Team Note of 2 3 5 8 14

Wookyung Jeong, Junghyun Lee, Sungmoon Jung

Compiled on October 18, 2023

Contents

1 Basic Template

1.1 C++ Basic Template

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

#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 all(c) (c).begin(), (c).end()
#define sz(x) (int)(x).size()

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("Write your absolute/relative path of input.txt", "r", stdin);
  // freopen("Write your absolute/relative path of output.txt", "w", stdout);
  // #endif

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

2 Data Structure

2.1 Segment Tree

```
// 2. Iterative Segment Tree
const int MAXN = 1010101;  // limit for array size
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;

// 5. Persistent Segment Tree
// TIME COMPLEXITY: O(n) for initialize PST, O(logn) for each query.
// SPACE COMPLEXITY: O(nlogm).
struct PST { // 1-indexed
    int flag; // array size
    struct Node { int l, r; ll val; };
    vector<Node> t;
    vector<int> root;

    void addNode() { t.push_back({ -1, -1, 0 }); }
    void build(int l, int r, int n)  {
        assert(0 <= n && n < sz(t));
        if (l == r) { t[n].val = a[l]; return; }
        addNode();
        t[n].l = sz(t) - 1;
        addNode();
        t[n].r = sz(t) - 1;

        int mid = (l + r) >> 1;
        build(l, mid, t[n].l);
        build(mid + 1, r, t[n].r);
        t[n].val = t[t[n].l].val + t[t[n].r].val;
    }
    void build(int Flag) {
        addNode();
        root.push_back(sz(t) - 1);
        flag = Flag;
        build(1, flag, root[0]);
    }
    void modify(int p, ll val, int l, int r, int n1, int n2) {
        assert(0 <= n1 && n1 < sz(t));
        assert(0 <= n2 && n2 < sz(t));
        if (p < l || r < p) { t[n2] = t[n1]; return; }
        if (l == r) { t[n2].val = val; return; }

        int mid = (l + r) >> 1;
        if (p <= mid) {
            t[n2].r = t[n1].r;
            addNode();
            t[n2].l = sz(t) - 1;
            modify(p, val, l, mid, t[n1].l, t[n2].l);
        }
        else {
            t[n2].l = t[n1].l;
            addNode();
            t[n2].r = sz(t) - 1;
            modify(p, val, mid + 1, r, t[n1].r, t[n2].r);
        }
        t[n2].val = t[t[n2].l].val + t[t[n2].r].val;
    }
    void modify(int p, ll val) {
        addNode();
        root.push_back(sz(t) - 1);
        modify(p, val, 1, flag, root[sz(root) - 2], root[sz(root) - 1]);
    }
    ll query(int l, int r, int n, int nl, int nr) {
        assert(0 <= n && n < sz(t));
        if (r < nl || nr < l) return 0;
        if (l <= nl && nr <= r) return t[n].val;
        int mid = (nl + nr) >> 1;
        return query(l, r, t[n].l, nl, mid) + query(l, r, t[n].r, mid + 1, nr);
    }
    ll query(int l, int r, int n) {
        assert(n < sz(root));
        return query(l, r, root[n], 1, flag);
    }
}pst;
```

2.2 Fenwick Tree

```
// 1. Fenwick Tree
struct Fenwick { // 0-indexed
  int flag, cnt; // array size
  vector<ll> arr, t;
  void build(int n) {
    for (flag = 1; flag < n; flag <<= 1, cnt++);
    arr.resize(flag);
    t.resize(flag);
    for (int i = 0; i < n; i++) cin >> arr[i];
    for (int i = 0; i < n; i++) {
      t[i] += arr[i];
      if (i | (i + 1) < flag) t[i | (i + 1)] += t[i];
    }
  }
  void add(int p, ll value) { // add value at position p
    arr[p] += value;
    while (p < flag) {
      t[p] += value;
      p |= p + 1;
    }
  }
  void modify(int p, ll value) { // set value at position p
```

```
    add(p, value - arr[p]);
  };
  ll query(int x) {
    ll ret = 0;
    while (x >= 0) ret += t[x], x = (x & (x + 1)) - 1;
    return ret;
  }
  ll query(int l, int r) {
    return query(r) - (l ? query(l - 1) : 0);
  }
  int kth(int k) { // find the kth smallest number (1-indexed)
    assert(t.back() >= k);
    int l = 0, r = arr.size();
    for (int i = 0; i <= cnt; i++) {
      int mid = (l + r) >> 1;
      ll val = mid ? t[mid - 1] : t.back();
      if (val >= k) r = mid;
      else l = mid, k -= val;
    }
    return l;
  }
}fw;

// 3. 2D Fenwick Tree
// INPUT: Given an 2D array of integers of size N * M.
// Can modify the value of the (x, y)th element.
// Can find the sum of elements from (sx, sy) to (ex, ey).
// OUTPUT: Given the query (1 sx sy ex ey), output the sum of elements from the interval (sx, sy) to (ex, ey)
// TIME COMPLEXITY: O(N * M) for initialize fenwick tree, O(logN * logM) for each query.
struct Fenwick2D { // 0-indexed
    int n, m, real_n, real_m;
    vector<vector<ll>> arr, t;
    void build(int N, int M) {
        real_n = N, real_m = M;

        n = m = 1;
        while (n < N) n <<= 1;
        while (m < M) m <<= 1;
        arr.resize(n, vector<ll>(m));
        t.resize(n, vector<ll>(m));

        for (int i = 0; i < real_n; i++) {
            for (int j = 0; j < real_m; j++) {
                cin >> arr[i][j];
            }
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                t[i][j] += arr[i][j];

                int ni = i | (i + 1), nj = j | (j + 1);
                if (ni < n) t[ni][j] += t[i][j];
                if (nj < m) t[i][nj] += t[i][j];
                if (ni < n && nj < m) t[ni][nj] -= t[i][j];
            }
        }
    }
    void add(int x, int y, ll value) { // add value at position (x, y)
        assert(0 <= x && x < real_n && 0 <= y && y < real_m);
        arr[x][y] += value;
        for (int i = x; i < n; i |= i + 1) {
            for (int j = y; j < m; j |= j + 1) {
                t[i][j] += value;
            }
        }
    }
    void modify(int x, int y, ll value) { // set value at position (x, y)
        assert(0 <= x && x < real_n && 0 <= y && y < real_m);
        add(x, y, value - arr[x][y]);
    }
    ll query(ll x, ll y) {
        assert(0 <= x && x < real_n && 0 <= y && y < real_m);
        ll ret = 0;
        for (int i = x; i >= 0; i = (i & (i + 1)) - 1) {
            for (int j = y; j >= 0; j = (j & (j + 1)) - 1) {
                ret += t[i][j];
            }
        }
        return ret;
    }
    ll query(ll sx, ll sy, ll ex, ll ey) {
        assert(0 <= sx && sx <= ex && ex < real_n);
        assert(0 <= sy && sy <= ey && ey < real_m);
        ll ret = query(ex, ey);
        if (sx) ret -= query(sx - 1, ey);
        if (sy) ret -= query(ex, sy - 1);
        if (sx && sy) ret += query(sx - 1, sy - 1);
        return ret;
    }
}fw2d;
```

2.3 Li Chao Tree

```
// INPUT: Initially, a 2d plane in which no linear function exists is given.
// Two types of queries are given.
// 1 a b : The linear function f(x) = ax + b is added.
// 2 x : Find the max(f(x)) among the linear functions given so far.
// OUTPUT: For each query 2 x, output the max(f(x)) among the linear functions given so far.
// TIME COMPLEXITY: O(qlogq)
#define Line pair<ll, ll>

const Line e = { 0, -1e18 };
struct LiChaoTree {
    ll f(Line l, ll x) { return l.first * x + l.second; }
    struct Node {
        ll xl, xr; int l, r;
        Line line;
    };
    vector<Node> t;
    void build(ll xlb, ll xub) {
        t.push_back({ xlb, xub, -1, -1, e });
    }
    void insert(Line newLine, int n = 0) {
        ll xl = t[n].xl, xr = t[n].xr;
        ll xmid = (xl + xr) >> 1;
```

```
        Line llow = t[n].line, lhigh = newLine;
        if (f(llow, xl) >= f(lhigh, xl)) swap(llow, lhigh);

        if (f(llow, xr) <= f(lhigh, xr)) {
            t[n].line = lhigh;
            return;
        }
        else if (f(llow, xmid) < f(lhigh, xmid)) {
            t[n].line = lhigh;
            if (t[n].r == -1) {
                t[n].r = sz(t);
                t.push_back({ xmid + 1, xr, -1, -1, e });
            }
            insert(llow, t[n].r);
        }
        else if (f(llow, xmid) >= f(lhigh, xmid)) {
            t[n].line = llow;
            if (t[n].l == -1) {
                t[n].l = sz(t);
                t.push_back({ xl, xmid, -1, -1, e });
            }
            insert(lhigh, t[n].l);
        }
    }
    ll query(ll x, int n = 0) {
        if (n == -1) return e.second;
        ll xl = t[n].xl, xr = t[n].xr;
        ll xmid = (xl + xr) >> 1;

        ll ret = f(t[n].line, x);
        if (x <= xmid) ret = max(ret, query(x, t[n].l));
        else ret = max(ret, query(x, t[n].r));
        return ret;
    }
}lct;

int main() {
    lct.build(-1'000'000'000'0001l, 1'000'000'000'0001l);

    int q; cin >> q;
    while (q--) {
        int op; ll a, b;
        cin >> op >> a;
        if (op == 1) {
            cin >> b;
            lct.insert({ a, b });
        }
        if (op == 2) cout << lct.query(a) << '\n';
    }
}
```

3 Graph

3.1 Bellman-Ford, Floyd-Warshall

```
// 2. Bellman-Ford Algorithm
// INPUT: Given a directed graph with weighted(possibly negative) edges and no negative cycles.
Given a starting vertex.
// OUTPUT: Outputs the shortest distance from the starting vertex to all vertices.
// TIME COMPLEXITY: O(VE)
struct wv {
  ll w; int v;
};
int n, m;
vector<wv> adj[101010];
vector<ll> upper(101010, (ll)1e18);
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; // Returns false <=> The graph has a negative cycle.
}

// 3. Floyd-Warshall Algorithm
// INPUT: Given a directed graph with weighted(possibly negative) edges and no negative cycles.
// OUTPUT: Outputs the shortest distance from all vertices to all vertices.
// TIME COMPLEXITY: O(V^3)
int n, m;
ll adj[1010][1010];
void floyd() {
  for (int i = 0; i < 1010; i++) {
    for (int j = 0; j < 1010; j++) {
      adj[i][j] = (ll)1e18;
    }
  }
  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]);
      }
    }
  }
}
```

3.2 SCC, 2-SAT

```
// 1. SCC (Kosaraju's Algorithm)
// INPUT: Given a directed graph.
// OUTPUT: Decompose this graph into SCCs and print them in lexicographical order.
// TIME COMPLEXITY: O(V + E)

// BOJ 2150 AC Code
// https://www.acmicpc.net/problem/2150

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

```cpp
using namespace std;
#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() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);

  input();
  kosaraju();
  print();
}

// 3. 2-SAT
// INPUT: A 2-CNF is given. 2-CNF is a boolean expression in the form (x ∨ y) ∧ (¬ y ∨ z) ∧
(x ∨ ¬ z) ∧ (z ∨ y).
// OUTPUT: Determine whether there exists a case where a given 2-CNF expression can be true.
(2-Satisfiability Problem)
// TIME COMPLEXITY: O(n + m) = O(n) (m = 2n)

// BOJ 11281 AC Code
// https://www.acmicpc.net/problem/11281

#include <bits/stdc++.h>
using namespace std;
#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 input() {
  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() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);

  memset(dfsn, -1, sizeof(dfsn));
  memset(ans, -1, sizeof(ans));
  input();

  // 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;

  return 0;
}
```

3.3 BCC

```
// A Biconnected Component (BCC) is a subset of vertices in an undirected graph that satisfies
the following conditions:
// (1) If you delete any vertex from a subset, the remaining vertices are connected to each
other.
// (2) Adding other vertices to this subset does not satisfy (1). (This is the largest set that
satisfies (1))
// TIME COMPLEXITY: O(V + E)

// A vertex at which the graph is divided into two or more components when the vertex is removed
is called a 'articulation point'.
// After decomposing the graph into BCCs, vertices belonging to two or more BCCs are
articulation point.
// Similarly, a edge at which the graph is divided into two or more components when the edge is
removed is called a 'articulation edge'.
// For all tree edges on a dfs spanning tree, if tmp > dfsn[now], the edge { now, next } is a
articulation edge.

#include <bits/stdc++.h>
using namespace std;
#define pii pair<int, int>

const int MAXV = 101010;

int n, m;
vector<int> adj[MAXV];
vector<vector<pii>> bcc;
set<int> aPoint;
set<pii> aEdge;

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 dfsn[MAXV], dCnt;
stack<pii> stk;
int dfs(int now, int prv) {
```

```
  int ret = dfsn[now] = ++dCnt;

  int childCnt = 0;
  for (int next : adj[now]) {
    if (next == prv) continue;

    // If an edge { now, next } has not yet been visited, 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 {
      childCnt++;
      int tmp = dfs(next, now);
      ret = min(ret, tmp);

      if (prv != -1 && tmp >= dfsn[now])
        aPoint.insert(now);
      if (tmp > dfsn[now])
        aEdge.insert({ min(now, next), max(now, next) });

      // If next cannot go to ancestor node of now, find BCC
      if (tmp >= dfsn[now]) {
        vector<pii> nowBCC;
        while (true) {
          pii t = stk.top();
          stk.pop();
          nowBCC.push_back(t);
          if (t == make_pair(now, next)) break;
        }
        bcc.push_back(nowBCC);
      }
    }
  }

  if (prv == -1 && childCnt > 1)
    aPoint.insert(now);

  return ret;
}

void getBCC() {
  memset(dfsn, -1, sizeof(dfsn));
  for (int v = 1; v <= n; v++)
    if (dfsn[v] == -1) dfs(v, -1);
}

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

  input();
  getBCC();
}
```

3.4 Euler Circuit

```
// Hierholzer's Algorithm
// INPUT: Given a undirected graph.
// OUTPUT: Print the path of the Euler circuit of the graph.
// Euler Path is a path in a finite graph that visits every edge exactly once.
// Similarly, an Euler Circuit is an Euler Path that starts and ends on the same vertex.
// TIME COMPLEXITY: O(VE)

// BOJ 1199 AC Code
// https://www.acmicpc.net/problem/1199

#include <bits/stdc++.h>
using namespace std;

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);
    }
  }
  eulerCircult.push_back(now);
}

int main() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
```

```
  input();

  if (!doesEulerCircuitExist()) {
    cout << -1;
    return 0;
  }

  for (int i = 1; i <= n; i++) nxt[i] = 1;
  dfs(1);
  for (auto i : eulerCircult)
    cout << i << ' ';
}
```

4 Tree

4.1 Heavy-Light Decomposition

```
// 2. HLD with Segment Tree
#include <bits/stdc++.h>
using namespace std;
#define ll long long
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() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
  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';
  }
}
```

4.2 Centroid Decomposition

```
#include <bits/stdc++.h>
using namespace std;
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() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
  cd(0, -1);
  return 0;
}
```

5 Network Flow

5.1 Dinic's Algorithm

```
// Dinic's Algorithm
// time complexity : O(V^2 * E)
#include <bits/stdc++.h>
using namespace std;
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() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
  input();
  int total = dinic();
  cout << total << '\n';
}
```

5.2 Hopcroft-Karp Algorithm

```
// Bipartite Matching Algorithm
// time complexity : O(E * sqrt(V))
#include <bits/stdc++.h>
using namespace std;
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() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
  input();
  int total = hopcroft();
  cout << total << '\n';
}
```

5.3 MCMF

```
#include <bits/stdc++.h>
using namespace std;
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() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
  input();
  mcmf();
  cout << mFlow << '\n' << mCost;
}
```

6 String

6.1 Rabin-Karp Algorithm

```
// BOJ 1786 AC Code
// https://www.acmicpc.net/problem/1786
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define sz(x) (int)(x).size()
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() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
  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

```
// BOJ 1786 AC Code
// https://www.acmicpc.net/problem/1786
#include <bits/stdc++.h>
using namespace std;
#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

```
// 1. Trie (Array Index)
#include <bits/stdc++.h>
using namespace std;
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() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
  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";
  }
}

// 2. Trie (Pointer)
#include <bits/stdc++.h>
using namespace std;
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() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
  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;
}
```

6.4 Aho-Corasick

```
// BOJ 9250 AC Code
// https://www.acmicpc.net/problem/9250
#include <bits/stdc++.h>
using namespace std;
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() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
  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)
// BOJ 9248 AC Code
// https://www.acmicpc.net/problem/9248
#include <bits/stdc++.h>
using namespace std;
#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() {
    cin.tie(NULL); cout.tie(NULL);
    ios_base::sync_with_stdio(false);
    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)
// BOJ 9248 AC Code
// https://www.acmicpc.net/problem/9248
#include <bits/stdc++.h>
using namespace std;
#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() {
    cin.tie(NULL); cout.tie(NULL);
    ios_base::sync_with_stdio(false);
    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
#include <bits/stdc++.h>
using namespace std;
#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.tie(NULL); cout.tie(NULL);
    ios_base::sync_with_stdio(false);
    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
#include <bits/stdc++.h>
using namespace std;
#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() {
    cin.tie(NULL); cout.tie(NULL);
    ios_base::sync_with_stdio(false);
    input();
    zfunction();
    int q; cin >> q;
    while (q--) {
```

```
        int x; cin >> x;
        cout << z[sz(s) - x] << '\n';
    }
}
```

7 Geometry

7.1 Convex Hull

```
// 1. Graham Scan
// 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);
  }
}

// 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;
}
```

7.2 Rotating Callipers

```
// BOJ 10254 AC Code
// https://www.acmicpc.net/problem/10254
#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;
        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() {
    cin.tie(NULL); cout.tie(NULL);
    ios_base::sync_with_stdio(false);
    int tc; cin >> tc;
    while (tc--) {
        init();
        input();
        rotatingCallipers();
        cout << ans1.x << ' ' << ans1.y << ' ';
        cout << ans2.x << ' ' << ans2.y << '\n';
    }
}
```

7.3 Ray Casting

```
// BOJ 1688 AC Code
// https://www.acmicpc.net/problem/1688
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#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 = { 10101010101ll, 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() {
    cin.tie(NULL); cout.tie(NULL);
    ios_base::sync_with_stdio(false);
    input();
    cout << (insidePolygon(a) ? 1 : 0) << '\n';
    cout << (insidePolygon(b) ? 1 : 0) << '\n';
    cout << (insidePolygon(c) ? 1 : 0) << '\n';
}
```

7.4 Sort by Angular

```
// Sort by Angular (Relative to Origin)
struct Point {
    ll x, y;
    bool operator<(const Point& rhs) const {
        return x ^ rhs.x ? x < rhs.x : y < rhs.y;
    }
};

const Point o = { 0, 0 };
vector<Point> p;

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));
}
inline ll dist(const Point& v) { return v.x * v.x + v.y * v.y; }

// If the angle between any two points and the origin is less than 180 degrees,
// they can be sorted through the cross product of the vectors.
// Therefore, the points were divided into 1st and 4th quadrants and 2nd and 3rd quadrants,
// and the points in the 1st and 4th quadrants were arranged in front.
void sortByAngular() {
    sort(p.begin(), p.end(), [&](const Point& lhs, const Point& rhs) {
        if ((lhs < o) ^ (rhs < o)) return (lhs < o) < (rhs < o);
        if (ccw(o, lhs, rhs)) return ccw(o, lhs, rhs) > 0;
        return dist(lhs) < dist(rhs);
    });
}
```

7.5 Bulldozer Trick

```
// Bulldozer Trick
// There are a total of O(N^2) results of sorting points on a two-dimensional plane based on an arbitrary axis.
// The Bulldozer Trick traverses all O(N^2) results in O(N^2logN) time.

struct Point {
    ll x, y;
    bool operator<(const Point& rhs) const {
        return tie(x, y) < tie(rhs.x, rhs.y);
    }
};

struct Line {
    int u, v; ll dx, dy; // u < v, dx >= 0;
    bool operator<(const Line& rhs) const {
        if (dy * rhs.dx != rhs.dy * dx) return dy * rhs.dx < rhs.dy * dx;
        return tie(u, v) < tie(rhs.u, rhs.v);
    }
    bool operator==(const Line& rhs) const {
        return dy * rhs.dx == rhs.dy * dx;
    }
};

int n, pos[2020];
Point p[2020];

void bulldozerTrick() {
    sort(p + 1, p + 1 + n);
    for (int i = 1; i <= n; i++) pos[i] = i;
```

```
    // find the slope between every two points.
    vector<Line> arr;
    for (int i = 1; i <= n; i++) {
        for (int j = i + 1; j <= n; j++) {
            arr.push_back({ i, j, p[j].x - p[i].x, p[j].y - p[i].y });
        }
    }
    sort(arr.begin(), arr.end());

    // can check one of the results of sorting points at here.

    for (int i = 0, j = 0; i < arr.size(); i = j) {
        while (j < arr.size() && arr[j] == arr[i]) j++; // all lines in [i, j) are same
        for (int k = i; k < j; k++) {
            int u = arr[k].u, v = arr[k].v;
            swap(p[pos[u]], p[pos[v]]);
            swap(pos[u], pos[v]);
        }

        // can check one of the results of sorting points at here.
    }
}
```

7.6 Minimum Enclosing Circle

```
// INPUT: Given N points in a 2D plane with integer coordinates.
// OUTPUT: Find the center and the radius of the minimum enclosing circle.
// A minimum enclosing circle is a circle in which all the points lie either inside the circle
or on its boundaries.
// TIME COMPLEXITY: O(N) (using random)

// BOJ 2626 AC Code
// https://www.acmicpc.net/problem/2626
#include <bits/stdc++.h>
using namespace std;

struct Point { long double x, y; };
struct Circle { Point c; long double r; };

long double dist(const Point& a, const Point& b) {
    return sqrt(pow(b.x - a.x, 2) + pow(b.y - a.y, 2));
}

Point getCircleCenter(const Point& a, const Point& b) {
    long double A = a.x * a.x + a.y * a.y;
    long double B = b.x * b.x + b.y * b.y;
    long double C = a.x * b.y - a.y * b.x;
    return { (b.y * A - a.y * B) / (2 * C), (a.x * B - b.x * A) / (2 * C) };
}
Circle circleFrom(const Point& a, const Point& b, const Point& c) {
    Point i = getCircleCenter({ b.x - a.x, b.y - a.y }, { c.x - a.x, c.y - a.y });
    i.x += a.x;
    i.y += a.y;
    return { i, dist(a, i) };
}
Circle circleFrom(const Point& a, const Point& b) {
    Point c = { (a.x + b.x) / 2.0, (a.y + b.y) / 2.0 };
    return { c, dist(a, b) / 2.0 };
}

Circle minimumEnclosingCircle(int n, const vector<Point>& p) {
    Circle ret = { { 0, 0 }, 0 };
    for (int i = 0; i < n; i++) {
        if (dist(ret.c, p[i]) <= ret.r) continue;
        ret.c = p[i], ret.r = 0;
        for (int j = 0; j < i; j++) {
            if (dist(ret.c, p[j]) <= ret.r) continue;
            ret = circleFrom(p[i], p[j]);
            for (int k = 0; k < j; k++) {
                if (dist(ret.c, p[k]) <= ret.r) continue;
                ret = circleFrom(p[i], p[j], p[k]);
            }
        }
    }
    return ret;
}

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

    int n; cin >> n;
    vector<Point> p(n);
    for (auto& i : p)
        cin >> i.x >> i.y;

    random_shuffle(p.begin(), p.end());

    Circle ans = minimumEnclosingCircle(n, p);

    cout << fixed;
    cout.precision(3);
    cout << ans.c.x << ' ' << ans.c.y << '\n' << ans.r;
}
```

8 Math

8.1 Sieve

```
// Sieve of Eratosthenes
// TIME COMPLEXITY: O(Nlog(log(N)))
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 += 2) {
    if (!isPrime[i]) continue;
    prime.push_back(i);
    for (ll j = i * i; j <= MAX; j += i * 2)
      isPrime[j] = 0;
  }
}

// Linear Sieve
// BOJ 16563 AC Code
// https://www.acmicpc.net/problem/16563
```

```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long
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() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
  linearSieve();
  int n; cin >> n;
  while (n--) {
    int x; cin >> x;
    factorization(x);
  }
}
```

8.2 Euclidean Algorithms

```cpp
// 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);
}

// Extended GCD

// Bezout's Lemma
// Let a and b be integers with d := gcd(a, b).
// Then there exist integers x and y such that ax + by = d.
// Moreover, the integers of the form ax + by 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.

// Results from Bezout's Lemma
// (1) : gcd(a,b) = d => gcd(a/d, b/d) = 1.
// (2) : gcd(a,b) = 1, a|c, b|c => ab|c.
// (3) : a|bc, gcd(a,b) = 1 => a|c.
// (4) : d|a, d|b <=> d|gcd(a,b)
// (5) : gcd(ab,ac) = |a|gcd(b,c)
// (6) : a|bc <=> (a/gcd(a,b))|c

// Linear Congreunce
// Finding set of x satisfies ax = b (mod n).
// We can think ax = b (mod n) as ax + ny = b.
// Then we need to find all (x, y) that satisfy ax + ny = b.
// ax + ny = b has a solution iff gcd(a,n) | b

// Modular Inverse
// In particular, if gcd(a,n) = 1, then we can find a unique x in (mod n) such that ax = 1 (mod n).
// This is called the modular inverse of a over (mod n), sometimes written a^{-1}.
// Multiplying the modular inverse is equivalent to dividing by a.

// TIME COMPLEXITY: O(log(AB))

// BOJ 14565 AC Code
// https://www.acmicpc.net/problem/14565
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define pll pair<ll, ll>
pair<pll, ll> egcd(ll a, ll b) {
  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;
  }
  // a * olds + b * oldt = d
  // oldr = gcd(a, b)
  return { { olds, oldt }, oldr };
}
ll linearCongruence(ll a, ll b, ll n) { // Find x such that ax = b (mod n).
  pair<pll, ll> res = egcd(a, n);
  // ax + ny = b has a solution iff gcd(a,n) | b.
  if (b % res.second) return -1;
  return (res.first.first + n) % n;
}
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.second == 1) return (res.first.first + p) % p;
  else return -1;
}
```

```
int main() {
  ll N, A;
  cin >> N >> A;
  cout << N - A << ' ' << modInv(A, N);
}
```

8.3 Fermat's Little Theorem

```
// BOJ 11401 AC Code
// https://www.acmicpc.net/problem/11401
#include <bits/stdc++.h>
using namespace std;
#define ll long long
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() {
    cin.tie(NULL); cout.tie(NULL);
    ios_base::sync_with_stdio(false);
    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;
}
```

8.4 Euler's Phi Function

```
// INPUT: Given a natural number n.
// OUTPUT: Find the number of natural numbers 1 ≤ k ≤ n such that GCD(n, k) = 1.
// TIME COMPLEXITY: O(sqrt(n))
// BOJ 11689 AC Code
// https://www.acmicpc.net/problem/11689
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define pll pair<ll, ll>
#define fr first
#define sc second
ll phi(ll x) { // Find phi(x) in O(sqrt(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.
    // phi(mn) = phi(m) * phi(n) if gcd(m, n) = 1
    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);
}
```

8.5 Chinese Remainder Theorem

```
// INPUT: Given 4 integers, M, N, p, q. (1 <= M, 1 <= N, 0 <= p < M, 0 <= q < N)
// OUTPUT: Solve a system of linear congruence, x = p (mod M), x = q (mod N).
// TIME COMPLEXITY: O(log(max(M, N)))
// BOJ 6064 AC Code
// https://www.acmicpc.net/problem/6064
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define pll pair<ll, ll>
#define fr first
#define sc second

ll gcd(ll x, ll y) {
  if (!y) return x;
  return gcd(y, x % y);
}
ll minv(ll x, ll y) {
  if (x == 0 && y == 1) return 0;
  if (x == 1) return 1;
  return y - minv(y % x, x) * y / x;
}

// x = U.fr (mod U.sc)
// x = V.fr (mod V.sc)
// returns solutions as x = ret.fr (mod ret.sc)
// if no solution, returns { -1, -1 }
pll crt(pll U, pll V) {
  if (U.sc == -1 || V.sc == -1) return { -1, -1 };
  if (U.sc == 1) return V;
  if (V.sc == 1) return U;
  ll g = gcd(U.sc, V.sc);
  ll l = U.sc * V.sc / g;
  // (U and V have a solution) iff (U.fr = U.sc (mod gcd(U.sc,V.sc)))
  // also the solution is unique in the range [0, lcm(U.sc,V.sc)).
  if ((V.fr - U.fr) % g) return { -1, -1 };

  ll u = U.sc / g, v = V.sc / g;
  ll mul = (V.fr - U.fr) / g;
  mul = mul * minv(u % v, v) % v;

  pll ret = { mul * U.sc + U.fr, l };
  ret.fr %= ret.sc, ret.fr = (ret.fr + ret.sc) % ret.sc;
  return ret;
}
pll solvingSystemOfLinearCongruence(const vector<pll>& a) {
```

```
  if (a.size() == 1) return a[0];
  pll ret = crt(a[0], a[1]);
  for (int i = 2; i < a.size(); i++) ret = crt(ret, a[i]);
  return ret;
}

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

  int tc; cin >> tc;
  while (tc--) {
    ll M, N, p, q;
    cin >> M >> N >> p >> q;
    p--, q--; // from the given input, 1 <= p <= M, 1 <= q <= N
    vector<pll> cg;
    cg.push_back({ p, M });
    cg.push_back({ q, N });
    pll ans = solvingSystemOfLinearCongruence(cg);
    cout << (ans.fr == -1 ? ans.fr : ans.fr + 1) << '\n';
  }
}
```

8.6 Binomial Coefficient

```
// nCr in O(r)
// Beware of integer overflow
ll binom(int n, int r) {
  if (r < 0 || n < r) return 0;
  r = min(r, n - r);
  ll ret = 1;
  for (ll i = 1; i <= r; i++) {
    ret *= n + 1 - i;
    ret /= i;
  }
  return ret;
}

// nCr (Pascal's Rule)
ll binomDP[1010][1010];
void init() {
  for (int i = 0; i < 1010; i++) {
    for (int j = 0; j < 1010; j++) {
      binomDP[i][j] = -1;
    }
  }
}
ll binom(int n, int r) {
  if (r < 0 || n < r) return 0;
  ll& ret = binomDP[n][r];
  if (ret != -1) return ret;
  if (n == 1) return ret = 1;
  return binom(n - 1, r - 1) + binom(n - 1, r);
}

// nCr mod p in O(1)
// BOJ 13977 AC Code
// https://www.acmicpc.net/problem/13977
#include <bits/stdc++.h>
using namespace std;
#define ll long long
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() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
  // 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

```
// BOJ 11444 AC Code
// https://www.acmicpc.net/problem/11444
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#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;
    }
};
Matrix matrixPower(const Matrix& val, ll exp) {
    if (exp == 1) return val;
    Matrix res = matrixPower(val, exp / 2);
    Matrix ret = res * res;
```

```
    if (exp & 1) ret = ret * val;
    return ret;
}
int main() {
    ll n; cin >> n;
    if (n == 1) {
        cout << 1;
        return 0;
    }
    // Base Matrix
    Matrix base;
    base.a.resize(2, vector<ll>(2));
    base.a[0][0] = base.a[0][1] = base.a[1][0] = 1;
    // Matrix Exponentiation
    Matrix ans = matrixPower(base, n - 1);
    cout << ans.a[0][0];
}
```

8.8 Catalan Number, Derangement Number

```
// Catalan Number
// BOJ 9343 AC Code
// https://www.acmicpc.net/problem/9343
#include <bits/stdc++.h>
using namespace std;
#define ll long long
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() {
    cin.tie(NULL); cout.tie(NULL);
    ios_base::sync_with_stdio(false);
    // 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)
```

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int MOD = 1e9 + 7;
const int MAX = 101010;
ll dp[MAX];
int main() {
    cin.tie(NULL); cout.tie(NULL);
    ios_base::sync_with_stdio(false);
    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 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());
}
```

8.10 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 DP Opt

```
// 1. Convex Hull Trick
// Recurrence: dp[i] = min(dp[j] + a[i] * b[j]) (j < i)
// Condition: b[i] >= b[i + 1]
// Naive Complexity: O(n^2)
// Optimized Complexity: O(nlogn) (if a[i] <= a[i + 1], it can also be done in O(n))

// BOJ 13263 AC Code
// https://www.acmicpc.net/problem/13263
#include <bits/stdc++.h>
using namespace std;
#define ll long long
struct Line { // f(x) = px + q, x >= s
    ll p, q;
    double s;
    Line(): Line(1, 0) {}
    Line(ll sp, ll sq): p(sp), q(sq), s(0) {}
};
double cross(const Line& u, const Line& v) {
    return (double)(v.q - u.q) / (u.p - v.p);
}
int n;
ll a[101010], b[101010];
ll dp[101010];
Line 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++) {
        Line 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() {
    cin.tie(NULL); cout.tie(NULL);
    ios_base::sync_with_stdio(false);
    input();
    convexHullTrick();
    cout << dp[n];
}

// 2. 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
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const ll INF = 1e18;
int n, opt[5050][5050];
ll a[5050], DP[5050][5050], psum[5050];
int main() {
    cin.tie(NULL); cout.tie(NULL);
    ios_base::sync_with_stdio(false);
    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';
    }
}

// 3. Divide and Conquer Optimization
// Recurrence: dp[t][i] = min(dp[t - 1][j] + c[j][i]) (j < i)
// 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].
// Naive Complexity: O(m * n^2)
// Optimized Complexity: O(m * nlogn)

// BOJ 13261 AC Code
// https://www.acmicpc.net/problem/13261
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const ll INF = 1e18;
int n, m;
ll a[8080], psum[8080];
ll dp[808][8080];
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++) {
        assert(i <= mid);
        ll val = dp[gr - 1][i] + (mid - i) * (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() {
    cin.tie(NULL); cout.tie(NULL);
    ios_base::sync_with_stdio(false);
    // input
```

```
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> a[i];
    // build prefix sum
    for (int i = 1; i <= n; i++)
        psum[i] = a[i] + psum[i - 1];
    // dp (dnc opt)
    for (int i = 1; i <= n; i++)
        dp[1][i] = i * psum[i];
    for (int i = 2; i <= m; i++)
        f(i, 0, n, 0, n);
    // output
    cout << dp[m][n];
}
```

9.2 Sqrt Decomposition, Mo's Algorithm

```
#include <bits/stdc++.h>
using namespace std;
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;
  }
};
// Zigzag Mo's (faster than basic Mo's Algorithm)
/*struct se {
  int s, e, idx;
  bool operator<(const se &rhs) const {
    if(s / sq != rhs.s / sq) return s / sq < rhs.s / sq;
    else return (s / sq) & 1 ? e < rhs.e : 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() {
  cin.tie(NULL); cout.tie(NULL);
  ios_base::sync_with_stdio(false);
  input();
  sort(q.begin(), q.end());
  f();
  for (auto& i : ans)
    cout << i << '\n';
}
```

9.3 Rotation Matrix, Manhattan Distance, Chebyshev Distance

```
Rotation Matrix (In Two Dimensions)

x' = x * cosθ - y * sinθ
y' = x * sinθ + y * cosθ

Manhattan Distance, Chebyshev Distance

For any two points (x1, y1), (x2, y2) on two-dimensional coordinates,
- the Manhattan distance is |x1 - x2| + |y1 - y2|.
- the Chebyshev distance is max(|x1 - x2|, |y1 - y2|).

Relationship between Manhattan distance and Chebyshev distance

Let x1' = x1 - y1, y1' = x1 + y1,
x2' = x2 - y2, y2' = x2 + y2. (using Rotation matrix)

Then max(|x1 - x2|, |y1 - y2|)
= (|x1' - x2'| + |y1' - y2'|) / 2.

Thus, Manhattan distance can be solved by replacing it with Chebyshev distance, and vice versa.
```

9.4 Random

```
#include <random>
using namespace std;
const int MAX = 1'000;
srand(time(NULL));
int x = rand() % MAX;
```

9.5 Ternary Search

```
void ternarySearch() {
    int l = 1, r = 10'000;
    while (r - l >= 3) {
        int mid1 = (2 * l + r) / 3;
        int mid2 = (l + 2 * r) / 3;
        int res1 = f(mid1), res2 = f(mid2);
        if (res1 <= res2) l = mid1;
        else r = mid2;
    }
    int res = 0;
    for (int i = l; i <= r; i++)
        res = max(res, f(i));
}
```