y] < slack[y]){
      slack[y] = lx[x] + ly[y] - cost[x][y];
      slackx[y] = x;
    }
  }
}

void augment(){
  if (max_match == n) return;

  int x, y, root;
  int q[MAXN], wr = 0, rd = 0;

  memset(S, false, sizeof(S));
  memset(T, false, sizeof(T));
  memset(ant, -1, sizeof(ant));

  for (x = 0; x < n; x++){
    if (xy[x] == -1){
      q[wr++] = root = x;
      ant[x] = -2;
      S[x] = true;
      break;
    }
  }

  for (y = 0; y < n; y++){
    slack[y] = lx[root] + ly[y] - cost[root][y];
    slackx[y] = root;
  }
```

```
  while (true){
    while (rd < wr){
      x = q[rd++];

      for (y = 0; y < n; y++){
        if (cost[x][y] == lx[x] + ly[y] && !T[y]){
          if (yx[y] == -1) break;

          T[y] = true;
          q[wr++] = yx[y];

          add_to_tree(yx[y], x);
        }
      }

      if (y < n) break;
    }

    if (y < n) break;

    update_labels();
    wr = rd = 0;

    for (y = 0; y < n; y++){
      if (!T[y] && slack[y] == 0){
        if (yx[y] == -1){
          x = slackx[y];
          break;
        }
        else{
          T[y] = true;
          if (!S[yx[y]]){
            q[wr++] = yx[y];
            add_to_tree(yx[y], slackx[y]);
          }
        }
      }
    }
    if (y < n) break;
  }

  if (y < n){
    max_match++;

    for (int cx = x, cy = y, ty; cx != -2; cx = ant[cx], cy = ty){
      ty = xy[cx];
      yx[cy] = cx;
      xy[cx] = cy;
    }

    augment();
  }
}

int hungarian(){
  int ret = 0;
  max_match = 0;

  memset(xy, -1, sizeof(xy));
  memset(yx, -1, sizeof(yx));
  init_labels();
  augment();

  for (int x = 0; x < n; x++){
    if(cost[x][xy[x]] > 0) ret += cost[x][xy[x]];
  }

  return ret;
}

int main(){
fio

cin >> cases;

while(cases--){
  cin >> c1 >> c2;
```

```
  n = max(c1,c2);

  rep(i,0,n-1){
    rep(j,0,n-1) cost[i][j] = -INF;
  }

  int u, v, w;

  while(true){
    cin >> u >> v >> w;

    if(u == 0 && v == 0 && w == 0) break;

    cost[u-1][v-1] = w;
  }

  int ans = hungarian();

  cout << ans << endl;
}
return 0;
}
```

1.3 SCC

Usage: All the cities which can reach all the others
Time Complexity: $\mathcal{O}(V + E)$

```
#include <bits/stdc++.h>

#define rep(i,begin,end) for(int i=begin;i<=end;i++)
#define repi(i,end,begin) for(int i=end;i>=begin;i--)
#define fio ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

using namespace std;

#define MAXN 100002

int n,m,comps;
int comp[MAXN], inDeg[MAXN];
bool visit[MAXN];
vector<int> g[MAXN], gt[MAXN];
vector<int> incTout;

void dfsG(int v){
  visit[v] = true;

  rep(i,0,(int)g[v].size() - 1){
    if(!visit[g[v][i]]) dfsG(g[v][i]);
  }

  incTout.push_back(v);
}

void dfsGt(int v){
  visit[v] = true;
  comp[v] = comps;

  rep(i,0,(int)gt[v].size() - 1){
    if(!visit[gt[v][i]]) dfsGt(gt[v][i]);
  }
}

int main(){
fio

cin >> n >> m;

int a,b;
rep(i,1,m){
  cin >> a >> b;
  g[b].push_back(a);
  gt[a].push_back(b);
}

rep(i,1,n){
  if(!visit[i]) dfsG(i);
}

memset(visit,false,sizeof(visit));
```

```
repi(i,(int)incTout.size() - 1,0){
  if(!visit[incTout[i]]){
    comps++;
    dfsGt(incTout[i]);
  }
}

rep(i,1,n){
  rep(j,0,(int)g[i].size() - 1){
    if(comp[i] != comp[g[i][j]]) inDeg[comp[g[i][j]]]++;
  }
}

int isZero = 0;
rep(i,1,comps){
  if(inDeg[i] == 0) isZero++;
}

if(isZero != 1) cout << "0" << endl;
else{
  vector<int> caps;

  rep(i,1,n){
    if(inDeg[comp[i]] == 0) caps.push_back(i);
  }

  cout << (int)caps.size() << endl;
  rep(i,0,(int)caps.size() - 1) cout << caps[i] << " ";
  cout << endl;
}
return 0;
}
```

1.4 Floyd Warshall

Usage: All the shortest paths (directed or undirected graph)
Do d[i][j] = INF and d[i][i] = 0 as preprocess
To retrieve the path store p[i][j] = k and do recursive function
Time Complexity: $\mathcal{O}(V^3)$

```
for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
          //If there is negative weight
            if (d[i][k] < INF && d[k][j] < INF)
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
            //If there is real weight
            if (d[i][k] + d[k][j] < d[i][j] - EPS)
          d[i][j] = d[i][k] + d[k][j];
        }
    }
}
```

1.5 Application Floyd Warshall: Find all pairs (i,j) which don't have a shortest path between them

Usage: Run Floyd-Warshall and check this condition
Time Complexity: $\mathcal{O}(V^3)$

```
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        for (int t = 0; t < n; ++t) {
            if (d[i][t] < INF && d[t][t] < 0 && d[t][j] < INF)
                d[i][j] = - INF;
        }
    }
}
```

1.6 Dijkstra

Usage: Shortest path (directed or undirected graph) with positive weights
Time Complexity: $\mathcal{O}(ElogV)$

```
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj;

void dijkstra(int s, vector<int> & d, vector<int> & p) {
    int n = adj.size();
    d.assign(n, INF);
    p.assign(n, -1);

    d[s] = 0;
```

```
    set<pair<int, int>> q;
    q.insert({0, s});
    while (!q.empty()) {
        int v = q.begin()->second;
        q.erase(q.begin());

        for (auto edge : adj[v]) {
            int to = edge.first;
            int len = edge.second;

            if (d[v] + len < d[to]) {
                q.erase({d[to], to});
                d[to] = d[v] + len;
                p[to] = v;
                q.insert({d[to], to});
            }
        }
    }
}
```

1.7 SPFA

Usage: Shortest path (directed or undirected graph) with possibly negative weights
Time Complexity: Worst case: $\mathcal{O}(EV)$
Average: $\mathcal{O}(E)$

```
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj;

bool spfa(int s, vector<int>& d) {
    int n = adj.size();
    d.assign(n, INF);
    vector<int> cnt(n, 0);
    vector<bool> inqueue(n, false);
    queue<int> q;

    d[s] = 0;
    q.push(s);
    inqueue[s] = true;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        inqueue[v] = false;

        for (auto edge : adj[v]) {
            int to = edge.first;
            int len = edge.second;

            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                if (!inqueue[to]) {
                    q.push(to);
                    inqueue[to] = true;
                    cnt[to]++;
                    if (cnt[to] > n)
                        return false;  // negative cycle
                }
            }
        }
    }
}
```

1.8 Bellman-Ford: Negative cycle from a vertex

Usage: Finding negative cycle from a vertex v in a directed or undirected graph with possibly negative weights
Time Complexity: $\mathcal{O}(EV)$

```
void solve()
{
    vector<int> d (n, INF);
    d[v] = 0;
    vector<int> p (n - 1);
    int x;
    for (int i=0; i<n; ++i)
    {
        x = -1;
        for (int j=0; j<m; ++j)
            if (d[e[j].a] < INF)
                if (d[e[j].b] > d[e[j].a] + e[j].cost)
                {
```

```
                    d[e[j].b] = max (-INF, d[e[j].a] + e[j].cost);
                    p[e[j].b] = e[j].a;
                    x = e[j].b;
                }
    }

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

        vector<int> path;
        for (int cur=y; ; cur=p[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] << ' ';
    }
}
```

1.9 Bellman-Ford: Find a negative cycle

Usage: Find if there is some negative cycle in a directed or undirected graph with possibly negative weights and storing one

Time Complexity: $\mathcal{O}(EV)$

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

int n, m;
vector<Edge> edges;
const int INF = 1000000000;

void solve()
{
    vector<int> d(n);
    vector<int> p(n, -1);
    int x;
    for (int i = 0; i < n; ++i) {
        x = -1;
        for (Edge e : edges) {
            if (d[e.a] + e.cost < d[e.b]) {
                d[e.b] = d[e.a] + e.cost;
                p[e.b] = e.a;
                x = e.b;
            }
        }
    }

    if (x == -1) {
        cout << "No negative cycle found.";
    } else {
        for (int i = 0; i < n; ++i)
            x = p[x];

        vector<int> cycle;
        for (int v = x;; v = p[v]) {
            cycle.push_back(v);
            if (v == x && cycle.size() > 1)
                break;
        }
        reverse(cycle.begin(), cycle.end());

        cout << "Negative cycle: ";
        for (int v : cycle)
            cout << v << ' ';
        cout << endl;
    }
}
```

1.10 Topological Sort

Usage: Directed graph without cycles
Time Complexity: $\mathcal{O}(E+V)$

```
int n; // number of vertices
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> ans;

void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
    }
    ans.push_back(v);
}

void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
    reverse(ans.begin(), ans.end());
}
```

1.11 Tarjan's off-line LCA

Usage: Tree with n nodes and m queries for LCA
Time Complexity: $\mathcal{O}(n+m)$ preprocess
$\mathcal{O}(1)$ per query

```
vector<vector<int>> adj;
vector<vector<int>> queries;
vector<int> ancestor;
vector<bool> visited;

void dfs(int v)
{
    visited[v] = true;
    ancestor[v] = v;
    for (int u : adj[v]) {
        if (!visited[u]) {
            dfs(u);
            union_sets(v, u);
            ancestor[find_set(v)] = v;
        }
    }
    for (int other_node : queries[v]) {
        if (visited[other_node])
            cout << "LCA of " << v << " and " << other_node
                 << " is " << ancestor[find_set(other_node)] <<
                 ".\n";
    }
}

void compute_LCAs() {
    // initialize n, adj and DSU
    // for (each query (u, v)) {
    //    queries[u].push_back(v);
    //    queries[v].push_back(u);
    // }

    ancestor.resize(n);
    visited.assign(n, false);
    dfs(0);
}
```

1.12 LCA by binary lifting

Usage: Tree with n nodes and m queries for LCA
Time Complexity: $\mathcal{O}(nlogn)$ preprocess
$\mathcal{O}(logn)$ per query

```
int n, l;
vector<vector<int>> adj;

int timer;
vector<int> tin, tout;
```

```cpp
vector<vector<int>> up;

void dfs(int v, int p)
{
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (int u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }

    tout[v] = ++timer;
}

bool is_ancestor(int u, int v)
{
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int u, int v)
{
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}

void preprocess(int root) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}
```

1.13 2SAT

Usage: Solves 2SAT problem
Time Complexity: $\mathcal{O}(n+m)$

```cpp
#define MAXN 100002

typedef vector<int> vi;

int n, m; //n must be two times the number of literals
int comp[MAXN];
bool visit[MAXN], val[MAXN / 2];
vi g[MAXN], gt[MAXN];
vi incTout;

void dfsG(int v)
{
    visit[v] = true;

    rep(i, 0, (int)g[v].size() - 1)
    {
        if (!visit[g[v][i]])
            dfsG(g[v][i]);
    }

    incTout.push_back(v);
}

void dfsGt(int v, int cn)
{
    comp[v] = cn;

    rep(i, 0, (int)gt[v].size() - 1)
    {
        if (comp[gt[v][i]] == -1)
            dfsGt(gt[v][i], cn);
    }
```

```
}

bool SATSolver()
{
    rep(i, 0, n)
    {
        if (!visit[i])
            dfsG(i);
    }

    memset(comp, -1, sizeof(comp));

    int compNumber = 0;
    repi(i, n - 1, 0)
    {
        if (comp[incTout[i]] == -1)
            dfsGt(incTout[i], compNumber++);
    }

    for (int i = 0; i < n - 1; i += 2) {
        if (comp[i] == comp[i + 1])
            return false;
        if (comp[i] < comp[i + 1])
            val[i / 2] = false;
        else
            val[i / 2] = true;
    }

    return true;
}
```

2 Data structures

2.1 Sparse table

Usage: Solves static RMQ query
Time Complexity: Preprocess $\mathcal{O}(nlogn)$ Query $\mathcal{O}(1)$

```
int log[MAXN+1];
log[1] = 0;
for (int i = 2; i <= MAXN; i++)
    log[i] = log[i/2] + 1;

int st[MAXN][K];

for (int i = 0; i < N; i++)
    st[i][0] = array[i];

for (int j = 1; j <= K; j++)
    for (int i = 0; i + (1 << j) <= N; i++)
        st[i][j] = min(st[i][j-1], st[i + (1 << (j - 1))][j - 1]);

int j = log[R - L + 1];
int minimum = min(st[L][j], st[R - (1 << j) + 1][j]);
```

2.2 2D Sparse table

Usage: Solves static RMQ query in 2D array
Time Complexity: Preprocess $\mathcal{O}(n^2(logn)^2)$ Query $\mathcal{O}(1)$

```
#define MAXN 502
#define LOGMAX 10

lli mat[MAXN][MAXN];
lli st[MAXN][MAXN][LOGMAX][LOGMAX];
int Log[MAXN];
int n, q, m;

int main()
{
    fio

    cin >> n >> q >> m;

    lli a;
    int i1, j1, i2, j2;
    rep(i, 1, m)
    {
        cin >> i1 >> j1 >> i2 >> j2;
        rep(j, i1, i2)
```

```
    {
        rep(z, j1, j2)
        {
            cin >> a;
            mat[j][z] += a;
        }
    }
}

Log[1] = 0;
rep(i, 2, n + 1) Log[i] = Log[i / 2] + 1;

rep(i, 1, n)
{
    rep(j, 1, n)
    {
        st[i][j][0][0] = mat[i][j];
    }
}

rep(logi, 1, Log[n + 1])
{
    rep(logj, 1, Log[n + 1])
    {
        for (int i = 1; i + (1 << logi) <= n + 1; i++) {
            for (int j = 1; j + (1 << logj) <= n + 1; j++) {
                st[i][j][logi][logj] = max(st[i][j][logi -
                1][logj - 1], st[i + (1 << (logi - 1))][j + (1
                << (logj - 1))][logi - 1][logj - 1]);
                st[i][j][logi][logj] = max(st[i][j][logi][logj],
                st[i + (1 << (logi - 1))][j][logi - 1][logj -
                1]);
                st[i][j][logi][logj] = max(st[i][j][logi][logj],
                st[i][j + (1 << (logj - 1))][logi - 1][logj -
                1]);
            }
        }
    }
}
    int s, Logi, Logj;
    lli ans;
    rep(i, 1, q)
    {
        cin >> i1 >> j1 >> s;

        i2 = i1 + s - 1;
        j2 = j1 + s - 1;
        Logi = Log[i2 - i1 + 1];
        Logj = Log[j2 - j1 + 1];
        ans = max(st[i1][j1][Logi][Logj], st[i2 - (1 << Logi) +
        1][j2 - (1 << Logj) + 1][Logi][Logj]);
        ans = max(ans, st[i2 - (1 << Logi) + 1][j1][Logi][Logj]);
        ans = max(ans, st[i1][j2 - (1 << Logj) + 1][Logi][Logj]);
        cout << ans << endl;
    }
    return 0;
}
```

2.3 Lazy segment tree

Usage: Range sum and range update
Time Complexity: Preprocess $\mathcal{O}(nlogn)$ Query $\mathcal{O}(logn)$

```
#include <bits/stdc++.h>

using namespace std;

#define MAXN 100010

typedef long long int lli;

lli input[MAXN];
lli segTree[4*MAXN];
lli lazy[4*MAXN];

void renewSegTree(int n){
for(int i=0;i<4*n;i++){
    segTree[i] = 0;
    lazy[i] = 0;
}
```

```
}

void constructSegTree(int low,int high,int pos){
if(low == high){
    segTree[pos] = input[low];
    return;
}

int mid = (low + high)/2;
constructSegTree(low,mid,2*pos+1);
constructSegTree(mid+1,high,2*pos+2);
segTree[pos] = segTree[2*pos+1] + segTree[2*pos+2];
}

void updateSegTreeRangeLazyAux(int startRange,int endRange,lli
delta,int low,int high,int pos){
if(low > high) return;

if(lazy[pos] != 0){
    segTree[pos]+=(high-low+1)*lazy[pos];
    if(low != high){
        lazy[2*pos+1] += lazy[pos];
        lazy[2*pos+2] += lazy[pos];
    }
    lazy[pos] = 0;
}

if(startRange > high || endRange < low) return;

if(startRange <= low && endRange >= high){
    segTree[pos] += (high-low+1)*delta;
    if(low != high){
        lazy[2*pos+1] += delta;
        lazy[2*pos+2] += delta;
    }
    return;
}

int mid = (low + high)/2;
updateSegTreeRangeLazyAux(startRange,endRange,delta,low,mid,2*pos+1);
updateSegTreeRangeLazyAux(startRange,endRange,delta,mid+1,high,2*pos+2);
segTree[pos] = segTree[2*pos+1] + segTree[2*pos+2];
}

lli SumQueryLazyAux(int qlow,int qhigh,int low,int high,int pos){
if(low > high) return 0;

if(lazy[pos] != 0){
    segTree[pos] += (high-low+1)*lazy[pos];
    if(low != high){
        lazy[2*pos+1] += lazy[pos];
        lazy[2*pos+2] += lazy[pos];
    }
    lazy[pos] = 0;
}

if(qlow > high || qhigh < low) return 0;

if(qlow <= low && qhigh >= high) return segTree[pos];

int mid = (low + high)/2;
return SumQueryLazyAux(qlow,qhigh,low,mid,2*pos+1) +
SumQueryLazyAux(qlow,qhigh,mid+1,high,2*pos+2);
}

void createSegmentTree(int n){
constructSegTree(0,n-1,0);
}

void updateSegTreeRangeLazy(int startRange,int endRange,lli
delta,int n){
updateSegTreeRangeLazyAux(startRange,endRange,delta,0,n-1,0);
}

lli SumQueryLazy(int qlow,int qhigh,int len){
return SumQueryLazyAux(qlow,qhigh,0,len-1,0);
}

int main(){
        int t,n,c,Q,p,q;
```

```
        lli v;

        scanf("%d",&t);

        for(int i=1;i<=t;i++){
            scanf("%d%d",&n,&c);

            if(i!=1) renewSegTree(n);

            for(int j=0;j<n;j++) input[j] = 0;

            createSegmentTree(n);

            for(int j=1;j<=c;j++){
                scanf("%d",&Q);

                if(Q == 0){
                    scanf("%d%d%lld",&p,&q,&v);
                    updateSegTreeRangeLazy(p-1, q-1, v,n);
                }
                else{
                    scanf("%d%d",&p,&q);
                    printf("%lld\n",SumQueryLazy(p-1,q-1,n));
                }
            }
        }
return 0;
}
```

2.4 Dynamic segment tree

Usage: Range sum query without preprocess
Time Complexity: Query $\mathcal{O}(logn)$

```
#include <bits/stdc++.h>

using namespace std;

struct node {
    int sum;
    node *left, *right;

    node(int low, int high)
    {
        sum = (high - low + 1);
        left = NULL;
        right = NULL;
    }

    void extend(int l, int r)
    {
        if (left == NULL) {
            int mid = (l + r) >> 1;
            left = new node(l, mid);
            right = new node(mid + 1, r);
        }
    }
};

node* Root;

void updateSegTree(int index, int delta, node* root, int l, int r)
{
    if (index < l || index > r)
        return;

    if (l == r) {
        root->sum += delta;
        return;
    }

    root->extend(l, r);
    updateSegTree(index, delta, root->left, l, (l+r) / 2);
    updateSegTree(index, delta, root->right, (l+r) / 2 + 1, r);
    root->sum = (root->left)->sum + (root->right)->sum;
}

int query(int k, node* root, int l, int r)
{
    if (l == r)
        return l;
```

```
    root->extend(l, r);
    if ((root->left)->sum >= k)
        return query(k, root->left, l, (l+r)/2);
    return query(k - ((root->left)->sum), root->right, (l+r)/2+1,
    r);
}

int main()
{
    int n, m, num;
    char op;

    scanf("%d%d", &n, &m);

    Root = new node(1, n);

    for (int i = 1; i <= m; i++) {
        scanf(" %c %d", &op, &num);
        int q = query(num, Root, 1, n);
        if (op == 'L')
            printf("%d\n", q);
        else
            updateSegTree(q, -1, Root, 1, n);
    }
    return 0;
}
```

2.5 Largest Rectangle in a histogram

Time Complexity: $\mathcal{O}(n)$

```
typedef long long int ll;

#define MAXN 100002

int n;
int h[MAXN];
stack<int> st;

int main(){
fio

while(true){
  cin >> n;

  if(n == 0) break;

  ll Max = -1;
  int Top;

  rep(i,1,n){
    cin >> h[i];

    if(st.empty() || h[i] >= h[st.top()]) st.push(i);
    else{
      while(true){
        Top = st.top();
        st.pop();

        if(st.empty()) Max = max(Max,1LL*(i - 1)*h[Top]);
        else Max = max(Max,1LL*(i - 1 - st.top())*h[Top]);

        if(st.empty() || h[i] >= h[st.top()]){
          st.push(i);
          break;
        }
      }
    }
  }

  while(!st.empty()){
      Top = st.top();
    st.pop();

    if(st.empty()) Max = max(Max,1LL*n*h[Top]);
    else Max = max(Max,1LL*(n - st.top())*h[Top]);
  }

  cout << Max << endl;
}
```

```
return 0;
}
```

3 Strings

3.1 LPS with hashing and binary search

Time Complexity: $\mathcal{O}(nlogn)$

```
typedef long long ll;

#define MAXN 100002
#define LOGMAXN 17

int n;
string s, sInv;
ll m = (ll)1000000009;
ll p1 = (ll)31;
ll p2 = (ll)37;
ll pow1[MAXN], pow2[MAXN];
ll hash1[MAXN], hash2[MAXN], hashInv1[MAXN], hashInv2[MAXN];

ll subHash(int p, int q, int st){
  if(st == 1){
    if(p == 0) return (hash1[q]*pow1[n+1])%m;
    else return (((hash1[q] - hash1[p-1] + m)%m)*pow1[n+1-p])%m;
  }
  else{
    if(p == 0) return (hash2[q]*pow2[n+1])%m;
    else return (((hash2[q] - hash2[p-1] + m)%m)*pow2[n+1-p])%m;
  }
}

ll subHashInv(int p, int q, int st){
  if(st == 1){
    if(p == 0) return (hashInv1[q]*pow1[n+1])%m;
    else return (((hashInv1[q] - hashInv1[p-1] +
    m)%m)*pow1[n+1-p])%m;
  }
  else{
    if(p == 0) return (hashInv2[q]*pow2[n+1])%m;
    else return (((hashInv2[q] - hashInv2[p-1] +
    m)%m)*pow2[n+1-p])%m;
  }
}

bool Equal(int p,int q){
  if(subHash(p,q,1) == subHashInv(n-q-1,n-p-1,1) && subHash(p,q,2)
  == subHashInv(n-q-1,n-p-1,2)) return true;
  return false;
}

bool pal(int sz){
  rep(i,0,n-sz){
    if(Equal(i,i+sz-1)) return true;
  }

  return false;
}

int main(){
fio

cin >> n;
cin >> s;

sInv = s;
rep(i,0,n-1){
  sInv[i] = s[n-1-i];
}

pow1[0] = 1LL;
rep(i,1,n+1) pow1[i] = (pow1[i-1]*p1)%m;
pow2[0] = 1LL;
rep(i,1,n+1) pow2[i] = (pow2[i-1]*p2)%m;

hash1[0] = (ll)(s[0] - 'a' + 1);
rep(i,1,n-1) hash1[i] = (hash1[i-1] + (((ll)(s[i] - 'a' +
1))*pow1[i])%m)%m;
hash2[0] = (ll)(s[0] - 'a' + 1);
```

```
rep(i,1,n-1) hash2[i] = (hash2[i-1] + (((ll)(s[i] - 'a' +
1))*pow2[i])%m)%m;

hashInv1[0] = (ll)(sInv[0] - 'a' + 1);
rep(i,1,n-1) hashInv1[i] = (hashInv1[i-1] + (((ll)(sInv[i] - 'a' +
1))*pow1[i])%m)%m;
hashInv2[0] = (ll)(sInv[0] - 'a' + 1);
rep(i,1,n-1) hashInv2[i] = (hashInv2[i-1] + (((ll)(sInv[i] - 'a' +
1))*pow2[i])%m)%m;

int ans = -1;
int begin = 1, end = (n/2) + 1, mid;
int cont = 0;

while(cont <= LOGMAXN && begin != end){
  cont++;
  mid = (begin + end)/2;
  if(pal(2*mid-1)) begin = mid;
  else end = mid;
}

if(begin == end) ans = 2*begin-1;
else{
  if(pal(2*end-1)) ans = 2*end-1;
  else ans = 2*begin-1;
}

begin = 1;
end = n/2;
cont = 0;

while(cont <= LOGMAXN && begin != end){
  cont++;
  mid = (begin + end)/2;
  if(pal(2*mid)) begin = mid;
  else end = mid;
}

if(begin == end){
  if(pal(2*begin)) ans = max(ans,2*begin);
}
else{
  if(pal(2*end)) ans = max(ans,2*end);
  else if(pal(2*begin)) ans = max(ans,2*begin);
}

cout << ans << endl;
return 0;
}
```

3.2 KMP: all occurrences of a string in another

Time Complexity: $\mathcal{O}(n+m)$

```
vector<int> KMP(string s, string t){
  int sSize = s.size();
  int tSize = t.size();

  vector<int> tPrefix(tSize);
  tPrefix[0] = 0;
  rep(i,1,tSize-1){
    int j = tPrefix[i-1];
    while(j > 0 && t[i] != t[j]) j = tPrefix[j-1];
    if(t[i] == t[j]) j++;
    tPrefix[i] = j;
  }

  vector<int> sPrefix(sSize);
  if(s[0] == t[0]) sPrefix[0] = 1;
  else sPrefix[0] = 0;
  rep(i,1,sSize-1){
    int j = sPrefix[i-1];
    if(j == tSize) j = tPrefix[j-1];
    while(j > 0 && s[i] != t[j]) j = tPrefix[j-1];
    if(s[i] == t[j]) j++;
    sPrefix[i] = j;
  }

  vector<int> occur;
  rep(i,0,sSize-1){
    if(sPrefix[i] == tSize) occur.push_back(i-tSize+1);
```

```
  }

  return occur;
}
```

3.3 Suffix Array and LCP array

Time Complexity: $\mathcal{O}(nlogn)$ for Suffix Array
$\mathcal{O}(n)$ for LCP array

```
vector<int> sort_cyclic_shifts(string (&s)){
  int n = (int)s.size();
  int alf = 256;

  vector<int> p(n), c(n), cnt(max(alf,n),0);

  rep(i,0,n-1) cnt[s[i]]++;

  rep(i,1,alf-1) cnt[i] += cnt[i-1];

  repi(i,n-1,0) p[--cnt[s[i]]] = i;


  c[p[0]] = 0;
  int classes = 1;

  rep(i,1,n-1){
    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++){
    rep(i,0,n-1){
      pn[i] = p[i] - (1<<h);

      if(pn[i] < 0) pn[i] += n;
    }
    fill(cnt.begin(),cnt.begin() + classes,0);

    rep(i,0,n-1) cnt[c[pn[i]]]++;

    rep(i,1,n-1) cnt[i] += cnt[i-1];

    repi(i,n-1,0) p[--cnt[c[pn[i]]]] = pn[i];

    cn[p[0]] = 0;
    classes = 1;

    rep(i,1,n-1){
      ii cur = {c[p[i]],c[(p[i] + (1<<h))%n]};
      ii prev = {c[p[i-1]],c[(p[i-1] + (1<<h))%n]};

      if(cur != prev) classes++;

      cn[p[i]] = classes-1;
    }

    c.swap(cn);
  }

  return p;
}

vector<int> suffix_array_construction(string s){
  s += "$";
  vector<int> sorted_shifts = sort_cyclic_shifts(s);
  sorted_shifts.erase(sorted_shifts.begin());
  return sorted_shifts;
}

vector<int> lcp_construction(string (&s), vector<int> (&p)){
  int n = (int)s.size();
  vector<int> rank(n,0);

  rep(i,0,n-1) rank[p[i]] = i;

  int k = 0;
```

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

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

    int j = p[rank[i] + 1];  ///neighbor of the i-th suffix on
    podium

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

    lcp[rank[i]] = k;

    if(k) k--;
  }

  return lcp;
}
```

4 Math

4.1 Binary exponentiaton

Time Complexity: $\mathcal{O}(logn)$

```
ll expBin(ll a, ll b){
  if(b == 0) return 1;

  a %= MOD;
  ll res = expBin(a,b/2);
  res %= MOD;

  if(b%2 == 1) return (((res*res)%MOD)*a)%MOD;
  return (res*res)%MOD;
}
```

4.2 Extended gcd

Time Complexity: $\mathcal{O}(log\ min(a, b))$

```
ll gcd(ll a, ll b, ll (&x), ll (&y)){
  if(a == 0){
    x = 0;
    y = 1;
    return b;
  }

  ll x1,y1;
  ll d = gcd(b%a,a,x1,y1);
  x = y1 - (b/a)*x1;
  y = x1;
  return d;
}
```

4.3 Modular inverse

Time Complexity: $\mathcal{O}(log\ min(a, m))$

```
ll invMod(ll a, ll m){
  ll x, y;
  ll d = gcd(a,m,x,y);
  return (x < 0) ? (x + m) : x;
}
```

4.4 Linear diophantine equation

Usage: Find one solution if it exists
Time Complexity: $\mathcal{O}(log\ min(a, b))$

```
bool sol(int a, int b, int c, int& x0, int& y0, int& g)
{
    g = gcd(abs(a), abs(b), x0, y0);

    if (c % g != 0)
        return false;

    x0 *= c / g;
    y0 *= c / g;

    if (a < 0)
        x0 = -x0;
```

```
    if (b < 0)
        y0 = -y0;

    return true;
}
```

4.5 Matrix exponentiation

Time Complexity: $\mathcal{O}(K^3 logn)$

```
ypedef long long int lli;
typedef vector<vector<lli>> matrix;

#define MAXM 102
#define MOD (lli)1000000007
#define fio ios_base::sync_with_stdio(false);cin.tie(NULL);

int m;

matrix mult(matrix A, matrix B){
matrix C(m+1,vector<lli>(m+1));
REP(i,1,m) REP(j,1,m) REP(k,1,m) C[i][j] = (C[i][j] +
(A[i][k]*B[k][j])%MOD)%MOD;
return C;
}

matrix expM(matrix A, lli n){
  if(n == 1) return A;
  else if((n%2) == 1) return mult(A,expM(A,n-1));
  else{
    matrix aux = expM(A,n/2);
        return mult(aux,aux);
  }
}
```

4.6 Sieve of Eratosthenes

Time Complexity: $\mathcal{O}(nloglogn)$ time and $\mathcal{O}(n)$ memory

```
int n;
vector<char> is_prime(n+1, true);
is_prime[0] = is_prime[1] = false;
for (int i = 2; i * i <= n; i++) {
    if (is_prime[i]) {
        for (int j = i * i; j <= n; j += i)
            is_prime[j] = false;
    }
}
```

4.7 Block Sieve

Time Complexity: $\mathcal{O}(nloglogn)$ time and $\mathcal{O}(\sqrt{n} + S)$ memory

```
int count_primes(int n) {
    const int S = 10000;

    vector<int> primes;
    int nsqrt = sqrt(n);
    vector<char> is_prime(nsqrt + 1, true);
    for (int i = 2; i <= nsqrt; i++) {
        if (is_prime[i]) {
            primes.push_back(i);
            for (int j = i * i; j <= nsqrt; j += i)
                is_prime[j] = false;
        }
    }

    int result = 0;
    vector<char> block(S);
    for (int k = 0; k * S <= n; k++) {
        fill(block.begin(), block.end(), true);
        int start = k * S;
        for (int p : primes) {
            int start_idx = (start + p - 1) / p;
            int j = max(start_idx, p) * p - start;
            for (; j < S; j += p)
                block[j] = false;
        }
        if (k == 0)
            block[0] = block[1] = false;
        for (int i = 0; i < S && start + i <= n; i++) {
            if (block[i])
```

```
                result++;
            }
        }
    return result;
}
```

4.8 Linear time sieve

Usage: Useful when finding all prime factors of numbers from 1 to n
Time Complexity: $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ memory

```
const int N = 10000000;
int lp[N+1];
vector<int> pr;

for (int i=2; i<=N; ++i) {
    if (lp[i] == 0) {
        lp[i] = i;
        pr.push_back (i);
    }
    for (int j=0; j<(int)pr.size() && pr[j]<=lp[i] && i*pr[j]<=N;
    ++j)
        lp[i * pr[j]] = pr[j];
}
```

4.9 Trial primality test

Time Complexity: $\mathcal{O}(\sqrt{n})$

```
bool isPrime(int x) {
    for (int d = 2; d * d <= x; d++) {
        if (x % d == 0)
            return false;
    }
    return true;
}
```

4.10 Fermat primality test

```
bool probablyPrimeFermat(int n, int iter=5) {
    if (n < 4)
        return n == 2 || n == 3;

    for (int i = 0; i < iter; i++) {
        int a = 2 + rand() % (n - 3);
        if (binpower(a, n - 1, n) != 1)
            return false;
    }
    return true;
}
```

4.11 Miller-Rabin primality test

```
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_composite(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 prime, else
returns false.
    if (n < 2)
        return false;

    int r = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }

    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

4.12 Trial factorization

Time Complexity: $\mathcal{O}(\sqrt{n})$

```
vector<long long> trial_division1(long long n) {
    vector<long long> factorization;
    for (long long d = 2; d * d <= n; d++) {
        while (n % d == 0) {
            factorization.push_back(d);
            n /= d;
        }
    }
    if (n > 1)
        factorization.push_back(n);
    return factorization;
}
```

4.13 Wheel factorization

Usage: Precompute primes until $\sqrt{n}$

```
vector<long long> primes;

vector<long long> trial_division4(long long n) {
    vector<long long> factorization;
    for (long long d : primes) {
        if (d * d > n)
            break;
        while (n % d == 0) {
            factorization.push_back(d);
            n /= d;
        }
    }
    if (n > 1)
        factorization.push_back(n);
    return factorization;
}
```