# Team Note of 2 3 5 8 14

Wookyung Jeong, Junghyun Lee, TBD

Compiled on October 12, 2023

## Contents

# 1 Basic Template

## 1.1 C++ Basic Template

```cpp
#include <bits/stdc++.h>
#include <cassert>
using namespace std;
#define ll long long
#define ull unsigned long long
#define ld long double
#define pii pair<int, int>
#define pll pair<ll, ll>
#define fr first
#define sc second
#define vt vector
#define all(c) (c).begin(), (c).end()
#define sz(x) (int)(x).size()

#define EXPAND( x ) x // Use this if MS Visual Studio doesn't expand __VA_ARGS__ correctly
#define F_OR(i, a, b, s) for (int i = (a); (s) > 0 ? i < (b) : i > (b); i += (s))
#define F_OR1(e) F_OR(i, 0, e, 1)
#define F_OR2(i, e) F_OR(i, 0, e, 1)
#define F_OR3(i, b, e) F_OR(i, b, e, 1)
#define F_OR4(i, b, e, s) F_OR(i, b, e, s)
#define GET5(a, b, c, d, e, ...) e
#define F_ORC(...) EXPAND( GET5(__VA_ARGS__, F_OR4, F_OR3, F_OR2, F_OR1) )
#define FOR(...) EXPAND( F_ORC(__VA_ARGS__ )(__VA_ARGS__) )
#define EACH(x, a) for (auto& x : a)

const double EPS = 1e-9;
const int INF = 1e9 + 7;
const int MOD = 1e9 + 7;
const int dy[] = { 0, 0, 1, -1, 1, 1, -1, -1 };
const int dx[] = { 1, -1, 0, 0, 1, -1, 1, -1 };

int main() {
  #ifndef ONLINE_JUDGE
  freopen("/Users/jeongwoo-kyung/Programming/CP-Codes/input.txt", "r", stdin);
  freopen("/Users/jeongwoo-kyung/Programming/CP-Codes/output.txt", "w", stdout);
  #endif

  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);

  int tc; cin >> tc;
  FOR(tt, 1, tc + 1) {

  }

  return 0;
}
```

### 1.2 #pragma GCC optimize

```cpp
#pragma GCC optimize("O3")
#pragma GCC optimize("Ofast")
#pragma GCC optimize("unroll-loops")
```

## 2 Data Structure

### 2.1 Union Find

```cpp
const int MAX = 1010101;
int n;
vector<int> uf(MAX, -1);
int find(int x) {
  if (uf[x] < 0) return x;
  return uf[x] = find(uf[x]);
}
void merge(int a, int b) {
  int A = find(a), B = find(b);
  if (A == B) return;
  uf[A] += uf[B];
  uf[B] = A;
}
int main() {
  cin >> n;
  int q; cin >> q;
  while (q--) {
    int op, u, v;
    cin >> op >> u >> v;
    if (op == 0) merge(u, v);
    if (op == 1) {
      if (find(u) == find(v)) cout << "YES\n";
      else cout << "NO\n";
    }
  }
}
```

### 2.2 Sparse Table

```cpp
const int MAX = 1e5, MAXD = 17;
int sp[MAX][MAXD];
int main() {
    // initialize
  for (int i = 0; i < MAX; i++) cin >> sp[i][0];
  for (int i = 0; i < MAX; i++) {
    for (int j = 1; j < MAXD; j++) {
      sp[i][j] = sp[sp[i][j - 1]][j - 1];
    }
  }
    // query
  int q; cin >> q;
  for (int i = 0; i < q; i++) {
    int n, x;
    cin >> n >> x;
    for (int i = MAXD - 1; i >= 0; i--) {
      if (n & sp[x][i]) x = sp[x][i];
    }
  }
}
```

### 2.3 Segment Tree

```cpp
/////////////////////////////////////////////////////
// 2.3.1 Segment Tree                              //
/////////////////////////////////////////////////////
int flag;  // array size
struct Seg {  // 1-indexed
  vector<ll> t;
  void build(int n) {
    for (flag = 1; flag < n; flag <<= 1);
    t.resize(2 * flag);
    for (int i = flag; i < flag + n; i++) cin >> t[i];
    for (int i = flag - 1; i >= 1; i--) t[i] = t[i << 1] + t[i << 1 | 1];
  }
  void modify(int p, ll value) {  // set value at position p
    for (t[p += flag - 1] = value; p > 1; p >>= 1) t[p >> 1] = t[p] + t[p ^ 1];
  }
  ll query(int l, int r, int n = 1, int nl = 1, int nr = flag) {  // sum on interval [l, r]
    if (r < nl || nr < l) return 0;
    if (l <= nl && nr <= r) return t[n];
    int mid = (nl + nr) / 2;
    return query(l, r, n << 1, nl, mid) + query(l, r, n << 1 | 1, mid + 1, nr);
  }
}seg;
int main() {
  int n, m, k;
  cin >> n >> m >> k;
  seg.build(n);
  for (int i = 0; i < m + k; i++) {
    ll op, x, y;
    cin >> op >> x >> y;
    if (op == 1) seg.modify(x, y);
    if (op == 2) cout << seg.query(x, y) << '\n';
  }
}
/////////////////////////////////////////////////////
// 2.3.2 Iterative Segment Tree                    //
/////////////////////////////////////////////////////
struct Seg {  // 0-indexed
  int n;  // array size
  ll t[2 * MAXN];
  void build(int N) {
    n = N;
    for (int i = 0; i < n; i++) cin >> t[n + i];
    for (int i = n - 1; i >= 1; i--) t[i] = t[i << 1] + t[i << 1 | 1];
  }
  void modify(int p, ll value) {  // set value at position p
    for (t[p += n] = value; p > 1; p >>= 1) t[p >> 1] = t[p] + t[p ^ 1];
  }
  ll query(int l, int r) {  // sum on interval [l, r)
    ll ret = 0;
    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if (l & 1) ret += t[l++];
      if (r & 1) ret += t[--r];
    }
    return ret;
  }
}seg;
int main() {
  int n, m, k;
  cin >> n >> m >> k;
  seg.build(n);
  for (int i = 0; i < m + k; i++) {
    ll op, x, y;
    cin >> op >> x >> y;
    if (op == 1) seg.modify(x - 1, y);
    if (op == 2) cout << seg.query(x - 1, y) << '\n';
  }
}
/////////////////////////////////////////////////////
// 2.3.3. Segment Tree with Lazy Propagation       //
/////////////////////////////////////////////////////
int flag;  // array size
struct Seg {  // 1-indexed
  vector<ll> t, lazy;
  void build(int n) {
    for (flag = 1; flag < n; flag <<= 1);
    t.resize(2 * flag);
    lazy.resize(2 * flag);
    for (int i = flag; i < flag + n; i++) cin >> t[i];
    for (int i = flag - 1; i >= 1; i--) t[i] = t[i << 1] + t[i << 1 | 1];
  }
  // add a value to all elements in interval [l, r]
  void modify(int l, int r, ll value, int n = 1, int nl = 1, int nr = flag) {
    propagate(n, nl, nr);
    if (r < nl || nr < l) return;
    if (l <= nl && nr <= r) {
      lazy[n] += value;
      propagate(n, nl, nr);
      return;
    }
    int mid = (nl + nr) >> 1;
    modify(l, r, value, n << 1, nl, mid);
    modify(l, r, value, n << 1 | 1, mid + 1, nr);
    t[n] = t[n << 1] + t[n << 1 | 1];
  }
  ll query(int l, int r, int n = 1, int nl = 1, int nr = flag) {  // sum on interval [l, r]
    propagate(n, nl, nr);
    if (r < nl || nr < l) return 0;
    if (l <= nl && nr <= r) return t[n];
    int mid = (nl + nr) / 2;
    return query(l, r, n << 1, nl, mid) + query(l, r, n << 1 | 1, mid + 1, nr);
  }
  void propagate(int n, int nl, int nr) {
    if (lazy[n] != 0) {
      if (n < flag) {
        lazy[n << 1] += lazy[n];
        lazy[n << 1 | 1] += lazy[n];
      }
      t[n] += lazy[n] * (nr - nl + 1);
      lazy[n] = 0;
    }
  }
}seg;
int main() {
  int n, m, k;
  cin >> n >> m >> k;
  seg.build(n);
  for (int i = 0; i < m + k; i++) {
    ll op, x, y, z;
    cin >> op >> x >> y;
    if (op == 1) {
      cin >> z;
      seg.modify(x, y, z);
    }
    if (op == 2) cout << seg.query(x, y) << '\n';
  }
}
/////////////////////////////////////////////////////
// 2.3.4. Dynamic Segment Tree                     //
/////////////////////////////////////////////////////
#define sz(x) (int)(x).size()
const int MAXL = 1, MAXR = 1000000;
struct Node {
    ll x; int l, r;
};
struct Dyseg {
    vector<Node> t = { { 0, -1, -1 }, { 0, -1, -1 } };
    void modify(int p, ll x, int n = 1, int nl = MAXL, int nr = MAXR) {
        if (p < nl || nr < p) return;
        t[n].x += x;
        if (nl < nr) {
            int mid = (nl + nr) >> 1;
            if (p <= mid) {
                if (t[n].l == -1) {
                    t[n].l = sz(t);
                    t.push_back({ 0, -1, -1 });
                }
                modify(p, x, t[n].l, nl, mid);
            }
            else {
                if (t[n].r == -1) {
                    t[n].r = sz(t);
                    t.push_back({ 0, -1, -1 });
                }
                modify(p, x, t[n].r, mid + 1, nr);
            }
        }
    }
    ll query(int l, int r, int n = 1, int nl = MAXL, int nr = MAXR) {
        if (r < nl || nr < l) return 0;
        if (l <= nl && nr <= r) return t[n].x;
        int mid = (nl + nr) >> 1;
        ll ret = 0;
        if (l <= mid) {
            if (t[n].l == -1) {
                t[n].l = sz(t);
                t.push_back({ 0, -1, -1 });
            }
            ret += query(l, r, t[n].l, nl, mid);
        }
        if (mid + 1 <= r) {
            if (t[n].r == -1) {
                t[n].r = sz(t);
                t.push_back({ 0, -1, -1 });
            }
            ret += query(l, r, t[n].r, mid + 1, nr);
        }
        return ret;
    }
}dyseg;
ll a[1010101];
int main() {
    int n, m, k;
    cin >> n >> m >> k;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        dyseg.modify(i, a[i]);
    }
    for (int i = 0; i < m + k; i++) {
        int op; ll x, y;
        cin >> op >> x >> y;
        if (op == 1) {
            dyseg.modify(x, y - a[x]);
            a[x] = y;
        }
        if (op == 2) cout << dyseg.query(x, y) << '\n';
    }
}
```

## 2.4 Merge Sort Tree

```
/////////////////////////////////////////////////////
// 2.4.1. Merge Sort Tree                          //
/////////////////////////////////////////////////////
#define sz(x) (int)(x).size()
#define all(c) (c).begin(), (c).end()
const int MAX = 1 << 17;
struct MergeSortTree {
    vector<int> t[MAX << 1];
    void build(const vector<int>& arr) {
        for (int i = 0; i < sz(arr); i++)
            t[i + 1 + MAX].push_back(arr[i]);
        for (int i = MAX - 1; i >= 1; i--) {
            t[i].resize(sz(t[i << 1]) + sz(t[i << 1 | 1]));
            merge(all(t[i << 1]), all(t[i << 1 | 1]), t[i].begin());
        }
    }
    int query(int l, int r, int k, int n = 1, int nl = 0, int nr = MAX - 1) { // 0-indexed,
    query on interval [l, r]
        if (nr < l || r < nl) return 0;
        if (l <= nl && nr <= r)
            return t[n].end() - upper_bound(all(t[n]), k);
        int mid = (nl + nr) >> 1;
        return query(l, r, k, n << 1, nl, mid) + query(l, r, k, n << 1 | 1, mid + 1, nr);
    }
}mstree;
int n;
vector<int> arr;
int main() {
    cin >> n;
    arr.resize(n);
    for (auto& i : arr) cin >> i;
    mstree.build(arr);
    int q; cin >> q;
    while (q--) {
        int a, b, c;
        cin >> a >> b >> c;
        int ans = mstree.query(a, b, c);
        cout << ans << '\n';
    }
}
/////////////////////////////////////////////////////
// 2.4.2. Iterative Merge Sort Tree                //
/////////////////////////////////////////////////////
#define sz(x) (int)(x).size()
#define all(c) (c).begin(), (c).end()
const int MAX = 1 << 17;
struct MergeSortTree {
    vector<int> t[MAX << 1];
    void build(const vector<int>& arr) {
        for (int i = 0; i < sz(arr); i++)
            t[i + 1 + MAX].push_back(arr[i]);
        for (int i = MAX - 1; i >= 1; i--) {
            t[i].resize(sz(t[i << 1]) + sz(t[i << 1 | 1]));
            merge(all(t[i << 1]), all(t[i << 1 | 1]), t[i].begin());
        }
    }
    int query(int l, int r, int k) { // 1-indexed, query on interval [l, r]
        l += MAX, r += MAX;
        int ret = 0;
        while (l <= r) {
            if (l & 1) ret += t[l].end() - upper_bound(all(t[l]), k), l++;
            if (~r & 1) ret += t[r].end() - upper_bound(all(t[r]), k), r--;
            l >>= 1, r >>= 1;
        }
        return ret;
    }
}mstree;
int n;
vector<int> arr;
int main() {
    cin >> n;
    arr.resize(n);
    for (auto& i : arr) cin >> i;
    mstree.build(arr);
    int q; cin >> q;
    while (q--) {
        int a, b, c;
        cin >> a >> b >> c;
        int ans = mstree.query(a, b, c);
        cout << ans << '\n';
    }
}
```

## 3 Graph

### 3.1 Dijkstra's, Bellman-Ford, Floyd-Warshall

```
// Bellman-Ford Algorithm
const int MAX = 101010;
const ll INF = 1e18;
struct wv {
  ll w; int v;
};
int n, m;
vector<wv> adj[MAX];
vector<ll> upper(MAX, INF);
void input() {
  cin >> n >> m;
  for (int i = 0; i < m; i++) {
    int u, v, w;
    cin >> u >> v >> w;
    adj[u].push_back({ w, v });
  }
}
int bellmanFord() {
  upper[1] = 0;
  int update = 1;
  for (int i = 0; i <= n; i++) {
    update = 0;
    for (int now = 1; now <= n; now++) {
      if (upper[now] == INF) continue;
      for (wv e : adj[now]) {
        int next = e.v;
        if (upper[next] > upper[now] + e.w) {
          upper[next] = upper[now] + e.w;
          update = 1;
        }
      }
    }
    if (!update) break;
  }
  return !update;
}
int main() {
  input();
  if (bellmanFord()) {
    for (int i = 2; i <= n; i++) {
      if (upper[i] == INF) cout << -1 << '\n';
      else cout << upper[i] << '\n';
    }
  }
  else cout << -1 << '\n';
}
// Floyd-Warshall Algorithm
const ll INF = 1e18;
const int MAXV = 101;
int n, m;
ll adj[MAXV][MAXV];
void init() {
  for (int i = 0; i < MAXV; i++) {
    for (int j = 0; j < MAXV; j++) {
      adj[i][j] = INF;
    }
  }
}
void input() {
  cin >> n >> m;
  for (int i = 0; i < m; i++) {
    int u, v; ll w;
    cin >> u >> v >> w;
    adj[u][v] = min(adj[u][v], w);
  }
}
void floyd() {
  for (int i = 1; i <= n; i++) adj[i][i] = 0;
  for (int k = 1; k <= n; k++) {
    for (int u = 1; u <= n; u++) {
      for (int v = 1; v <= n; v++) {
        adj[u][v] = min(adj[u][v], adj[u][k] + adj[k][v]);
      }
    }
  }
}
int main() {
  init();
  input();
  floyd();
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
      if (adj[i][j] == INF) cout << 0 << ' ';
      else cout << adj[i][j] << ' ';
    }
    cout << '\n';
  }
}
```

### 3.2 Topological Sort

```
/////////////////////////////////////////////////////
// 3.4.1. Topological Sort - DFS                   //
/////////////////////////////////////////////////////
const int MAX = 101010;
int n, m;
vector<int> adj[MAX];
stack<int> stk;
int vi[MAX], fi[MAX], isCycle;
void input() {
  cin >> n >> m;
  for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;
    adj[u].push_back(v);
  }
}
void dfs(int v) {
  vi[v] = 1;
  for (int next : adj[v]) {
    if (!vi[next]) dfs(next);
    else if (!fi[next]) isCycle = 1;
  }
  fi[v] = 1;
  stk.push(v);
}
void topologicalSort() {
  for (int i = 1; i <= n; i++) {
    if (!vi[i]) dfs(i);
  }
  if (isCycle) cout << 0;
  else {
    while (!stk.empty()) {
      cout << stk.top() << ' ';
      stk.pop();
    }
  }
}
int main() {
  input();
  topologicalSort();
}
/////////////////////////////////////////////////////
// 3.4.2. Topological Sort - Indegree              //
/////////////////////////////////////////////////////
const int MAX = 101010;
int n, m;
vector<int> adj[MAX], ts;
int ind[MAX], isCycle;
void input() {
  cin >> n >> m;
  for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;
    adj[u].push_back(v);
    ind[v]++;
  }
}
void topologicalSort() {
  queue<int> q;
  for (int i = 1; i <= n; i++)
    if (ind[i] == 0) q.push(i);
  for (int i = 0; i < n; i++) {
    if (q.empty()) {
      isCycle = 1;
      break;
    }
    int v = q.front();
    q.pop();
    ts.push_back(v);
    for (int next : adj[v]) {
      ind[next]--;
      if (ind[next] == 0) q.push(next);
    }
  }
  if (isCycle) cout << 0;
  else {
    for (int i = 0; i < ts.size(); i++)
      cout << ts[i] << ' ';
  }
}
int main() {
  input();
  topologicalSort();
}
```

### 3.3 SCC

```
/////////////////////////////////////////////////////
// 3.5.1. Kosaraju's Algorithm                     //
/////////////////////////////////////////////////////
#define sz(x) (int)(x).size()
const int MAXV = 10101;
int n, m;
vector<int> adj[MAXV], radj[MAXV];
int in[MAXV], out[MAXV], num, p[2 * MAXV];
int vi[MAXV], cnt;
vector<vector<int>> scc;
void input() {
  cin >> n >> m;
  for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;
    adj[u].push_back(v);
    radj[v].push_back(u);
  }
}
void dfs(int v) {
  in[v] = ++num;
  for (auto& i : radj[v]) {
    if (!in[i]) dfs(i);
  }
  out[v] = ++num;
  p[num] = v;
}
void flood(int v) {
  scc[cnt].push_back(v);
  vi[v] = cnt;
  for (auto& i : adj[v]) {
    if (!vi[i]) flood(i);
  }
}
void kosaraju() {
  for (int v = 1; v <= n; v++) {
    if (!in[v]) dfs(v);
  }
  for (int v = 2 * n; v >= 1; v--) {
    if (!p[v]) continue;
    if (vi[p[v]]) continue;
    cnt++;
    scc.resize(cnt + 1);
    flood(p[v]);
  }
}
void print() {
  for (auto& i : scc)
    sort(i.begin(), i.end());
  sort(scc.begin(), scc.end());
  cout << sz(scc) - 1 << '\n';
  for (int i = 1; i < sz(scc); i++) {
    auto& arr = scc[i];
    for (auto& j : arr) cout << j << ' ';
    cout << -1 << '\n';
  }
}
int main() {
  input();
  kosaraju();
  print();
}
/////////////////////////////////////////////////////
// 3.5.2. Tarjan's Algorithm                      //
/////////////////////////////////////////////////////
const int MAXV = 101010;
int n, m, label[MAXV], labelCnt;
int SCCnum[MAXV], SCCcnt, finished[MAXV];
vector<int> adj[MAXV];
stack<int> stk;
vector<vector<int>> SCC;
void input() {
  cin >> n >> m;
  for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;
    adj[u].push_back(v);
  }
}
int dfs(int v) {
  label[v] = labelCnt++;
  stk.push(v);
  int ret = label[v];
  for (int next : adj[v]) {
    // Unvisited node.
    if (label[next] == -1) ret = min(ret, dfs(next));
    // Visited but not yet classified as SCC. In other words, edge { v, next } is back edge.
    else if (!finished[next]) ret = min(ret, label[next]);
  }
  // If there is no edge to the ancestor node among itself and its descendants, find scc.
  if (ret == label[v]) {
    vector<int> vSCC;
    while (1) {
      int t = stk.top();
      stk.pop();
      vSCC.push_back(t);
      SCCnum[t] = SCCcnt;
      finished[t] = 1;
      if (t == v) break;
    }
    SCC.push_back(vSCC);
    SCCcnt++;
  }
  return ret;
}
void getSCC() {
  memset(label, -1, sizeof(label));
  for (int v = 1; v <= n; v++)
    if (label[v] == -1) dfs(v);
}
int main() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
  input();
  getSCC();
  return 0;
}
```

### 3.4 BCC

```
#define pii pair<int, int>
const int MAXV = 101010;
int n, m, dfsn[MAXV + 5], dCnt;
vector<int> adj[MAXV + 5];
stack<pii> stk;
vector<vector<pii>> bcc;
void input() {
  cin >> n >> m;
  for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
}
int dfs(int now, int prv) {
  int ret = dfsn[now] = ++dCnt;
  for (int next : adj[now]) {
    if (next == prv) continue;
    // If an edge { now, next } has not yet been visited, it puts an edge on the stack.
    if (dfsn[now] > dfsn[next]) stk.push({ now, next });
    // Back edge
    if (dfsn[next] != -1) ret = min(ret, dfsn[next]);
    // Tree edge
    else {
      int tmp = dfs(next, now);
      ret = min(ret, tmp);
      // if next cannot go to ancestor node of now, find BCC
      if (tmp >= dfsn[now]) {
        vector<pair<int, int>> nowBCC;
        while (true) {
          pair<int, int> t = stk.top();
          stk.pop();
          nowBCC.push_back(t);
          if (t == pair<int, int>(now, next)) break;
        }
        bcc.push_back(nowBCC);
      }
    }
  }
  return ret;
}
void getBCC() {
  memset(dfsn, -1, sizeof(dfsn));
  for (int v = 1; v <= n; v++)
    if (dfsn[v] == -1) dfs(v, 0);
}
int main() {
  input();
  getBCC();
}
```

## 3.5 2-SAT

```
#define pii pair<int, int>
#define fr first
#define sc second
const int MAXV = 20202;
int n, m;
int dfsn[MAXV], dCnt, sNum[MAXV], sCnt;
int finished[MAXV];
vector<int> adj[MAXV];
stack<int> stk;
pii p[MAXV];
int ans[MAXV / 2];
inline int inv(int x) {
  // negative number -a indicates ¬a.
  return (x > 0) ? 2 * (x - 1) : 2 * (-x - 1) + 1;
}
void twoCnf(int a, int b) {
  // (a ∨ b) iff (¬a → b) iff (¬b → a)
  adj[inv(-a)].push_back(inv(b));
  adj[inv(-b)].push_back(inv(a));
}
void buildGraph() {
  cin >> n >> m;
  for (int i = 0; i < m; i++) {
    int a, b;
    cin >> a >> b;
    twoCnf(a, b);
  }
}
int dfs(int now) {
  int ret = dfsn[now] = ++dCnt;
  stk.push(now);
  for (int next : adj[now]) {
    if (dfsn[next] == -1) ret = min(ret, dfs(next));
    else if (!finished[next]) ret = min(ret, dfsn[next]);
  }
  if (ret >= dfsn[now]) {
    while (1) {
      int t = stk.top();
      stk.pop();
      sNum[t] = sCnt;
      finished[t] = 1;
      if (t == now) break;
    }
    sCnt++;
  }
  return ret;
}
int isSatisfiable() {
  // determining satisfiability
  int isS = 1;
  for (int v = 0; v < 2 * n; v += 2) {
    // if x and ¬x is in same scc, then the proposition is not satisfiable
    if (sNum[v] == sNum[v + 1]) {
      isS = 0;
      break;
    }
  }
  return isS;
}
void findValueOfEachVariable() {
  // order of scc is the reverse of the topological sort
  for (int v = 0; v < 2 * n; v++) {
    p[v] = { sNum[v], v };
  }
  sort(p, p + 2 * n);
  // determining true/false of each variable
  for (int i = 2 * n - 1; i >= 0; i--) {
    int v = p[i].sc;
    if (ans[v / 2 + 1] == -1)
      ans[v / 2 + 1] = (v & 1) ? 1 : 0;
  }
  for (int v = 1; v <= n; v++)
    cout << ans[v] << ' ';
}
int main() {
  memset(dfsn, -1, sizeof(dfsn));
  memset(ans, -1, sizeof(ans));
  buildGraph();
  // finding scc
  for (int v = 0; v < 2 * n; v++)
    if (dfsn[v] == -1) dfs(v);
  if (isSatisfiable()) {
    cout << 1 << '\n';
    findValueOfEachVariable();
  }
  else cout << 0;
}
```

## 3.6 Euler Circuit

```
// Hierholzer's Algorithm
const int MAXV = 1010;
int n, adj[MAXV][MAXV], nxt[MAXV];
vector<int> eulerCircult;
void input() {
  cin >> n;
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
      cin >> adj[i][j];
    }
  }
}
int doesEulerCircuitExist() {
  // If the degree of all nodes in the graph is even, then an euler circuit exists.
  // Otherwise, the euler circuit does not exist.
  // We can do similar way to determine the existence of euler path.
  // If only two vertices have odd degree, than an eular path exists. Otherwise, the euler path
  does not exist.
  for (int i = 1; i <= n; i++) {
    int deg = 0;
    for (int j = 1; j <= n; j++) {
      deg += adj[i][j];
    }
    if (deg & 1) return 0;
  }
  return 1;
}
void dfs(int now) {
  for (int& x = nxt[now]; x <= n; x++) {
    while (x <= n && adj[now][x]) {
      adj[now][x]--;
      adj[x][now]--;
      dfs(x);
    }
  }
  eulerCircuit.push_back(now);
}
int main() {
  input();
  if (!doesEulerCircuitExist()) {
    cout << "Euler Circuit does not exist";
    return 0;
  }
  for (int i = 1; i <= n; i++) nxt[i] = 1;
  dfs(1);
  for (auto i : eulerCircuit)
    cout << i << ' ';
}
```

# 4 Tree

## 4.1 LCA in O(logN) (Sparse Table)

```
const int MAX = 101010, MAXD = 16;  // 2^MAXD = 65536
vector<int> adj[MAX];
int n, dep[MAX], par[MAX][MAXD + 1];
void input() {
  cin >> n;
  for (int i = 0; i < n - 1; i++) {
    int u, v;
    cin >> u >> v;
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
}
void dfs(int now, int prv) {
  par[now][0] = prv;
  dep[now] = dep[prv] + 1;
  for (auto i : adj[now]) {
    if (i == prv) continue;
    dfs(i, now);
  }
}
void buildSparseTable() {
  for (int i = 1; i <= MAXD; i++) {
    for (int v = 1; v <= n; v++) {
      par[v][i] = par[par[v][i - 1]][i - 1];
    }
  }
}
int lca(int u, int v) {
  if (dep[u] < dep[v]) swap(u, v);
  int diff = dep[u] - dep[v];
  for (int i = MAXD; i >= 0; i--)
    if (diff & (1 << i)) u = par[u][i];
  if (u == v) return u;
  for (int i = MAXD; i >= 0; i--) {
    if (par[u][i] ^ par[v][i]) {
      u = par[u][i];
      v = par[v][i];
    }
  }
  return par[u][0];
}
int main() {
  input();
  dfs(1, 0);
  buildSparseTable();
  int Q; cin >> Q;
  while (Q--) {
    int u, v;
    cin >> u >> v;
    cout << lca(u, v) << '\n';
  }
}
```

## 4.2 Heavy-Light Decomposition

```
const int MAXV = 202020;
int flag;  // array size
struct Seg {  // 1-indexed
  vector<ll> t;
  void build(int n) {
    for (flag = 1; flag < n; flag <<= 1);
    t.resize(2 * flag);
  }
  void modify(int p, ll value) {  // set value at position p
    for (t[p += flag - 1] = value; p > 1; p >>= 1) t[p >> 1] = t[p] + t[p ^ 1];
  }
  ll query(int l, int r, int n = 1, int nl = 1, int nr = flag) {  // sum on interval [l, r]
    if (r < nl || nr < l) return 0;
    if (l <= nl && nr <= r) return t[n];

    int mid = (nl + nr) / 2;
    return query(l, r, n << 1, nl, mid) + query(l, r, n << 1 | 1, mid + 1, nr);
  }
}seg;
vector<int> adj[MAXV], g[MAXV];
int siz[MAXV], dep[MAXV], par[MAXV];
int top[MAXV], in[MAXV], out[MAXV], pv;
void dfs(int v, int prv) {
  for (auto& i : adj[v]) {
    if (i == prv) continue;
    g[v].push_back(i);
    dfs(i, v);
  }
}
int dfs1(int v) {
  siz[v] = 1;
  for (auto& i : g[v]) {
    dep[i] = dep[v] + 1, par[i] = v;
    siz[v] += dfs1(i);
    if (siz[i] > siz[g[v][0]]) swap(i, g[v][0]);
  }
  return siz[v];
}
void dfs2(int v) {
  in[v] = ++pv;
  for (auto& i : g[v]) {
    top[i] = (i == g[v][0] ? top[v] : i);
    dfs2(i);
  }
  out[v] = pv;
}
void modify(int v, ll value) {
  seg.modify(in[v], value);
}
ll query(int u, int v) {
  ll ret = 0;
  while (top[u] ^ top[v]) {
    if (dep[top[u]] < dep[top[v]]) swap(u, v);
    int st = top[u];
    ret += seg.query(in[st], in[u]);
    u = par[st];
  }
  if (dep[u] > dep[v]) swap(u, v);
  ret += seg.query(in[u], in[v]);
  return ret;
}
int main() {
  int n, q;
  cin >> n >> q;
  for (int i = 0; i < n - 1; i++) {
    int u, v;
    cin >> u >> v;
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  dfs(1, 0);
  top[1] = 1;
  dfs1(1);
  dfs2(1);
  while (q--) {
    int op, a, b;
    cin >> op >> a >> b;
    if (op == 1) modify(a, b);
    else cout << query(a, b) << '\n';
  }
}
```

### 4.3 Centroid Decomposition

```
const int MAXV = 202020;
vector<int> adj[MAXV];
int used[MAXV], siz[MAXV], dep[MAXV], cdtree[MAXV];
int getSize(int now, int prv) {
  siz[now] = 1;
  for (auto i : adj[now]) {
    if (used[i] || prv == i) continue;
    siz[now] += getSize(i, now);
  }
  return siz[now];
}
int getCent(int now, int prv, int cnt) {
  for (auto& i : adj[now]) {
    if (used[i] || i == prv) continue;
    if (siz[i] > cnt / 2) return getCent(i, now, cnt);
  }
  return now;
}
void cd(int now, int prv) {
  int cnt = getSize(now, prv);
  int cent = getCent(now, prv, cnt);
  cdtree[now] = prv;
  used[cent] = 1;
  for (auto i : adj[cent])
    if (!used[i]) cd(i, cent);
}
int main() {
  cd(0, -1);
}
```

## 5 Network Flow

### 5.1 Maximum Flow

```
//////////////////////////////////////////////////////
// 5.1.1. Maximum Flow                              //
//////////////////////////////////////////////////////
// time complexity : O(V * E^2)
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int MAXV = 1010;
const int INF = 1e9 + 7;
int n;
ll c[MAXV][MAXV], f[MAXV][MAXV];
vector<int> adj[MAXV];
int prv[MAXV];
void input() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        int u, v, cap;
        cin >> u >> v >> cap;
        c[u][v] += cap;
        adj[u].push_back(v);
        // add reverse edge
        adj[v].push_back(u);
    }
}
void bfs(int st, int en) {
    memset(prv, -1, sizeof(prv));
    queue<int> q;
    q.push(st);
    prv[st] = 0;
    while (!q.empty() && prv[en] == -1) {
        int now = q.front();
        q.pop();
        for (int next : adj[now]) {
            if (prv[next] == -1 && c[now][next] - f[now][next] > 0) {
                q.push(next);
                prv[next] = now;
            }
        }
    }
}
ll flow(int st, int en) {
    ll block = INF;
    for (int i = en; i != st; i = prv[i]) {
        block = min(block, c[prv[i]][i] - f[prv[i]][i]);
    }
    for (int i = en; i != st; i = prv[i]) {
        f[prv[i]][i] += block;
        f[i][prv[i]] -= block;
    }
    return block;
}
ll maxFlow(int st, int en) {
    ll ret = 0;
    while (1) {
        bfs(st, en);
        if (prv[en] == -1) break;
        ret += flow(st, en);
    }
    return ret;
}
int main() {
    input();
    ll total = maxFlow(1, n);
    cout << total << '\n';
}
/////////////////////////////////////////////////////
// 5.1.2. Maximum Flow - Struct Edge               //
/////////////////////////////////////////////////////
// time complexity : O(V * E^2)
const int MAXV = 1010;
const int INF = 1e9 + 7;
struct edge {
  int v;
  ll c, f;
  edge* dual; // pointer to reverse edge
  edge() : edge(-1, 0) {}
  edge(int v1, ll c1) : v(v1), c(c1), f(0), dual(nullptr) {}
  ll residual() {
    return c - f;
  }
  void addFlow(int f1) {
    f += f1;
    dual->f -= f1;
  }
};
int n;
vector<edge*> adj[MAXV + 5];
int prv[MAXV + 5];
edge* path[MAXV + 5];
void input() {
  cin >> n;
  for (int i = 0; i < n; i++) {
    int n1, n2, cap;
    cin >> n1 >> n2 >> cap;
    edge* e1 = new edge(n2, cap), * e2 = new edge(n1, 0);
    e1->dual = e2, e2->dual = e1;
    adj[n1].push_back(e1);
    adj[n2].push_back(e2);
  }
}
void bfs(int st, int en) {
  memset(prv, -1, sizeof(prv));
  queue<int> q;
  q.push(st);
  prv[st] = 0;
  while (!q.empty() && prv[en] == -1) {
    int now = q.front();
    q.pop();
    for (auto* e : adj[now]) {
      int next = e->v;
      if (prv[next] == -1 && e->residual() > 0) {
        q.push(next);
        prv[next] = now;
        path[next] = e;
      }
    }
  }
}
ll flow(int st, int en) {
  ll block = INF;
  for (int i = en; i != st; i = prv[i]) {
    block = min(block, path[i]->residual());
  }
  for (int i = en; i != st; i = prv[i]) {
    path[i]->addFlow(block);
  }
  return block;
}
ll maxFlow(int st, int en) {
  ll ret = 0;
  while (1) {
    bfs(st, en);
    if (prv[en] == -1) break;
    ret += flow(st, en);
  }
  return ret;
}
int main() {
  input();
  ll total = maxFlow(1, n);
  cout << total << '\n';
```

```
}
```

## 5.2 Dinic's Algorithm

```
// Dinic's Algorithm
// time complexity : O(V^2 * E)
const int INF = 1e9 + 7;
const int MAXV = 505;
int N, st = 0, en = MAXV + 1;
vector<int> adj[MAXV + 5];
int c[MAXV + 5][MAXV + 5], f[MAXV + 5][MAXV + 5];
int level[MAXV + 5], work[MAXV + 5];
void input() {
  // TODO
}
void bfs() {
  memset(level, -1, sizeof(level));
  level[st] = 0;
  queue<int> q;
  q.push(st);
  while (!q.empty()) {
    int now = q.front();
    q.pop();
    for (int next : adj[now]) {
      if (level[next] == -1 && c[now][next] - f[now][next] > 0) {
        level[next] = level[now] + 1;
        q.push(next);
      }
    }
  }
}
int dfs(int now, int flow) {
  if (now == en) return flow;
  for (int& i = work[now]; i < adj[now].size(); i++) {
    int next = adj[now][i];
    if (level[next] == level[now] + 1 && c[now][next] - f[now][next] > 0) {
      int df = dfs(next, min(c[now][next] - f[now][next], flow));
      if (df > 0) {
        f[now][next] += df;
        f[next][now] -= df;
        return df;
      }
    }
  }
  return 0;
}
int dinic() {
  int ret = 0;
  while (true) {
    bfs();
    if (level[en] == -1) break;
    memset(work, 0, sizeof(work));
    while (true) {
      int flow = dfs(st, INF);
      if (flow == 0) break;
      ret += flow;
    }
  }
  return ret;
}
int main() {
  input();
  int total = dinic();
  cout << total << '\n';
}
```

## 5.3 Bipartite Matching

```
// all edges have a capacity of 1
// O(VE)
const int MAXV = 1010;
int n, m, A[MAXV], B[MAXV];
vector<int> adj[MAXV];
bool visited[MAXV];
void input() {
  cin >> n >> m;
  for (int i = 1; i <= n; i++) {
    int cnt; cin >> cnt;
    while (cnt--) {
      int x; cin >> x;
      adj[i].push_back(x);
    }
  }
}
bool dfs(int a) {
  visited[a] = 1;
  for (int b : adj[a]) {
    if (B[b] == -1 || (!visited[B[b]] && dfs(B[b]))) {
      A[a] = b;
      B[b] = a;
      return 1;
    }
  }
  return 0;
}
int bipartiteMatch() {
  memset(A, -1, sizeof(A));
  memset(B, -1, sizeof(B));
  int ret = 0;
  for (int i = 1; i <= n; i++) {
    memset(visited, 0, sizeof(visited));
    if (dfs(i)) ret++;
  }
  return ret;
}
int main() {
  input();
  int ans = bipartiteMatch();
  cout << ans << '\n';
}
```

## 5.4 Hopcroft-Karp Algorithm

```
// Bipartite Matching Algorithm
// time complexity : O(E * sqrt(V))
const int INF = 1e9 + 7;
const int MAXV = 10101;
int n, A[MAXV], B[MAXV], dist[MAXV];
bool used[MAXV];
```

```
vector<int> adj[MAXV];
void input() {
  // TODO
}
void bfs() {
  queue<int> q;
  for (int i = 0; i < n; i++) {
    if (!used[i]) {
      dist[i] = 0;
      q.push(i);
    }
    else dist[i] = INF;
  }
  while (!q.empty()) {
    int a = q.front();
    q.pop();
    for (int b : adj[a]) {
      if (B[b] != -1 && dist[B[b]] == INF) {
        dist[B[b]] = dist[a] + 1;
        q.push(B[b]);
      }
    }
  }
}
bool dfs(int a) {
  for (int b : adj[a]) {
    if (B[b] == -1 || (dist[B[b]] == dist[a] + 1 && dfs(B[b]))) {
      used[a] = true;
      A[a] = b;
      B[b] = a;
      return true;
    }
  }
  return false;
}
int hopcroft() {
  memset(A, -1, sizeof(A));
  memset(B, -1, sizeof(B));
  int ret = 0;
  while (true) {
    bfs();
    int flow = 0;
    for (int i = 0; i < n; i++)
      if (!used[i] && dfs(i)) flow++;
    if (flow == 0) break;
    ret += flow;
  }
  return ret;
}
int main() {
  input();
  int total = hopcroft();
  cout << total << '\n';
}
```

## 5.5 MCMF

```
const int INF = 1e9 + 7;
const int MAXV = 1010;
int N, M, st = 0, en = 1001;
int c[MAXV][MAXV], f[MAXV][MAXV];
int d[MAXV][MAXV], prv[MAXV];
vector<int> adj[MAXV];
int mFlow, mCost;
void input() {
  // TODO
}
void spfa() {
  memset(prv, -1, sizeof(prv));
  vector<int> dist(MAXV, INF);
  vector<bool> inQ(MAXV);
  queue<int> q;
  q.push(st);
  dist[st] = 0, inQ[st] = true;
  while (!q.empty()) {
    int now = q.front();
    q.pop();
    inQ[now] = false;
    for (int next : adj[now]) {
      if (dist[now] + d[now][next] < dist[next] && c[now][next] - f[now][next] > 0) {
        dist[next] = dist[now] + d[now][next];
        prv[next] = now;
        if (!inQ[next]) {
          inQ[next] = true;
          q.push(next);
        }
      }
    }
  }
}
void flow() {
  int block = INF;
  for (int i = en; i != st; i = prv[i]) {
    block = min(block, c[prv[i]][i] - f[prv[i]][i]);
  }
  for (int i = en; i != st; i = prv[i]) {
    mCost += d[prv[i]][i] * block;
    f[prv[i]][i] += block;
    f[i][prv[i]] -= block;
  }
  mFlow += block;
}
void mcmf() {
  while (1) {
    spfa();
    if (prv[en] == -1) break;
    flow();
  }
}
int main() {
  input();
  mcmf();
  cout << mFlow << '\n' << mCost;
}
```

## 6 String

### 6.1 Rabin-Karp Algorithm

```
const int MAX = 1010101;
const int MOD1 = 1e9 + 7, MOD2 = 1e9 + 9;
string T, P;
ll d = 128, dexp1[MAX], dexp2[MAX];
vector<int> ans;
void rabinKarp() {
  int len = sz(P);
  ll p1 = 0, p2 = 0, t1 = 0, t2 = 0;
  for (int i = 0; i < len; i++) {
    p1 = (d * p1 + P[i]) % MOD1;
    p2 = (d * p2 + P[i]) % MOD2;
    t1 = (d * t1 + T[i]) % MOD1;
    t2 = (d * t2 + T[i]) % MOD2;
  }
  if (p1 == t1 && p2 == t2) ans.push_back(0);
  for (int i = 1; i < sz(T) - len + 1; i++) {
    t1 = (d * (t1 - dexp1[len - 1] * T[i - 1]) + T[i + len - 1]) % MOD1;
    t1 = (t1 + MOD1) % MOD1;
    t2 = (d * (t2 - dexp2[len - 1] * T[i - 1]) + T[i + len - 1]) % MOD2;
    t2 = (t2 + MOD2) % MOD2;
    if (p1 == t1 && p2 == t2) ans.push_back(i);
  }
}
int main() {
  dexp1[0] = dexp2[0] = 1;
  for (int i = 1; i < MAX; i++) {
    dexp1[i] = d * dexp1[i - 1] % MOD1;
    dexp2[i] = d * dexp2[i - 1] % MOD2;
  }
  getline(cin, T);
  getline(cin, P);
  rabinKarp();
  cout << sz(ans) << '\n';
  for (int i : ans) cout << i + 1 << ' ';
}
```

### 6.2 KMP Algorithm

```
#define sz(x) (int)(x).size()
vector<int> getpi(const string& P) {
  vector<int> pi(sz(P));
  for (int i = 1, j = 0; i < sz(P); i++) {
    while (j > 0 && P[i] != P[j]) j = pi[j - 1];
    if (P[i] == P[j]) pi[i] = ++j;
  }
  return pi;
}
vector<int> kmp(const string& T, const string& P) {
  vector<int> ret;
  vector<int> pi = getpi(P);
  for (int i = 0, j = 0; i < sz(T); i++) {
    while (j > 0 && T[i] != P[j]) j = pi[j-1];
    if (T[i] == P[j]) {
      if (j == sz(P) - 1) {
        ret.push_back(i - (sz(P) - 1));
        j = pi[j];
      }
      else ++j;
    }
  }
  return ret;
}
int main() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
  string T, P;
  getline(cin, T);
  getline(cin, P);
  vector<int> ans = kmp(T, P);
  cout << sz(ans) << '\n';
  for (int i : ans)
    cout << i + 1 << '\n';
}
```

### 6.3 Trie

```
/////////////////////////////////////////////////////
// 6.3.1. Trie - Pointer                           //
/////////////////////////////////////////////////////
const char st = 'a';
const int MAXC = 'z' - 'a' + 1;
struct trie {
  trie* child[MAXC];
  bool term;
  trie() {
    fill(child, child + MAXC, nullptr);
    term = false;
  }
  ~trie() {
    for (int i = 0; i < MAXC; i++)
      if (child[i]) delete child[i];
  }
  void insert(const string& s, int key = 0) {
    if (s.size() == key) term = true;
    else {
      int next = s[key] - st;
      if (!child[next]) child[next] = new trie;
      child[next]->insert(s, key + 1);
    }
  }
  bool find(const string& s, int key = 0) {
    if (s.size() == key) return term;
    else {
      int next = s[key] - st;
      if (!child[next]) return false;
      else return child[next]->find(s, key + 1);
    }
  }
};
int main() {
  trie* root = new trie;
  int N; cin >> N;
  for (int i = 0; i < N; i++) {
    string s; cin >> s;
    root->insert(s);
  }
  int Q; cin >> Q;
  while (Q--) {
    string s; cin >> s;
    if (root->find(s)) cout << "Is exist.\n";
    else cout << "Is not exist.\n";
  }
  delete root;
}
/////////////////////////////////////////////////////
// 6.3.2. Trie - Array Index                       //
/////////////////////////////////////////////////////
const char st = '0';
const int MAXC = '9' - '0' + 1;
const int MAXN = 100 * 100 * MAXC + 1;
struct trie {
  int cnt, t[MAXN][MAXC];
  bool term[MAXN];
  void clear() {
    memset(t, 0, sizeof(t));
    memset(term, 0, sizeof(term));
    cnt = 0;
  }
  void insert(string& s) {
    int here = 0;
    for (char& i : s) {
      if (!t[here][i - st]) t[here][i - st] = ++cnt;
      here = t[here][i - st];
    }
    term[here] = true;
  }
  bool find(string& s) {
    int here = 0;
    for (int i = 0; i < s.size(); i++) {
      if (!t[here][s[i] - st]) return false;
      here = t[here][s[i] - st];
      if (i == s.size() - 1 && term[here]) return true;
    }
    return false;
  }
};
trie T;
int main() {
  int N; cin >> N;
  for (int i = 0; i < N; i++) {
    string s; cin >> s;
    T.insert(s);
  }
  int Q; cin >> Q;
  while (Q--) {
    string s; cin >> s;
    if (T.find(s)) cout << "Is exist.\n";
    else cout << "Is not exist.\n";
  }
}
```

## 6.4 Aho-Corasick

```
const char st = 'a';
const int MAXC = 'z' - 'a' + 1;
struct trie {
  trie* child[MAXC];
  trie* fail;
  bool term;
  trie() {
    fill(child, child + MAXC, nullptr);
    fail = nullptr;
    term = false;
  }
  ~trie() {
    for (int i = 0; i < MAXC; i++)
      if (child[i]) delete child[i];
  }
  void insert(const string& s, int key = 0) {
    if (s.size() == key) term = true;
    else {
      int next = s[key] - st;
      if (!child[next]) child[next] = new trie;
      child[next]->insert(s, key + 1);
    }
  }
};
trie* root = new trie;
void getFail() {
  queue<trie*> q;
  q.push(root);
  root->fail = root;
  while (!q.empty()) {
    trie* now = q.front();
    q.pop();
    for (int i = 0; i < MAXC; i++) {
      trie* next = now->child[i];
      if (!next) continue;
      if (now == root) next->fail = root;
      else {
        trie* t = now->fail;
        while (t != root && !t->child[i])
          t = t->fail;
        if (t->child[i]) t = t->child[i];
        next->fail = t;
      }
      if (next->fail->term) next->term = true;
      q.push(next);
    }
  }
}
bool isMatch(const string& s) {
  trie* now = root;
  bool ret = false;
  for (int c = 0; c < s.size(); c++) {
    int next = s[c] - st;
    while (now != root && !now->child[next])
      now = now->fail;
    if (now->child[next])
      now = now->child[next];
    if (now->term) {
      ret = true;
      break;
    }
  }
  return ret;
}
int main() {
  int N; cin >> N;
  for (int i = 0; i < N; i++) {
    string s; cin >> s;
    root->insert(s);
  }
  getFail();
  int M; cin >> M;
  for (int i = 0; i < M; i++) {
    string s; cin >> s;
    if (isMatch(s)) cout << "YES\n";
    else cout << "NO\n";
  }
  delete root;
}
```

### 6.5 Suffix Array

```
// Manber-Myers Algorithm for Suffix Array
// Time Conplexity: O(nlog^2n)
// Kasai's Algorithm for LCP(Longest Common Prefix)
// Time Complexity: O(n)
#define sz(x) (int)(x).size()
vector<int> buildsa(const string& s) {
    int n = sz(s);
    vector<int> sa(n), r(n + 1), nr(n + 1);
    for (int i = 0; i < n; i++) sa[i] = i, r[i] = s[i];
    for (int d = 1; d < n; d <<= 1) {
        auto cmp = [&](int i, int j) {
            if (r[i] ^ r[j]) return r[i] < r[j];
            return r[i + d] < r[j + d];
        };
        sort(sa.begin(), sa.end(), cmp);
        nr[sa[0]] = 1;
        for (int i = 1; i < n; i++)
            nr[sa[i]] = nr[sa[i - 1]] + cmp(sa[i - 1], sa[i]);
        r = nr;
    }
    return sa;
}
vector<int> buildlcp(const string& s, const vector<int>& sa) {
    int n = sz(s);
    vector<int> lcp(n), isa(n);
    for (int i = 0; i < n; i++) isa[sa[i]] = i;
    for (int k = 0, i = 0; i < n; i++) if (isa[i]) {
        for (int j = sa[isa[i] - 1]; s[i + k] == s[j + k]; k++);
        lcp[isa[i]] = (k ? k-- : 0);
    }
    return lcp;
}
int main() {
    string s; cin >> s;
    vector<int> sa = buildsa(s);
    vector<int> lcp = buildlcp(s, sa);
    for (auto& i : sa) cout << i + 1 << ' ';
    cout << '\n';
    cout << "x ";
    for (int i = 1; i < sz(lcp); i++) cout << lcp[i] << ' ';
}
// Manber-Myers Algorithm for Suffix Array
// Time Conplexity: O(nlogn)
// Kasai's Algorithm for LCP(Longest Common Prefix)
// Time Complexity: O(n)
#define sz(x) (int)(x).size()
vector<int> buildsa(const string& s) {
    int n = sz(s), m = max(256, n) + 1;
    vector<int> sa(n), r(2 * n), nr(2 * n), cnt(m), idx(n);
    for (int i = 0; i < n; i++) sa[i] = i, r[i] = s[i];
    for (int d = 1; d < n; d <<= 1) {
        auto cmp = [&](int i, int j) {
            if (r[i] ^ r[j]) return r[i] < r[j];
            return r[i + d] < r[j + d];
        };
        for (int i = 0; i < m; i++) cnt[i] = 0;
        for (int i = 0; i < n; i++) cnt[r[i + d]]++;
        for (int i = 1; i < m; i++) cnt[i] += cnt[i - 1];
        for (int i = n - 1; ~i; i--) idx[--cnt[r[i + d]]] = i;
        for (int i = 0; i < m; i++) cnt[i] = 0;
        for (int i = 0; i < n; i++) cnt[r[i]]++;
        for (int i = 1; i < m; i++) cnt[i] += cnt[i - 1];
        for (int i = n - 1; ~i; i--) sa[--cnt[r[idx[i]]]] = idx[i];
        nr[sa[0]] = 1;
        for (int i = 1; i < n; i++) nr[sa[i]] = nr[sa[i - 1]] + cmp(sa[i - 1], sa[i]);
        for (int i = 0; i < n; i++) r[i] = nr[i];
        if (r[sa[n - 1]] == n) break;
    }
    return sa;
}
vector<int> buildlcp(const string& s, const vector<int>& sa) {
    int n = sz(s);
    vector<int> lcp(n), isa(n);
    for (int i = 0; i < n; i++) isa[sa[i]] = i;
    for (int k = 0, i = 0; i < n; i++) if (isa[i]) {
        for (int j = sa[isa[i] - 1]; s[i + k] == s[j + k]; k++);
        lcp[isa[i]] = (k ? k-- : 0);
    }
    return lcp;
}
int main() {
    string s; cin >> s;
    vector<int> sa = buildsa(s);
    vector<int> lcp = buildlcp(s, sa);
    for (auto& i : sa) cout << i + 1 << ' ';
    cout << '\n';
    cout << "x ";
    for (int i = 1; i < sz(lcp); i++) cout << lcp[i] << ' ';
}
```

### 6.6 Manacher's Algorithm

```
// Manacher's Algorithm
// Find all palindromes in string in O(N)
// BOJ 14444 AC Code
// https://www.acmicpc.net/problem/14444
```

```
#define sz(x) (x).size()
int n; // n: length of string
string s;
vector<int> p; // p[i]: the radius of the palindrome at the current position i
void manacher() {
    // Preprocessing for determining even-length pelindromes
    n = sz(s);
    s.resize(n << 1 | 1);
    p.resize(n << 1 | 1);
    for (int i = n - 1; i >= 0; i--) {
        s[i << 1 | 1] = s[i];
        s[i << 1] = '#';
    }
    n <<= 1;
    s[n++] = '#';
    // Processing
    int r = -1, c = -1;
    // r: end of palindrome
    // c: center of palindrome
    for (int i = 0; i < n; i++) {
        if (i <= r) p[i] = min(r - i, p[c * 2 - i]);
        else p[i] = 0;
        while (1) {
            if (i - p[i] - 1 < 0 || i + p[i] + 1 >= n) break;
            if (s[i + p[i] + 1] != s[i - p[i] - 1]) break;
            p[i]++;
        }
        if (i + p[i] > r) {
            r = i + p[i], c = i;
        }
    }
}
int main() {
    cin >> s;
    manacher();
    // Get answer
    int ans = 0;
    for (int i = 0; i < n; i++) {
        ans = max(ans, p[i]);
    }
    cout << ans;
}
```

### 6.7 Z Algorithm

```
// Z Algorithm
// Given a string S of length n, the Z Algorithm produces an array Z
// where Z[i] is the length of the longest substring starting from S[i] which is also a prefix
of S
// BOJ 13713 AC Code
// https://www.acmicpc.net/problem/13713
#define sz(x) (int)(x).size()
const int MAXS = 1010101;
string s;
int z[MAXS];
void input() {
    string du; cin >> du;
    for (int i = sz(du) - 1; i >= 0; i--)
        s.push_back(du[i]);
}
void zfunction() {
    z[0] = sz(s);
    int l = 0, r = 0;
    for (int i = 1; i < sz(s); i++) {
        if (i > r) {
            l = r = i;
            while (r < sz(s) && 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 < sz(s) && s[r - l] == s[r]) r++;
                z[i] = r - l; r--;
            }
        }
    }
}
int main() {
    input();
    zfunction();
    int q; cin >> q;
    while (q--) {
        int x; cin >> x;
        cout << z[sz(s) - x] << '\n';
    }
}
```

## 7 Geometry

### 7.1 CCW Algorithm

```
#define pii pair<int, int>
struct point {
  ll x, y;
};
ll ccw(const point& a, const point& b, const point& c) {
  // res > 0 -> ccw, res < 0 -> cw, res = 0 -> colinear
  ll res = (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a.y);
  return (res > 0 ? 1 : (res < 0 ? -1 : 0));
}
// Does the line segment ab and cd intersect?
bool isCross(point a, point b, point c, point d) {
  ll ab = ccw(a, b, c) * ccw(a, b, d);
  ll cd = ccw(c, d, a) * ccw(c, d, b);
  if (ab == 0 && cd == 0) {
    pii A = { a.x, a.y }, B = { b.x, b.y }, C = { c.x, c.y }, D = { d.x, d.y };
    if (A > B) swap(A, B);
    if (C > D) swap(C, D);
    return (A <= D && C <= B);
  }
  else return (ab <= 0 && cd <= 0);
}
```

### 7.2 Convex Hull

```
////////////////////////////////////////////////////////
// 7.2.1. Graham Scam                                  //
```

```cpp
////////////////////////////////////////////////////////////////////
// BOJ 1708 AC Code
// https://www.acmicpc.net/problem/1708
#include <bits/stdc++.h>
using namespace std;
#define ll long long
struct point {
  ll x, y;
  bool operator<(const point& rhs) const {
    if (y != rhs.y) return y < rhs.y;
    return x < rhs.x;
  }
};
int N;
vector<point> p;
vector<int> st;
ll ccw(const point& a, const point& b, const point& c) {
  // res > 0 -> ccw, res < 0 -> cw, res = 0 -> colinear
  ll res = (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a.y);
  return (res > 0 ? 1 : (res < 0 ? -1 : 0));
}
void input() {
  cin >> N;
  for (int i = 0; i < N; i++) {
    int x, y;
    cin >> x >> y;
    p.push_back({ x, y });
  }
}
ll dist(const point& p1, const point& p2) {
    return (p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y);
}
bool cmp(const point& p1, const point& p2) {
    return (ccw(p[0], p1, p2) > 0 || (ccw(p[0], p1, p2) == 0 && dist(p[0], p1) < dist(p[0],
    p2)));
}
void grahamScan() {
  sort(p.begin(), p.end());
  sort(p.begin() + 1, p.end(), cmp);
  st.push_back(0);
  st.push_back(1);
  for (int next = 2; next < N; next++) {
    while (st.size() >= 2) {
      int first = st.back();
      st.pop_back();
      int second = st.back();
      if (ccw(p[second], p[first], p[next]) > 0) {
        st.push_back(first);
        break;
      }
    }
    st.push_back(next);
  }
}
int main() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
  input();
  grahamScan();
  cout << st.size();
  return 0;
}
////////////////////////////////////////////////////////////////////
// 7.2.2. Monotone Chain                                          //
////////////////////////////////////////////////////////////////////
// BOJ 1708 AC Code
// https://www.acmicpc.net/problem/1708
#include <bits/stdc++.h>
using namespace std;
#define ll long long
struct point {
  ll x, y;
  bool operator<(const point& rhs) const {
    if (x != rhs.x) return x < rhs.x;
    else return y < rhs.y;
  }
};
int N;
vector<point> p;
vector<int> dh, uh;
ll ccw(const point& a, const point& b, const point& c) {
  // res > 0 -> ccw, res < 0 -> cw, res = 0 -> colinear
  ll res = (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a.y);
  return (res > 0 ? 1 : (res < 0 ? -1 : 0));
}
void input() {
  cin >> N;
  for (int i = 0; i < N; i++) {
    int x, y;
    cin >> x >> y;
    p.push_back({ x, y });
  }
}
void monotoneChain() {
  sort(p.begin(), p.end());
  // calculate lower hull
  dh.push_back(0);
  dh.push_back(1);
  for (int next = 2; next < N; next++) {
    while (dh.size() >= 2) {
      int first = dh.back();
      dh.pop_back();
      int second = dh.back();
      if (ccw(p[second], p[first], p[next]) > 0) {
        dh.push_back(first);
        break;
      }
    }
    dh.push_back(next);
  }
  // calculate upper hull
  uh.push_back(N - 1);
  uh.push_back(N - 2);
  for (int next = N - 3; next >= 0; next--) {
    while (uh.size() >= 2) {
      int first = uh.back();
      uh.pop_back();
      int second = uh.back();
      if (ccw(p[second], p[first], p[next]) > 0) {
        uh.push_back(first);
        break;
      }
    }
    uh.push_back(next);
  }
}
int main() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
  input();
  monotoneChain();
  cout << (int)dh.size() + (int)uh.size() - 2;
  return 0;
}
```

## 7.3 Rotating Callipers

```
// BOJ 10254 AC Code
// https://www.acmicpc.net/problem/10254
struct point {
    ll x, y;
    bool operator<(const point& rhs) const {
        if (x != rhs.x) return x < rhs.x;
        return y < rhs.y;
    }
};
int n;
vector<point> p, ch;
point ans1, ans2;
void init() {
    n = 0;
    p.clear();
    ch.clear();
    ans1 = ans2 = { 0, 0 };
}
void input() {
    cin >> n;
    p.resize(n);
    ch.resize(n);
    for (auto& i : p) cin >> i.x >> i.y;
}
ll ccw(const point& a, const point& b, const point& c) {
  // res > 0 -> ccw, res < 0 -> cw, res = 0 -> colinear
  ll res = (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a.y);
  return (res > 0 ? 1 : (res < 0 ? -1 : 0));
}
ll cccw(point a, point b, point c, point d) {
    d.x -= c.x - b.x;
    d.y -= c.y - b.y;
    return ccw(a, b, d);
}
ll dist(const point& p1, const point& p2) {
    return (p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y);
}
bool cmp(const point& p1, const point& p2) {
    return (ccw(p[0], p1, p2) > 0 || (ccw(p[0], p1, p2) == 0 && dist(p[0], p1) < dist(p[0],
    p2)));
}
void rotatingCallipers() {
    sort(p.begin(), p.end());
    sort(p.begin() + 1, p.end(), cmp);
    ch[0] = p[0];
    ch[1] = p[1];
    ll fl = 2, cnt = 2;
    for (int i = 2; i < n; i++) {
        while (fl >= 2 && ccw(ch[fl - 2], ch[fl - 1], p[i]) <= 0) fl--;
        ch[fl] = p[i], fl++;
    }
    ll fl2 = 1, mx = 0;
    for (int i = 0; i < fl; i++) {
        while ((fl2 + 1) != i && cccw(ch[i], ch[i + 1], ch[fl2 % fl], ch[(fl2 + 1) % fl]) > 0) {
            if (mx < dist(ch[i], ch[fl2 % fl])) {
                ans1 = ch[i], ans2 = ch[fl2 % fl];
                mx = dist(ch[i], ch[fl2 % fl]);
            }
            fl2++;
        }
        if (mx < dist(ch[i], ch[fl2 % fl])) {
            ans1 = ch[i], ans2 = ch[fl2 % fl];
            mx = dist(ch[i], ch[fl2 % fl]);
        }
    }
}
int main() {
    int tc; cin >> tc;
    while (tc--) {
        init();
        input();
        rotatingCallipers();
        cout << ans1.x << ' ' << ans1.y << ' ';
        cout << ans2.x << ' ' << ans2.y << '\n';
    }
}
```

## 7.4 Ray Casting

```
// BOJ 1688 AC Code
// https://www.acmicpc.net/problem/1688
#define pii pair<int, int>
struct point {
    ll x, y;
    bool operator==(const point& rhs) const {
        return x == rhs.x && y == rhs.y;
    }
    bool operator<=(const point& rhs) const {
        if (x < rhs.x || (x == rhs.x && y <= rhs.y)) return 1;
        else return 0;
    }
};
int n;
vector<point> p;
point a, b, c;
void input() {
    cin >> n;
    p.resize(n);
    for (auto& i : p) {
        cin >> i.x >> i.y;
    }
    p.push_back(p[0]);
    cin >> a.x >> a.y;
    cin >> b.x >> b.y;
    cin >> c.x >> c.y;
}
ll ccw(const point& a, const point& b, const point& c) {
    // res > 0 -> ccw, res < 0 -> cw, res = 0 -> colinear
    ll res = (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a.y);
    return (res > 0 ? 1 : (res < 0 ? -1 : 0));
}
// Does the line segment ab and cd intersect?
bool isCross(point a, point b, point c, point d) {
    ll ab = ccw(a, b, c) * ccw(a, b, d);
    ll cd = ccw(c, d, a) * ccw(c, d, b);
    if (ab == 0 && cd == 0) {
        pii A = { a.x, a.y }, B = { b.x, b.y }, C = { c.x, c.y }, D = { d.x, d.y };
        if (A > B) swap(A, B);
        if (C > D) swap(C, D);
        return (A <= D && C <= B);
    }
    else return (ab <= 0 && cd <= 0);
}
bool insidePolygon(point v) {
    point u = { 1010101010ll, v.y + 1 };
    for (int i = 0; i < n; i++) {
        if (p[i] == v) return 1;
    }
    for (int i = 0; i < n; i++) {
        if (!ccw(p[i], p[i + 1], v) && (p[i] <= v ^ p[i + 1] <= v)) return 1;
    }
    int cnt = 0;
    for (int i = 0; i < n; i++) {
        cnt += isCross(p[i], p[i + 1], u, v);
    }
    return cnt & 1;
}
int main() {
    input();
    cout << (insidePolygon(a) ? 1 : 0) << '\n';
    cout << (insidePolygon(b) ? 1 : 0) << '\n';
    cout << (insidePolygon(c) ? 1 : 0) << '\n';
}
```

# 8 Math

## 8.1 Sieve of Eratosthenes

```
const int MAX = 1e6;
bool isPrime[MAX + 1];
vector<int> prime(1, 2);
void getPrime() {
  fill(isPrime + 2, isPrime + MAX + 1, 1);
  for (ll i = 4; i <= MAX; i += 2)
    isPrime[i] = 0;
  for (ll i = 3; i <= MAX; i++) {
    if (isPrime[i]) prime.push_back(i);
      for (ll j = i * i; j <= MAX; j += i * 2)
        isPrime[j] = 0;
  }
}
```

## 8.2 Linear Sieve

```
// BOJ 16563 AC Code
// https://www.acmicpc.net/problem/16563
const int MAXN = 5000000;
vector<int> sp(MAXN + 1);
vector<ll> prime;
// Determine prime numbers between 1 and MAXN in O(MAXN)
void linearSieve() {
  for (int i = 2;i <= MAXN; i++) {
    if (!sp[i]) {
      prime.push_back(i);
      sp[i] = i;
    }
    for (auto j : prime) {
      if (i * j > MAXN) break;
      sp[i * j] = j;
      if (i % j == 0) break;
    }
  }
}
// factorization in O(log x)
void factorization(int x) {
  while (x > 1) {
    cout << sp[x] << ' ';
    x /= sp[x];
  }
  cout << '\n';
}
int main() {
  linearSieve();
  int n; cin >> n;
  while (n--) {
    int x; cin >> x;
    factorization(x);
  }
}
```

## 8.3 GCD, LCM

```
ll gcd(ll a, ll b) {
  if (b == 0) return a;
  else return gcd(b, a % b);
}
ll lcm(ll a, ll b) {
  return a * b / gcd(a, b);
}
```

## 8.4 Extended GCD

```
// BOJ 14565 AC Code
// https://www.acmicpc.net/problem/14565
#define pll pair<ll, ll>
#define fr first
#define sc second
// Bézout's identity
// Let a and b be integers with greatest common divisor d.
// Then there exist integers x and y such that ax + by = d.
// Moreover, the integers of the form az + bt are exactly the multiples of d.

// If the integers x1 and y1 satisfy a * x1 + b * y1 = d,
// x2 := x1 + k * b / gcd(a, b) and
// y2 := y1 - k * a / gcd(a, b) also satisfy a * x2 + b * y2 = d for some integer k.
pair<pll, ll> egcd(ll a, ll b) { // time complexity: O(max(loga, logb))
    ll s = 0, olds = 1;
    ll t = 1, oldt = 0;
    ll r = b, oldr = a;
    while (r != 0) {
        ll q = oldr / r;
        ll tmp = oldr - q * r;
        oldr = r, r = tmp;
        tmp = olds - q * s;
        olds = s, s = tmp;
        tmp = oldt - q * t;
        oldt = t, t = tmp;
    }
    // The integers x and y are called Bézout coefficients for (a, b)
    // Bézout coefficients: (olds, oldt)
    // greatest common divisor: oldr
    // quotients by the gcd: (t, s)
    return { { olds, oldt }, oldr };
}
ll modInv(ll a, ll p) { // Find x such that ax = 1 (mod p).
    pair<pll, ll> res = egcd(a, p);
    // Modular inverse exists iff gcd(a, p) = 1.
    if (res.sc == 1) return (res.fr.fr + p) % p;
    else return -1;
}
int main() {
    ll N, A;
    cin >> N >> A;
    cout << N - A << ' ' << modInv(A, N);
}
```

## 8.5 Fermat's Little Theorem

```
// BOJ 11401 AC Code
// https://www.acmicpc.net/problem/11401
const int MOD = 1e9 + 7;
// Fermat's little theorem
// A / B = A * B^{p - 2} (mod p)
ll powxy(ll x, ll y) {
    if (y == 0) return 1;
    if (y == 1) return x;
    ll res = powxy(x, y / 2);
    return res * res % MOD * (y & 1 ? x : 1) % MOD;
}
int main() {
    ll fac[4040404] = { 1, };
    for (int i = 1; i < 4040404; i++)
        fac[i] = i * fac[i - 1] % MOD;
    int n, r;
    cin >> n >> r;
    // print nCr (mod 1e9+7)
    cout << fac[n] * powxy(fac[r], MOD - 2) % MOD * powxy(fac[n - r], MOD - 2) % MOD;
}
```

## 8.6 nCr mod p in O(1)

```
// BOJ 13977 AC Code
// https://www.acmicpc.net/problem/13977
const int MOD = 1e9 + 7;
const int MAXN = 4040404;
ll fac[MAXN], inv[MAXN], facInv[MAXN];
ll binom(int n, int r) {
  return fac[n] * facInv[r] % MOD * facInv[n - r] % MOD;
}
int main() {
  // Preprocessing in O(N)
  fac[0] = fac[1] = inv[1] = 1;
  facInv[0] = facInv[1] = 1;
  for (int i = 2; i < MAXN; i++) {
    fac[i] = i * fac[i - 1] % MOD;
    inv[i] = -(MOD / i) * inv[MOD % i] % MOD;
    if (inv[i] < 0) inv[i] += MOD;
    facInv[i] = facInv[i - 1] * inv[i] % MOD;
  }
  // Answer each query in O(1)
  int q; cin >> q;
  while (q--) {
    int n, r;
    cin >> n >> r;
    cout << binom(n, r) << '\n';
  }
}
```

## 8.7 Matrix

```
// Unverified code, many features to be added
#define sz(x) (int)(x).size()
const int MOD = 1e9 + 7;
struct Matrix {
  vector<vector<ll>> a;
  Matrix operator*(const Matrix& rhs) const {
    Matrix ret;
    ret.a.resize(sz(a), vector<ll>(sz(rhs.a[0])));
    for (int y = 0; y < sz(ret.a); y++) {
      for (int x = 0; x < sz(ret.a[y]); x++) {
        ll sum = 0;
        for (int i = 0; i < sz(a[y]); i++) {
          sum = (sum + a[y][i] * rhs.a[i][x]) % MOD;
        }
        ret.a[y][x] = sum;
      }
    }
    return ret;
  }
};
```

## 8.8 Catalan Number, Derangement Number

```
// Catalan Number
// BOJ 9343 AC Code
// https://www.acmicpc.net/problem/9343
const int MOD = 1e9 + 7;
const int MAXN = 2020202;
```

```
ll fac[MAXN], inv[MAXN], facInv[MAXN];
ll catalanNumber(int n) { // Cn = 2nCn / (n + 1) = (2n)! / (n!(n + 1)!)
    return fac[2 * n] * facInv[n] % MOD * facInv[n + 1] % MOD;
}
int main() {
    // Preprocessing in O(N)
    fac[0] = fac[1] = inv[1] = 1;
    facInv[0] = facInv[1] = 1;
    for (int i = 2; i < MAXN; i++) {
        fac[i] = i * fac[i - 1] % MOD;
        inv[i] = -(MOD / i) * inv[MOD % i] % MOD;
        if (inv[i] < 0) inv[i] += MOD;
        facInv[i] = facInv[i - 1] * inv[i] % MOD;
    }
    // Answer each query in O(1)
    int q; cin >> q;
    while (q--) {
        int n; cin >> n;
        cout << catalanNumber(n) << '\n';
    }
}
// Derangement Number
// Counting derangements of a set amounts to the hat-check problem,
// in which one considers the number of ways in which n hats
// can be returned to n people such that no hat makes it back to its owner.

// Recurrence relation:
// f1 = 0, f2 = 1.
// fi = (i - 1) * (f{i-1} + f{i-2}) (i >= 3)
const int MOD = 1e9 + 7;
const int MAX = 101010;
ll dp[MAX];
int main() {
    dp[1] = 0, dp[2] = 1;
    for (int i = 3; i < MAX; i++) {
        dp[i] = (i - 1) * (dp[i - 1] + dp[i - 2]) % MOD;
    }
    int n; cin >> n;
    cout << dp[n];
}
```

## 8.9 Euler's Phi Function

```
// BOJ 11689 AC Code
// https://www.acmicpc.net/problem/11689
#define pll pair<ll, ll>
#define fr first
#define sc second
// Find phi(x) in sqrt(x).
ll phi(ll x) {
    vector<pll> p;
    // Factorization in O(sqrt(x)).
    for (ll i = 2; i <= sqrt(x); i++) {
        ll res = 1;
        while (x % i == 0) {
            x /= i, res *= i;
        }
        if (res > 1) p.push_back({ res, i });
    }
    if (x > 1) p.push_back({ x, x });
    // Find phi(x).
    // phi(p^k) = p^{k-1} * (p - 1) for any prime number p.
    ll ret = 1;
    for (auto& i : p) {
        ret *= (i.fr / i.sc) * (i.sc - 1);
    }
    return ret;
}
int main() {
    ll n; cin >> n;
    cout << phi(n);
}
```

## 8.10 FFT

```
typedef complex<double> base;
void fft(vector<base> &a, bool inv) {
  int n = a.size(), j = 0;
  vector<base> roots(n / 2);
  for (int i = 1; i < n; i++){
    int bit = (n >> 1);
    while (j >= bit) {
      j -= bit;
      bit >>= 1;
    }
    j += bit;
    if (i < j) swap(a[i], a[j]);
  }
  double ang = 2 * acos(-1) / n * (inv ? -1 : 1);
  for (int i = 0; i < n / 2; i++){
    roots[i] = base(cos(ang * i), sin(ang * i));
  }
  for (int i = 2; i <= n; i <<= 1){
    int step = n / i;
    for (int j = 0; j < n; j += i){
      for (int k = 0; k < i / 2; k++) {
        base u = a[j + k], v = a[j + k + i / 2] * roots[step * k];
        a[j + k] = u + v;
        a[j + k + i / 2] = u - v;
      }
    }
  }
  if (inv) for (int i = 0; i < n; i++) a[i] /= n;
}
void multiply(const vector<ll> &v, const vector<ll> &w, vector<ll>& res) {
  vector<base> fv(v.begin(), v.end()), fw(w.begin(), w.end());
  int n = 2; while (n < v.size() + w.size()) n <<= 1;
  fv.resize(n); fw.resize(n);
  fft(fv, 0); fft(fw, 0);
  for (int i = 0; i < n; i++) fv[i] *= fw[i];
  fft(fv, 1);
  res.resize(n);
  for (int i = 0; i < n; i++) res[i] = (ll)round(fv[i].real());
}
```

## 8.11 Gauss-Jordan Elimination

```
// Inverse Matrix
void inverse_matrix(vector<vector<double>> &a){
    int n = a.size();
    int m = n + n;
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++j)
            a[i].push_back(i==j);
    for(int c = 0, r = 0; c < m && r < n; ++c){
        int p = r; // pivot row
        for(int i = r; i < n; ++i)
            if(a[p][c] < a[i][c])
                p = i;
        if(a[p][c] == 0){ puts("no inverse"); return; };
        for(int j = 0; j < m; ++j)
            swap(a[p][j], a[r][j]);
        double t = a[r][c];
        for(int j = 0; j < m; ++j)
            a[r][j] /= t;
        for(int i = 0; i < n; ++i) if(i != r){
            double t = a[i][c];
            for(int j = c; j < m; ++j)
                a[i][j] -= a[r][j] * t;
        }
        ++r;
    }
    for(int i=0;i<n;++i,puts(""))
        for(int j=0;j<n;++j)
            printf("%lf ",a[i][n+j]);
}
// Gauss-Jordan Elimination modulo p
vector<int> gauss_mod(vector<vector<int>> &a,int mod){
    vector<int> inv(mod); // modulo inverse 전처리
    inv[1] = 1;
    for(int i = 2; i < mod; ++i)
        inv[i] = mod - (mod/i) * inv[mod%i] % mod;
    int n = a.size();
    int m = a[0].size();
    vector<int> w(m, -1); // i번째 열에 있는 pivot이 몇 번째 행에 있는지 저장
    for(int c = 0, r = 0; c < m && r < n; ++c){
        int p = r; // pivot row
        for(int i = r; i < n; ++i)
            if(a[p][c] < a[i][c])
                p = i;
        if(a[p][c] == 0) continue; // free variable
        for(int j = 0; j < m; ++j)
            swap(a[p][j], a[r][j]);
        w[c] = r;
        int t = a[r][c];
        for(int j = 0; j < m; ++j)
            a[r][j] = a[r][j] * inv[t] % mod;
        for(int i = 0; i < n; ++i) if(i != r){
            int t = a[i][c];
            for(int j = c; j < m; ++j)
                a[i][j] = (a[i][j] - a[r][j] * t % mod + mod) % mod;
        }
        ++r;
    }
    for(int i = 0; i < n; ++i) // existence of solution
        if(count(a[i].begin(), --a[i].end(), 0) == m-1 && a[i][m-1])
            return vector<int>(); // no solution
    vector<int> ans(m);
    for(int i = 0; i < m; ++i)
        if(~w[i]) ans[i] = a[w[i]][m-1];
    return ans; // solution exist
}
// Gauss-Jordan Elimination modulo 2
const int sz = 500;
bitset<sz> gauss_bit(vector<bitset<sz>> &a){
    int n = a.size();
    int m = a[0].size();
    vector<int> w(m, -1);
    for(int c = 0, r = 0; c < m && r < n; ++c){
        for(int i = r; i < n; ++i)
            if(a[i][c]){
                swap(a[i],a[r]);
                break;
            }
        if(a[r][c] == 0) continue;
        w[c] = r;
        for(int i = 0; i < n; ++i) if(i != r)
            if(a[i][c]) a[i] ^= a[r];
        ++r;
    }
    // .. same
}
```

## 9 Misc

### 9.1 2D Prefix Sum

```
const int MAX = 1010;
ll arr[MAX][MAX], psum[MAX][MAX];
void buildPsum() {
      FOR(i, 1, MAX) {
            FOR(j, 1, MAX) {
                  psum[i][j] += arr[i][j];
                  psum[i][j] += psum[i][j - 1];
                  psum[i][j] += psum[i - 1][j];
                  psum[i][j] -= psum[i - 1][j - 1];
            }
      }
}
```

### 9.2 Convex Hull Trick

```
// BOJ 13263 AC Code
// https://www.acmicpc.net/problem/13263
struct lf { // f(x) = px + q, x >= s
    ll p, q;
    double s;
    lf(): lf(1, 0) {}
    lf(ll sp, ll sq): p(sp), q(sq), s(0) {}
};
double cross(const lf& u, const lf& v) {
    return (double)(v.q - u.q) / (u.p - v.p);
}
int n;
ll a[101010], b[101010];
ll dp[101010];
lf ch[101010];
void input() {
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    for (int i = 1; i <= n; i++) cin >> b[i];
}
void convexHullTrick() {
    int top = 1;
    for (int i = 2; i <= n; i++) {
        lf g(b[i - 1], dp[i - 1]);
        while (top > 1) {
            g.s = cross(ch[top - 1], g);
            if (ch[top - 1].s < g.s) break;
            --top;
        }
        ch[top++] = g;
        int l = 1, r = top - 1;
        while (l < r) {
            int mid = (l + r + 1) >> 1;
            if (a[i] < ch[mid].s) r = mid - 1;
            else l = mid;
        }
        int fpos = l;
        dp[i] = ch[fpos].p * a[i] + ch[fpos].q;
    }
}
int main() {
    input();
    convexHullTrick();
    cout << dp[n];
}
```

## 9.3 DP Opt

```
// Knuth Optimization
// Recurrence: DP[i][j] = min(DP[i][k] + DP[k + 1][j]) + C[i][j] (i <= k < j)
// Condition: C[i][j] is a monge array (satisfies C[a][c] + C[b][d] <= C[a][d] + C[b][c]),
//            and satisfies C[a][d] >= C[b][c] for a <= b <= c <= d
// Naive Complexity: O(n^3)
// Optimized Complexity: O(n^2)

// Letopt[i][j] be the value of k that minimizes DP[i][j]
// The following holds: opt[i][j - 1] <= opt[i][j] <= opt[i + 1][j]

// BOJ 13974 AC Code
// https://www.acmicpc.net/problem/13974
const ll INF = 1e18;
int n, opt[5050][5050];
ll a[5050], DP[5050][5050], psum[5050];
int main() {
    int tc; cin >> tc;
    while (tc--) {
        cin >> n;
        for (int i = 1; i <= n; i++) {
            cin >> a[i];
            psum[i] = a[i] + psum[i - 1];
        }
        for (int i = 1; i <= n; i++) {
            DP[i][i] = 0;
            opt[i][i] = i;
        }
        for (int i = n - 1; i >= 1; i--) {
            for (int j = i + 1; j <= n; j++) {
                ll mn = INF, mnk = -1;
                for (int k = opt[i][j - 1]; k <= opt[i + 1][j]; k++) {
                    ll res = DP[i][k] + DP[k + 1][j] + (psum[j] - psum[i - 1]);
                    if (res < mn) {
                        mn = res;
                        mnk = k;
                    }
                }
                DP[i][j] = mn;
                opt[i][j] = mnk;
            }
        }
        cout << DP[1][n] << '\n';
    }
}

// Divide and Conquer Optimization
// Recurrence: DP[t][i] = min(DP[t - 1][j] + C[j][i]) (1 <= j < n)
// Condition: Let opt[t][i] be j with the smallest value of DP[t - 1][j] + C[j][i]. It must
satisfy opt[t][i] <= opt[t][i + 1].
// BOJ 12766 AC Code
// https://www.acmicpc.net/problem/12766
const ll INF = 1e18;
const int MAX = 5050;
int n, b, s, r;
ll w[MAX], dp[MAX][MAX], psum[MAX];
struct wv {
    ll w; int v;
    bool operator<(const wv& rhs) const {
        return w > rhs.w;
    }
};
vector<wv> adj[MAX], radj[MAX];
void input() {
    cin >> n >> b >> s >> r;
    for (int i = 0; i < r; i++) {
        int u, v, d;
        cin >> u >> v >> d;
        adj[u].push_back({ d, v });
        radj[v].push_back({ d, u });
    }
}
vector<ll> dist(MAX, INF), rdist(MAX, INF);
void dijkstra() {
    priority_queue<wv> pq;
    pq.push({ 0, b + 1 });
    dist[b + 1] = 0;
    while (!pq.empty()) {
        int v = pq.top().v;
        ll w = pq.top().w;
        pq.pop();
        if (w > dist[v]) continue;
        for (auto& i : adj[v]) {
            if (dist[i.v] > w + i.w) {
                dist[i.v] = w + i.w;
                pq.push({ w + i.w, i.v });
            }
        }
    }
    pq.push({ 0, b + 1 });
    rdist[b + 1] = 0;
    while (!pq.empty()) {
        int v = pq.top().v;
        ll w = pq.top().w;
        pq.pop();
        if (w > rdist[v]) continue;
        for (auto& i : radj[v]) {
            if (rdist[i.v] > w + i.w) {
                rdist[i.v] = w + i.w;
                pq.push({ w + i.w, i.v });
            }
        }
    }
}
void f(int gr, int l, int r, int nl, int nr) {
    int mid = (l + r) >> 1, idx = -1;
    ll& res = dp[gr][mid];
    res = INF;
    for (int i = nl; i <= min(mid, nr); i++) {
        ll val = dp[gr - 1][i] + (mid - i - 1) * (psum[mid] - psum[i]);
        if (res > val) {
            res = val, idx = i;
        }
    }
    if (l < r) {
        f(gr, l, mid, nl, idx);
        f(gr, mid + 1, r, idx, nr);
    }
}
int main() {
    input();
    dijkstra();
    for (int i = 1; i <= b; i++) {
        w[i] = dist[i] + rdist[i];
    }
    sort(w + 1, w + 1 + b);
    for (int i = 1; i <= b; i++) {
        psum[i] = w[i] + psum[i - 1];
    }
    for (int i = 1; i <= b; i++) {
        dp[1][i] = (i - 1) * psum[i];
    }
    for (int i = 2; i <= s; i++) {
        f(i, i, b, 0, b);
    }
    cout << dp[s][b];
}
```

## 9.4 Sqrt Decomposition, Mo's Algorithm

```
int sq;
struct se {
  int s, e, idx;
  bool operator<(const se& rhs) const {
    if (s / sq != rhs.s / sq) return s / sq < rhs.s / sq;
    return e < rhs.e;
  }
};
vector<se> q;
vector<int> ans;
void input() {
  // TODO: 1. receive input 2. resize q, ans 3. calculate sq
}
void add(int idx) {
  // TODO: add value at idx from data structure
}
void del(int idx) {
  // TODO: remove value at idx from data structure
}
int query() {
  // TODO: extract the current answer of the data structure
}
void f() {
  int s = q[0].s, e = q[0].e;
  // TODO: initialize data structure
  ans[q[0].idx] = query();
  for (int i = 1; i < q.size(); i++) {
    while (q[i].s < s) add(--s);
    while (e < q[i].e) add(++e);
    while (s < q[i].s) del(s++);
    while (q[i].e < e) del(e--);
    ans[q[i].idx] = query();
  }
}
int main() {
  input();
  sort(q.begin(), q.end());
  f();
  for (auto& i : ans)
    cout << i << '\n';
}
```

## 9.5 Checklist

```
- 비슷한 문제를 풀어본 적이 있던가?
- 단순한 방법에서 시작할 수 있을까? (Brute Force)
- 내가 문제를 푸는 과정을 수식화할 수 있을까? (예제를 직접 해결해보면서)
- 문제를 단순화할 수 없을까?
- 그림으로 그려볼 수 있을까?
- 수식으로 표현할 수 있을까?
- 문제를 분해할 수 있을까?
- 뒤에서부터 생각해서 풀 수 있을까?
- 순서를 강제할 수 있을까?
- 특정 형태의 답만을 고려할 수 있을까? (정규화)
- a = b: a만 움직이기, b만 움직이기, 두 개 동시에 움직이기, 반대로 움직이기
- 말도 안 되는 것들을 한 번은 생각해보기 / 당연하다고 생각한 것 다시 생각해보기
- 확률: DP, 이분 탐색
- 최대/최소: 이분 탐색, 그리디(Prefix 고정, Exchange Argument), DP(순서 고정)
```