Team Note of tony9402

Compiled on February 17, 2025

Contents

1 Data Structure 1
1.1 Union Find Roll Back 1

2 Graph 1
2.1 Dinic 1
2.2 Mcmf 2
2.3 2Sat 2
2.4 Scc 2
2.5 Dominator Tree 2
2.6 Dynamic Segment Tree 3
2.7 Dynamic Segment Tree With Lazy 3
2.8 Fenwick 3
2.9 Gomory Hu 4
2.10 Hld 4
2.11 Lca 4
2.12 Pst 4
2.13 Segment Tree 5
2.14 Segment Tree With Lazy 5

3 Others 5
3.1 Fastinput 5
3.2 Main 5

4 String 6
4.1 Aho Corasick 6
4.2 Kmp 6
4.3 Z 6

5 Geometry 6
5.1 Ccw 6

1 Data Structure

1.1 Union Find Roll Back

```
struct UnionFind {
  vector<int> par, rank;
  stack<tuple<int, int, int>> st;
  UnionFind(int N) {
    par = rank = vector<int>(N + 1);
    iota(par.begin(), par.end(), 0);
  }

  int find(int x) { return par[x] == x ? x : find(par[x]); }
  bool merge(int u, int v) {
    u = find(u); v = find(v);
    if(u == v) return false;
    if(rank[u] < rank[v]) swap(u, v);
    par[v] = u;
    st.emplace(u, v, rank[u] == rank[v]);
    if(rank[u] == rank[v]) ++rank[u];
    return true;
  }
  void revert(int cnt) {
    while(cnt --> 0) {
      auto [u, v, c] = st.top(); st.pop();
      par[v] = v;
      if(c) -- rank[u];
    }
  }
  int conn(int u, int v) { return find(u) == find(v); }
};
```

2 Graph

2.1 Dinic

```
struct Dinic {
  struct Node {
    int node_idx, cost, flow, rev;
    Node(int _nxt = -1, int _cost = 0, int _rev =
    -1):node_idx(_nxt),cost(_cost),flow(0),rev(_rev) { }
    int spare() { return cost - flow; }
    void setRev(int _rev) { rev = _rev; }
  };
  vector<Node> nodes;
  vector<vector<int>> G;
  vector<int> level;
  vector<int> work;

  int src, snk, asrc, asnk, N;
  Dinic(int _N) {
    src = _N + 1;
    snk = src + 1;
    asrc = snk + 1;
    asnk = asrc + 1;
    N = asnk;
    G.resize(N + 1);
  }
  bool bfs(int s, int e) {
    level = vector<int>(N + 1, -1);
    level[s] = 0;
    queue<int> Q; Q.push(s);
    while(!Q.empty()) {
      int cur = Q.front(); Q.pop();
      for(const int &x: G[cur]) {
        Node &nxt = nodes[x];
        if(nxt.spare() > 0 && level[nxt.node_idx] == -1) {
          level[nxt.node_idx] = level[cur] + 1;
          Q.push(nxt.node_idx);
        }
      }
    }
    return ~level[e];
  }
  int dfs(int s, int e, int f) {
    if(s == e) return f;
    for(int &i = work[s]; i < (int)G[s].size(); ++i) {
      Node &nxt = nodes[G[s][i]];
      if(nxt.spare() > 0 && level[nxt.node_idx] == level[s] +
      1) {
        int ret = dfs(nxt.node_idx, e, min(f, nxt.spare()));
        if(ret > 0) {
          nxt.flow += ret;
          nodes[nxt.rev].flow -= ret;
          return ret;
        }
      }
    }
    return 0;
  }
  int flow(int s, int e) {
    int ret = 0;
    while(bfs(s, e)) {
      work = vector<int>(N + 1, 0);
      while(true) {
        int x = dfs(s, e, numeric_limits<int>::max());
        if(x == 0) break;
        ret += x;
      }
    }
    return ret;
  }
  void addEdge(int u, int v, int cost, bool is_directed =
  true, bool is_unique = false) {
    if(is_unique) {
      for(const int &x: G[u]) {
        if(nodes[x].node_idx == v) {
          nodes[x].cost += cost;
          if(!is_directed) return;
          break;
        }
      }
      if(!is_directed) {
        for(const int &x: G[v]) {
          if(nodes[x].node_idx == u) {
            nodes[x].cost += cost;
            return;
          }
        }
      }
    }
    int a = (int)nodes.size(), b = a + 1;
    Node uv = Node(v, cost, b);
    Node vu = Node(u, is_directed ? 0 : cost, a);
    nodes.push_back(uv); nodes.push_back(vu);
    G[u].push_back(a); G[v].push_back(b);
  }
  void addLREdge(int u, int v, int lower, int upper) {
    if(lower) {
      addEdge(asrc, v, lower);
      addEdge(u, asnk, lower);
    }
    addEdge(u, v, upper - lower);
  }
  int flow() { return flow(src, snk); }
  int lrflow() { return flow(asrc, asnk); }
};
```

2.2 Mcmf

```
template<typename T>
struct MinCostMaxFlow {
  struct Edge {
    int edge_id, node_idx, cost, flow, rev;
    T dist;
    Edge(int _edge_id, int _node_idx, int _cost, T _dist, int
    _rev):edge_id(_edge_id),node_idx(_node_idx),cost(_cost),flow(0),dist(_dist),rev(_rev)
    { }
    int spare() { return cost - flow; }
  };

  vector<Edge> edges;
  vector<vector<int>> G;
  vector<pair<int, int>> par;
  vector<T> dist;

  int src, snk, N;
  T INF;

  MinCostMaxFlow(int _N) {
    src = _N + 1;
    snk = src + 1;
    N = snk;

    INF = numeric_limits<T>::max();

    G.resize(N + 1);
    par.resize(N + 1, make_pair(-1, -1));
  }
  bool spfa(int s, int e) {
    vector<int> InQ(N + 1);
    dist = vector<T>(N + 1, INF);

    dist[s] = 0;
    deque<int> dq; dq.push_back(s);
    InQ[s] = 1;

    while(!dq.empty()) {
      int cur = dq.front(); dq.pop_front();
      InQ[cur] = 0;
      for(const int &x: G[cur]) {
        Edge &e = edges[x];
        if(e.spare() > 0 && dist[e.node_idx] > dist[cur] +
        e.dist) {
          dist[e.node_idx] = dist[cur] + e.dist;
          par[e.node_idx] = make_pair(cur, e.edge_id);
          if(InQ[e.node_idx] == 0) {
            dq.push_back(e.node_idx);
            InQ[e.node_idx] = 1;
          }
        }
      }
    }
    return dist[e] != INF;
  }

  // min_cost, max_flow
  pair<T, int> flow_after_spfa(int s, int e) {
    int mn = numeric_limits<int>::max();
    for(int cur = e; cur != s; cur = par[cur].first) {
      mn = min(mn, edges[par[cur].second].spare());
    }
    if(mn == 0) return make_pair<T, int>(-1, -1);
    T min_cost = 0;
    int max_flow = mn;
    for(int cur = e; cur != s; cur = par[cur].first) {
      min_cost += (T)mn * edges[par[cur].second].dist;
      edges[par[cur].second].flow += mn;
      edges[edges[par[cur].second].rev].flow -= mn;
    }
    return make_pair(min_cost, max_flow);
  }
  pair<T, int> flow(int s, int e) {
    pair<T, int> ret;
    while (spfa(s, e)) {
      pair<T, int> cur = flow_after_spfa(s, e);
      if (cur.first == -1) break;
      ret.first += cur.first;
      ret.second += cur.second;
    }
    return ret;
  }

  // addEdge
  void addEdge(int u, int v, int cost, T dist) {
    int a = edges.size();
    int b = a + 1;

    Edge uv = Edge(a, v, cost, dist, b);
    Edge vu = Edge(b, u, 0, -dist, a);

    edges.push_back(uv);
    edges.push_back(vu);

    G[u].push_back(a);
    G[v].push_back(b);
  }

  pair<T, int> flow() { return flow(src, snk); }
};
```

2.3 2Sat

```
// 1-indexed, a xor b = (a or b) and (¬a or ¬b)
int getIdx(int x) { return abs(x) << 1 | (x < 0); }
void addEdge(Graph<int> &G, int u, int v) {
  u = getIdx(u), v = getIdx(v);
  G.addEdge(u ^ 1, v); G.addEdge(v ^ 1, u);
}
bool avaiable(Graph<int> &G) {
  SCC scc(G);
  int N = G.size() - 2 >> 1;
  for(int i = 1; i <= N; ++i) {
    if(scc.scc_id[i << 1] == scc.scc_id[i << 1 | 1]) return
    false;
  }
  return true;
}
```

2.4 Scc

```
// 1-indexed, Need Graph template
struct SCC {
  int N, id;
  Graph<int> G;
  vector<int> D, scc_id;
  vector<vector<int>> scc;
  stack<int> st;

  SCC(const Graph<int> &_G):G(_G) {
    id = 0;
    N = G.size();
    D.resize(N + 1);
    scc_id.resize(N + 1, -1);
    for(int i = 1; i <= N; ++i) if(!D[i]) dfs(i);
  }
  int dfs(int cur) {
    D[cur] = ++id;
    st.push(cur);
    int par = D[cur];
    for(const auto &nxt: G[cur]) {
      if(!D[nxt]) par = min(par, dfs(nxt));
      else if(scc_id[nxt] == -1) par = min(par, D[nxt]);
    }
    if(par == D[cur]) {
      scc.emplace_back();
      while(!st.empty()) {
        int x = st.top(); st.pop();
        scc_id[x] = (int)scc.size() - 1;
        scc.back().push_back(x);
        if(x == cur) break;
      }
    }
    return par;
  }
  int size() { return scc.size(); }
  vector<int> &operator[] (const int idx) { return scc[idx]; }
  Graph<int> graph() {
    int K = size();
    Graph<int> sccG(K);
    for(int i = 1; i <= N; ++i) {
      for(const int &nxt: G[i]) {
        if(scc_id[i] == scc_id[nxt]) continue;
        sccG.addEdge(scc_id[i], scc_id[nxt]);
      }
    }
    for(int i = 0; i < K; ++i) {
      sort(sccG[i].begin(), sccG[i].end());
      sccG[i].erase(unique(sccG[i].begin(), sccG[i].end()),
      sccG[i].end());
    }
    return sccG;
  }
};
```

2.5 Dominator Tree

```
vector<int> DominatorTree(const vector<vector<int>> &G, int
start_node) {
  int N = (int)G.size();
  vector<vector<int>> rG(N);
  for (int cur = 0; cur < N; ++cur) {
    for (int nxt : G[cur]) rG[nxt].push_back(cur);
  }

  vector<int> uf(N), sdom_id(N), idom(N, -1), sdom(N, -1);
  for (int i = 0; i < N; ++i) uf[i] = sdom_id[i] = i;
  function<int(int)> find = [&](int x) -> int {
    if (uf[x] == x) return x;
    int tmp = find(uf[x]);
    if (sdom[sdom_id[x]] > sdom[sdom_id[uf[x]]]) sdom_id[x] =
    sdom_id[uf[x]];
    return uf[x] = tmp;
  };
  vector<int> numbering, par(N);
  function<void(int)> dfs = [&](int cur) -> void {
    sdom[cur] = numbering.size();
    numbering.push_back(cur);
    for (int nxt : G[cur]) {
      if (sdom[nxt] != -1) continue;
      par[nxt] = cur;
      dfs(nxt);
    }
  };
  dfs(start_node);
  int K = (int)numbering.size();
  vector<vector<int>> buf(N);
  vector<int> final_uf(N);
  for (int i = K - 1; i >= 0; --i) {
    int u = numbering[i];
    if (sdom[u] == -1) continue;
    for (int v : rG[u]) {
      if (sdom[v] == -1) continue;
      find(v);
      if (sdom[u] > sdom[sdom_id[v]]) sdom[u] =
      sdom[sdom_id[v]];
    }
    buf[numbering[sdom[u]]].push_back(u);
    for (int nxt : buf[par[u]]) {
      find(nxt);
      final_uf[nxt] = sdom_id[nxt];
    }
    buf[par[u]].clear();
    uf[u] = par[u];
  }
  idom[start_node] = start_node;
  for (const int &x : numbering) {
    if (sdom[x] == sdom[final_uf[x]]) idom[x] = sdom[x];
    else idom[x] = idom[final_uf[x]];
  }
  for (const int &x : numbering) {
    if (x != start_node) idom[x] = numbering[idom[x]];
  }
  return idom;
}
```

2.6 Dynamic Segment Tree

```
const int MAXL = 1000000000;
template<typename T>
struct DynamicSegment{
  struct Node{
    int l, r; // range
    T data;
    Node *left, *right;
    Node():l(1),r(MAXL),data(0),left(nullptr),right(nullptr) {
    }
    void extend(){
      if(l == r)return;
      if(left == nullptr){ //if leaf node
        left  = new Node();
        right =  new Node();
        int mid = (l + r) / 2;
        left->l  = l;
        left->r  = mid;
        right->l = mid + 1;
        right->r = r;
      }
      return;
    }
  };
  Node *tree;

  DynamicSegment() { tree = new Node(); }
  void update(Node *cur, int x, T data){
    if(x < cur->l || cur->r < x)return;
    if(cur->l == cur->r)return cur->data = data, (void)0;
    cur->extend();
    update(cur->left, x, data);
    update(cur->right, x, data);
    cur->data = mergeNode(cur->left->data, cur->right->data);
  }
  void update(int x, T data){ update(tree, x, data); }

  T query(Node *cur, int l, int r){
    if(cur->l > cur->r || cur->r < l || cur->l > r)return
    T(0);
    if(l <= cur->l && cur->r <= r)return cur->data;
    cur->extend();
    return mergeNode(query(cur->left, l, r), query(cur->right,
    l, r));
  }
  T query(int l, int r){ return query(tree, l, r); }
  T mergeNode(T a, T b){ return a + b; }
};
DynamicSegment<long long> tree;
```

2.7 Dynamic Segment Tree With Lazy

```
const int MAXL = 1000000000;
template<typename T>
struct DynamicSegmentLazy{
  struct Node{
    int l, r; // range
    T data, lazy;
    Node *left, *right;
    Node():l(1),r(MAXL),data(0),lazy(0),left(0),right(0) { }
    void extend(T lzy=0){
      if(l == r)return;
      if(left == 0){ //if leaf node
        left  = new Node();
        right =  new Node();
        int m = (l + r) / 2;
        left->l  = l;
        left->r  = m;
        right->l = m + 1;
        right->r = r;
      }
      left->lazy += lzy;
      right->lazy += lzy;
      return;
    }
  };
  Node *tree;

  DynamicSegmentLazy() { tree = new Node(); }
  void pushdown(Node *cur){
    if(cur->lazy){
      cur->data += (cur->r - cur->l + 1) * cur->lazy;
      cur->extend(cur->lazy);
      cur->lazy = 0;
    }
  }

  void update(Node *cur, int l, int r, T data){
    pushdown(cur);
    if(cur->l > cur->r || cur->l > r || l > cur->r)return;
    if(l <= cur->l && cur->r <= r){
      cur->data += (cur->r - cur->l + 1) * data;
      if(cur->l != cur->r)cur->extend(data);
      return;
    }
    cur->extend();
    update(cur->left, l, r, data);
    update(cur->right, l, r, data);
    cur->data = mergeNode(cur->left->data, cur->right->data);
  }
  void update(int l, int r, T data){ update(tree, l, r, data);
  }

  T query(Node *cur, int l, int r){
    if(cur->l > cur->r || cur->l > r || l > cur->r)return
    T(0);
    pushdown(cur);
    if(l <= cur->l && cur->r <= r)return cur->data;
    cur->extend();
    return mergeNode(query(cur->left, l, r), query(cur->right,
    l, r));
  }
  T query(int l, int r){ return query(tree, l, r); }
  T merge(T a, T b) {
    return a + b;
  }
};
```

2.8 Fenwick

```
template<typename T> struct Fenwick {
  int N;
  vector<T> tree;
  Fenwick(int _N):N(_N) { tree.resize(N + 1); }
  void update(int idx, T data) {
    for( ; idx <= N; idx += idx & -idx) tree[idx] += data;
  }
  T query(int idx) {
    T ret = 0;
    for( ; idx; idx -= idx & -idx) ret += tree[idx];
    return ret;
  }
  T query(int l, int r) {
    return query(r) - query(l - 1);
  }
};
```

2.9 Gomory Hu

```
vector<int> par(N);
int ans = 0;
for (int i = 1; i < N; ++i) {
  Dinic dinic(N);
  for (auto [u, v]: edges) dinic.addEdge(u, v, 1, false);
  int src = i, snk = par[i];
  int flow = dinic.flow(src, snk);
  ans = max(ans, flow);
  for (int j = i + 1; j < N; ++j) {
    if (dinic.level[j] != -1 && par[j] == par[i]) par[j] = i;
  }
}
```

2.10 Hld

```
struct HLD {
  Graph<int> G;
  vector<int> par, top, dep, siz, in, out;
  Segment<int> seg; // Option
  int id;
  HLD(Graph<int> G):G(G) {
    int N = (int)G.size();
    siz = par = top = dep = in = out = vector<int>(N);
    seg = Segment<int>(N); // Option
    id = 0;
  }
  void dfs(int cur=1, int prev=0) {
    siz[cur] = 1;
    par[cur] = prev;
    dep[cur] = dep[prev] + 1;
    for(int &nxt : G[cur]) {
      if(nxt == prev) continue;
      dfs(nxt, cur);
      siz[cur] += siz[nxt];
      if(siz[nxt] > siz[G[cur][0]]) swap(nxt, G[cur][0]);
    }
  }
  void dfs2(int cur=1, int prev=0) {
    in[cur] = ++id;
    if(cur == 1) top[cur] = 1;
    for(int nxt: G[cur]) {
      if(nxt == prev)continue;
      top[nxt] = (nxt == G[cur][0] ? top[cur] : nxt);
      dfs2(nxt, cur);
    }
    out[cur] = id;
  }
  int lca(int a, int b) {
    while(top[a] != top[b]) {
      if(dep[top[a]] < dep[top[b]]) swap(a, b);
      a = par[top[a]];
    }
    if(in[a] > in[b]) swap(a, b);
    return a;
  }
  void update(int, int);
  int query(int, int);
};
```

2.11 Lca

```
// 1-index, dist (ll), Need Graph Template
struct LCA {
  int N, sz;
  Graph<pair<int, int>> G;
  vector<int> dep;
  vector<ll> dist;
  vector<vector<int>> par;
  LCA(const Graph<pair<int, int>> &_G):G(_G) {
    for(sz = 1; (1 << sz) < N; ++ sz);
    N = G.size();
    dep = vector<int>(N + 1);
    dist = vector<ll>(N + 1);
    par = vector<vector<int>>(sz, vector<int>(N + 1));
    dfs(1, 0);
    for(int j = 1; j < sz; ++j) for(int i = 1; i <= N; ++i)
    par[j][i] = par[j - 1][par[j - 1][i]];
  }
  void dfs(int cur, int prev) {
    dep[cur] = dep[prev] + 1;
    for(const auto &[nxt, w]: G[cur]) {
      if(nxt == prev) continue;
      par[0][nxt] = cur;
      dist[nxt] = dist[cur] + w;
      dfs(nxt, cur);
    }
  }
  int lca(int u, int v) {
    if(dep[u] > dep[v]) swap(u, v);
    for(int i = sz - 1; ~i; --i) if(dep[u] <= dep[par[i][v]])
    v = par[i][v];
    if(u == v) return u;
    for(int i = sz - 1; ~i; --i) if(par[i][u] != par[i][v]) u
    = par[i][u], v = par[i][v];
    return par[0][u];
  }
  ll distance(int u, int v) { return dist[u] + dist[v] - 2 *
  dist[lca(u, v)]; }
  int kth(int u, int v, int k) {
    int l = lca(u, v), dif = dep[u] - dep[l] + 1;
    if(dif < k) k = dep[v] - dep[l] + dif - k, u = v, v = l;
    else --k;
    for(int i = sz - 1; ~i; --i) if(k & (1 << i)) u =
    par[i][u];
    return u;
  }
};
```

2.12 Pst

```
template<typename T>
struct PST{
  struct Node{
    Node *left, *right;
    T data;
    Node(Node *l = nullptr, Node *r = nullptr, T v=0):left(l),
    right(r), data(v) { }
    Node *push(int l, int r, int x, T _data) {
      if(r < x || x < l) return this;
      if(l == r) return new Node(0, 0, this->data + _data);
      int mid = l + (r - l) / 2;
      Node *L = left->push(l, mid, x, _data);
      Node *R = right->push(mid + 1, r, x, _data);
      return new Node(L, R, L->data + R->data);
    }
  };
  Node *roots[100002];
  int siz;

  PST() { setting(); }
  PST(int N) { setting(N); }
  void setting(int N = 2e9 + 10){
    siz = N;
    roots[0] = new Node();
    roots[0]->left = roots[0]->right = roots[0];
  }

  void expand(int p){ roots[p] = roots[p - 1]; }
  void update(int p, int idx, T data, bool _expand=false){
    if(_expand) expand(p);
    roots[p] = roots[p]->push(1, siz, idx, data);
  }

  T query(Node *cur, int l, int r, int s, int e){
    if(s <= l && r <= e)return cur->data;
    if(e <  l || r <  s)return 0;
    int mid = l + (r - l) / 2;
    return query(cur->left, l, mid, s, e) + query(cur->right,
    mid + 1, r, s, e);
  }
  T query(int s, int e, int p){ return query(roots[p], 1, siz,
  s, e); }

  T kth(Node *s, Node *e, int l, int r, int k){
    if(l == r)return l;
    int mid = l + (r - l) / 2;
    T data = e->left->data - s->left->data;
    if(data >= k)return kth(s->left, e->left, l, mid, k);
    return kth(s->right, e->right, mid + 1, r, k - data);
  }
  T kth(int s, int e, int k){ return kth(roots[s], roots[e],
  1, siz, k); }
};
```

2.13 Segment Tree

```
template<typename T>
struct Segment {
  vector<T> tree;
  int siz;

  Segment(int N = 1 << 17) {
    for(siz = 1; siz < N; siz <<= 1);
    tree = vector<T>(siz << 1);
  }
  void build() {
    for(int i = siz - 1; i > 0; --i) {
      tree[i] = tree[i << 1] + tree[i << 1 | 1];
    }
  }
  void update(int idx, T data) {
    tree[idx += siz] = data;
    while(idx >>= 1) tree[idx] = tree[idx << 1] + tree[idx <<
    1 | 1];
  }
  T query(int l, int r) {
    T ret_L = T(), ret_R = T();
    for(l += siz, r += siz; l <= r; l >>= 1, r >>= 1) {
      if(l & 1) ret_L = ret_L + tree[l ++];
      if(~r & 1) ret_R = tree[r --] + ret_R;
    }
    return ret_L + ret_R;
  }
  T& operator[](const int &idx) { return tree[idx + siz]; }
};
```

2.14 Segment Tree With Lazy

```
template<typename T>
struct SegmentLazy {
  vector<T> tree, lazy;
  int siz;

  SegmentLazy(int N = 1 << 17) {
    for(siz = 1; siz < N; siz <<= 1);
    lazy = tree = vector<T>(siz << 1);
  }
  void putItem(int idx, T data) { tree[idx + siz] = data; }
  void build() {
    for(int i = siz - 1; i; --i) tree[i] = merge(tree[i << 1],
    tree[i << 1 | 1]);
  }
  void propagate(int l, int r, int pos) {
    if(!lazy[pos]) return;
    if(l != r) {
      lazy[pos << 1] = merge(lazy[pos << 1], lazy[pos]);
      lazy[pos << 1 | 1] = merge(lazy[pos << 1 | 1],
      lazy[pos]);
    }
    tree[pos] += lazy[pos] * (r - l + 1);
    lazy[pos] = 0;
  }
  void update(int l, int r, int s, int e, int pos, T data) {
    if(s <= l && r <= e) {
      lazy[pos] += data;
      propagate(l, r, pos);
      return;
    }
    propagate(l, r, pos);
    if(e <  l || r <  s) return;
    int mid = (l + r) / 2;
    update(l, mid, s, e, pos << 1, data);
    update(mid + 1, r, s, e, pos << 1 | 1, data);
    tree[pos] = merge(tree[pos << 1], tree[pos << 1 | 1]);
  }
  void update(int s, int e, T data) { update(0, siz - 1, s, e,
  1, data); }
  T query(int l, int r, int s, int e, int pos) {
    propagate(l, r, pos);
    if(s <= l && r <= e) return tree[pos];
    if(e <  l || r <  s) return 0;
    int mid = (l + r) / 2;
    return merge(query(l, mid, s, e, pos << 1), query(mid + 1,
    r, s, e, pos << 1 | 1));
  }
  T query(int s, int e) { return query(0, siz - 1, s, e, 1); }
  T merge(T a, T b) {
    return a + b;
  }
};
```

3 Others

3.1 Fastinput

```
// eof 추가해야함.
#define BUFFERMAX 1 << 19
struct IO {
  char buf[BUFFERMAX];
  char _read() {
    static int idx = BUFFERMAX;
    if(idx == BUFFERMAX) fread(buf, 1, BUFFERMAX, stdin), idx
    = 0;
    return buf[idx++];
  }
  char readChar() {
    char ret = _read();
    while(ret == 10 || ret == 32) ret = _read();
    return ret;
  }
  string readString() {
    string ret = "";
    char now = _read();
    while(now == 10 || now == 32) now = _read();
    while(true) {
      ret += now;
      now = _read();
      if(now == 10 || now == 32) break;
    }
    return ret;
  }
  template<typename T> T readInt() {
    T ret = 0;
    bool minus = false;
    char now = _read();
    while(now == 10 || now == 32) now = _read();
    if(now == '-') minus = true, now = _read();
    while(48 <= now && now <= 57) {
      ret = ret * 10 + now - 48;
      now = _read();
    }
    if(minus) ret *= -1;
    return ret;
  }
  void read(int &x) { x = readInt<int>(); }
  void read(long long &x) { x = readInt<long long>(); }
  void read(char &x) { x = readChar(); }
  void read(string &x) { x = readString(); }
  template<typename Type, typename... Types> void read(Type
  &arg, Types &...args) {  read(arg); read(args...); }
} io;

template<typename T>
IO& operator>> (IO& in, T &x) { in.read(x); return in; }

#define cin io
#define istream IO
```

3.2 Main

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

using namespace std;

#define all(x) (x).begin(),(x).end()
#define rall(x) (x).rbegin(), (x).rend()
#define sz(x) ((int)(x).size())
#define sortall(x) sort(all(x))
#define Unique(x) (x).erase(unique(all(x)), (x).end())
#define compress(x) sortall(x); Unique(x)

typedef bool i1;
typedef char i8;
typedef short i16;
typedef int i32;
typedef long long i64;

typedef unsigned char u8;
typedef unsigned short u16;
typedef unsigned int u32;
typedef unsigned long long u64;

typedef float f16;
typedef double f32;
typedef long double f64;

template<typename T> using Vec = vector<T>;
template<typename T> using Que = queue<T>;
template<typename T> using Dec = deque<T>;

template<int fp=0> struct fastio { fastio() {
ios::sync_with_stdio(false); cin.tie(0); if(fp)cout<<fixed<<'
'<<setprecision(fp); } };

template<typename First, typename Second> inline istream&
operator>>(istream &in, pair<First, Second> &_data) {
in>>_data.first>>_data.second; return in; }
```

```
template<typename First, typename Second> inline ostream&
operator<<(ostream &out, pair<First, Second> &_data) {
out<<_data.first<<' '<<_data.second; return out; }
template<typename First, typename Second, typename Third>
inline istream& operator>>(istream &in, tuple<First, Second,
Third> &_data) {
in>>get<0>(_data)>>get<1>(_data)>>get<2>(_data); return in; }
template<typename First, typename Second, typename Third>
inline ostream& operator<<(ostream &out, tuple<First, Second,
Third> &_data) { out<<get<0>(_data)<<' '<<get<1>(_data)<<'
'<<get<2>(_data); return out; }

template<typename T> auto Vector(const int N, const T& value)
{ return vector(N, value); }
template<typename...Ts> auto Vector(const int N, Ts... args) {
return vector(N, Vector(args...)); }
template<typename InputType> void in(InputType& x) { cin>>x; }
template<typename InputType, typename... InputTypes> void
in(InputType& x, InputTypes& ...y) { cin>>x; in(y...); }
template<typename IterableInputType> void
vin(IterableInputType &V, int skip=0) { for(auto &x: V)
if(--skip < 0) cin >> x; }

template<const int p=0, typename OutputType> void
out(OutputType x) { cout<<x<<' '; }
template<const int p=0, typename OutputType, typename...
OutputTypes> void out(OutputType x, OutputTypes ...y) {
cout<<fixed<<setprecision(p)<<x<<' '; out<p>(y...); }
template<const int p=0, typename IterableOutputType> void
vout(const IterableOutputType &V, int skip=0) { for(auto &x:
V) if(--skip<0) out<p>(x); }

template<i64 modulo=numeric_limits<i64>::max(), typename... T>
i64 Sum(T... x) { return (... + x) % modulo; }
template<i64 modulo=numeric_limits<i64>::max(), typename... T>
i64 Mul(T... x) { return (... * x) % modulo; }

constexpr int dy[] = {-1,1,0,0,-1,-1,1,1,-2,-1,1,2,2,1,-1,-2};
constexpr int dx[] = {0,0,-1,1,-1,1,-1,1,1,2,2,1,-1,-2,-2,-1};

int main() {
  fastio<>();



  return 0;
}
```

4 String

4.1 Aho Corasick

```
struct AhoCorasick {
  struct Trie {
    Trie *nxt[26];
    Trie *fail;
    bool output;

    Trie() {
      for(int i=0;i<26;++i) nxt[i]=nullptr;
      fail=nullptr;
      output=false;
    }
    ~Trie() {
      for(int i=0;i<26;++i) if(nxt[i]) delete nxt[i];
    }
  } *root;
  AhoCorasick() { root = new Trie(); }
  void insert(const string &S) {
    Trie *cur = root;
    int N = (int)S.size();
    for(int i = 0; i < N; ++i) {
      int nxt = S[i] - 'a';
      if(cur->nxt[nxt] == nullptr) cur->nxt[nxt] = new Trie();
      cur = cur->nxt[nxt];
    }
    cur->output=true;
  }
  void build() {
    queue<Trie*> Q;
    root->fail = root;
    Q.push(root);
    while(!Q.empty()) {
      Trie* cur = Q.front(); Q.pop();

      for(int i = 0; i < 26; ++i) {
        Trie *next = cur->nxt[i];
        if(next == nullptr) continue;
        if(cur == root) next->fail = root;
        else {
          Trie *dst = cur->fail;
          while(dst != root && dst->nxt[i] == nullptr) dst =
          dst->fail;
          if(dst->nxt[i]) dst = dst->nxt[i];
          next->fail = dst;
        }
        if(next->fail->output) next->output = true;
        Q.push(next);
      }
    }
  }
  bool find(const string &S) {
    Trie *cur = root;
    int N = (int)S.size();
    for(int i = 0; i < N; ++i) {
      int nxt = S[i] - 'a';
      while(cur != root && cur->nxt[nxt] == nullptr) cur =
      cur->fail;
      if(cur->nxt[nxt]) cur = cur->nxt[nxt];
      if(cur->output) return true;
    }
    return false;
  }
};
```

4.2 Kmp

```
template<typename T> struct KMP {
  vector<int> fail;
  vector<int> failure(const T &Q) {
    fail.resize((int)Q.size() + 1);
    for(int i = 1, j = 0; i < (int)Q.size(); ++i) {
      while(j > 0 && Q[i] != Q[j]) j = fail[j - 1];
      if(Q[i] == Q[j]) fail[i] = ++j;
    }
    return fail;
  }
  vector<int> kmp(const T& P, const T& Q) {
    if(fail.size() == 0) failure(Q);
    vector<int> res;
    for(int i = 0, j = 0; i < (int)P.size(); ++i) {
      while(j > 0 && P[i] != Q[j]) j = fail[j - 1];
      if(P[i] == Q[j]) {
        if(j + 1 == (int)Q.size()) res.push_back(i -
        (int)Q.size() + 1), j = fail[j];
        else ++j;
      }
    }
    return res;
  }
};
```

4.3 Z

```
template<typename T> vector<int> Z(const T &V) {
  int N = (int)V.size();
  vector<int> ret(N); ret[0] = N;
  for(int i = 1, l = 0, r = 0; i < N; ++i) {
    if(i < r) ret[i] = min(r - i - 1, ret[i - l]);
    while(i + ret[i] < N && V[i + ret[i]] == V[ret[i]]) ++
    ret[i];
    if(i + ret[i] > r) r = i + ret[i], l = i;
  }
  return ret;
}
```

5 Geometry

5.1 Ccw

```
  T x, y;
  Point():Point(0, 0) { }
  Point(T _x, T _y):x(_x),y(_y) { }
  Point operator+(Point p) { return Point(x+p.x,y+p.y); }
  Point operator-(Point p) { return Point(x-p.x,y-p.y); }
  T operator*(Point p) { return x*p.y-y*p.x; }
  bool operator==(Point p) { return x == p.x && y == p.y; }
  template<typename OT> void operator=(Point<OT> p) {
  *this=Point(p.x,p.y); }
  void t() { swap(x, y); }
};
template<typename T> inline istream& operator>>(istream &in,
Point<T> &o) { in >> o.x >> o.y; return in; }
template<typename T> inline ostream& operator<<(ostream &out,
Point<T> &o) { out << o.x << ' ' << o.y; return out; }
// -1: 반시계, 0: 평행, 1: 시계
template<typename T> int ccw(Point<T> a, Point<T> b, Point<T>
c) {
  T x = a * b + b * c + c * a;
  return (x > 0) - (x < 0);
}
template<typename T> struct Line {
  Point<T> p1, p2;
  Line():Line(0, 0) {}
  Line(T a, T b):Line(Point<T>(0, 0), Point<T>(a, b)) { }
  Line(Point<T> a, Point<T> b):p1(a),p2(b) {
    if(p1.x > p2.x) swap(p1, p2);
    else if(p1.x == p2.x && p1.y > p2.y) swap(p1, p2);
  }
  T dx() { return p1.x - p2.x; }
  T dy() { return p1.y - p2.y; }
  T ccw() { return p1 * p2; }
  void t() { p1.t(); p2.t(); }
};
// 0: 교점 0개, 1: 교점 1개 (끝점 O), 2: 교점 1개 (끝점 X), 3:
교점 ∞개
// 4: 평행 교점 1개, 5: 평행 교점 ∞개
template<typename T> int intersect(Line<T> l1, Line<T> l2) {
  int ca = ccw(l1.p1, l1.p2, l2.p1), cb = ccw(l1.p1, l1.p2,
  l2.p2);
  int cc = ccw(l2.p1, l2.p2, l1.p1), cd = ccw(l2.p1, l2.p2,
  l1.p2);
  if(ca == 0 && cb == 0 && cc == 0 && cd == 0) {
    if(l1.p1.x == l1.p2.x && l2.p1.x == l2.p2.x && l1.p2.x ==
    l2.p1.x) l1.t(), l2.t();
    int A = l1.p1.x, B = l1.p2.x, C = l2.p1.x, D = l2.p2.x;
    if(A > D || B < C) return 0;
    if(A == D || B == C) return 4;
    return 5;
  }
  if(ca * cb <= 0 && cc * cd <= 0) return (!ca || !cb || !cc
  || !cd) ? 1 : 2;
  return 0;
}
template<typename T, typename AT> pair<int, Point<AT>>
intersection_point(Line<T> l1, Line<T> l2) {
  int chk = intersect(l1, l2);
  if(chk == 0 || chk == 3) return make_pair(chk, Point<AT>());
  if(chk == 1 || chk == 4) {
    Point<AT> ans;
    if(l1.p1 == l2.p1 || l1.p1 == l2.p2) ans = l1.p1;
    else if(l1.p2 == l2.p1 || l1.p2 == l2.p2) ans = l1.p2;
    else if(ccw(l1.p1, l1.p2, l2.p1) == 0) ans = l2.p1;
    else if(ccw(l1.p1, l1.p2, l2.p2) == 0) ans = l2.p2;
    else if(ccw(l2.p1, l2.p2, l1.p1) == 0) ans = l1.p1;
    else if(ccw(l2.p1, l2.p2, l1.p2) == 0) ans = l1.p2;
    return make_pair(1, ans);
  }
  T a = l1.ccw() * l2.dx() - l1.dx() * l2.ccw();
  T b = l1.ccw() * l2.dy() - l1.dy() * l2.ccw();
  T d = l1.dx() * l2.dy() - l1.dy() * l2.dx();
  return make_pair(chk, Point<AT>(1. * a / d, 1. * b / d));
}
```