ICPC Team Notebook

typedef unsigned long long uwu
Sorbonne Université
SWERC 2025

A curated reference of algorithms and data structures

SCIENCES
SORBONNE
UNIVERSITÉ

November 13, 2025

Contents

1 Number Theory

1.1 Euler totient

Number of ints $\leq$ n coprime to n (c.f. sieve for primes).

```
int phi(int n, vi& primes) {
    int res = n;
    for (int p : primes) {
        if (1LL * p * p > n) break;
        if (n % p == 0) {
            while (n % p == 0) n /= p;
            res -= res / p;
        }
    }
    if (n > 1) res -= res / n;
    return res;
}
```

1.2 Fast exponentiation

$O(\log b)$: fast a^b mod p.

```
int modular_exp(int a, int b, int p){
    int res = 1;
    while(b > 0){
        if(b & 1) res = (1LL * a * res) % p;
        b = b >> 1;
        a = (1LL * a * a) % p;
    }
    return res;
}
```

1.3 Sieve of Eratosthenes

$O(n \log\log n)$ sieve[i] = 0 if i prime, spf otherwise for fast fact

```
vi sieve(int n){
    vi sieve(n,0);
    for(int i = 2; i*i < n ; i++)
        if(!sieve[i])
            for(int j = i*i ; j <n ; j += i)
                if(!sieve[j]) sieve[j] = i;
    return sieve;
}
```

2 Data Structures

2.1 Fenwick Tree

Point update, prefix and range sum, $O(\log n)$, $O(n)$ build

```
struct Fenwick {
    int n; vector<ll> t; //using long long, int might overflow.
    Fenwick(vector<ll>& a): n(a.size()), t(n+1,0) {
        for(int i = 1; i <=n; i++) { // Builds tree from array in O(n)
            t[i] += a[i-1];
            int p = i+(i&-i);
            if(p<=n) t[p] += t[i];
        }
    }
    void add(int i,long long v){
        for(;i<=n;i+=i&-i) t[i]+=v;
    }
    long long sum(int i){
        long long r=0;
        for(; i>0 ;i -= i&-i) r += t[i];
        return r;
    }
    long long sum(int l,int r){ return sum(r)-sum(l-1); }
};
```

2.2 Segment Tree(s)

Regular seg tree: point update, range sum query, $O(\log n)$

```
struct SegTree {
    int n; vector<ll> t;
    SegTree(vi &a): n(a.size()), t(4*n) { build(a,1,0,n); }
    void build(vi &a,int v,int l,int r){
        if(r-l == 1){ t[v]=a[l]; return; }
        int m = (l+r)/2;
        build(a, 2*v, l, m); build(a, 2*v+1, m, r);
        t[v] = t[2*v] + t[2*v+1];
    }
    ll sum_aux(int v,int l,int r,int ql,int qr){
        if(qr <= l || r <= ql) return 0;
        if(ql <= l && r <= qr) return t[v];
        int m = (l+r)/2;
        return sum_aux(2*v, l,m,ql,qr) + sum_aux(2*v+1, m,r,ql,qr);
    }
    ll sum(int ql,int qr){ return sum_aux(1,0,n,ql,qr); }
    void upd_aux(int v,int l,int r,int pos,ll val){
        if(r-l==1){ t[v]=val; return; }
        int m = (l+r)/2;
        if(pos<m) upd_aux(2*v,l,m,pos,val);
        else upd_aux(2*v+1,m,r,pos,val);
        t[v]=t[2*v]+t[2*v+1];
    }
    void upd(int pos,ll val){ upd_aux(1,0,n,pos,val); }
};
```

Lazy SegTree: range add, range sum query, O(log n)

```
struct LazySegTree {
    int n; vector<ll> t, lazy;
    LazySegTree(vi &a):n(a.size()),t(4*n),lazy(4*n){build(a,1,0,n);}
    void build(vi &a,int v,int l,int r){
        if(r-l == 1){ t[v]=a[l]; return; }
        int m = (l+r)/2;
        build(a,2*v,l,m); build(a,2*v+1,m,r);
        t[v] = t[2*v]+t[2*v+1];
    }
    void push(int v,int l,int r){
        if(lazy[v]==0) return;
        t[v] += lazy[v]*(r-l);
        if(r-l>1){
            lazy[2*v] += lazy[v];
            lazy[2*v+1] += lazy[v];
        }
        lazy[v]=0;
    }
    void upd_aux(int v,int l,int r,int ql,int qr,ll val){
        push(v,l,r);
        if(qr<=l || r<=ql) return;
        if(ql<=l && r<=qr){
            lazy[v] += val; push(v,l,r);
            return;
        }
        int m = (l+r)/2;
        upd_aux(2*v, l,m,ql,qr,val);
        upd_aux(2*v+1, m,r,ql,qr,val);
        t[v] = t[2*v] + t[2*v+1];
    }
    ll sum_aux(int v,int l,int r,int ql,int qr){
        push(v,l,r);
        if(qr<=l || r<=ql) return 0;
        if(ql<=l && r<=qr) return t[v];
        int m = (l+r)/2;
        return sum_aux(2*v, l,m,ql,qr) + sum_aux(2*v+1, m,r,ql,qr);
    }
    void upd(int ql,int qr,ll val){ upd_aux(1,0,n,ql,qr,val); }
    ll sum(int ql,int qr){ return sum_aux(1,0,n,ql,qr); }
};
```

Iterative SegTree: point update, range sum query, O(log n)

```
struct IterSegTree {
    int n; vector<ll> t;
    IterSegTree(vi &a): n(a.size()), t(2*n) {
        for(int i=0;i<n;i++) t[n+i]=a[i];
        for(int i=n-1;i>0;i--) t[i]=t[i<<1]+t[i<<1|1];
    }
    void upd(int pos,ll val){
        pos+=n; t[pos]=val;
        for(pos>>=1; pos>0; pos>>=1) t[pos]=t[pos<<1]+t[pos<<1|1];
    }
    ll sum(int l,int r){
        ll res=0;
        for(l+=n,r+=n;l<r;l>>=1,r>>=1){
            if(l&1) res+=t[l++];
            if(r&1) res+=t[--r];
        }
        return res;
    }
};
```

2.3 Union find

Complexity: effectively O(1)

```
struct union_find{
    vector<int> rank, parent;
    union_find(int n){
        rank.resize(n, 0); parent.resize(n);
        for (int i = 0; i < n; i++) parent[i] = i;
    }
    int find(int i){
        int root = parent[i];
        if (parent[root] != root) return parent[i] = find(root);
        return root;
    }
    void unite(int x, int y) {
        int xRoot = find(x);
        int yRoot = find(y);
        if (xRoot == yRoot) return;
        if (rank[xRoot] < rank[yRoot]) parent[xRoot] = yRoot;
        else if (rank[yRoot] < rank[xRoot]) parent[yRoot] = xRoot;
        else{
            parent[yRoot] = xRoot;
            rank[xRoot]++;
        }
    }
};
```

3 Graph Algorithms

3.1 2-SAT

Check if n variables satisfy $(x_i \vee x_j) \wedge (\neg x_k \vee x_l) \wedge \ldots$

```
struct TwoSAT {
    int n; SCC g; // c.f. strongly connected components
    TwoSAT(int vars): n(vars), g(2*vars) {}
    int var(int i){ return i; }
    int neg(int i){ return i+n; }
    void add_clause(int a, bool a_val, int b, bool b_val){
        int u = a_val ? var(a) : neg(a);
        int v = b_val ? var(b) : neg(b);
        g.add_edge(u^1, v); // ¬a -> b u XOR 1
        g.add_edge(v^1, u); // ¬b -> a v XOR 1
    }
    bool satisfiable(vi& ans){
        g.kosaraju();
        ans.assign(n,0);
        for(int i=0;i<n;i++){
            if(g.scc[var(i)] == g.scc[neg(i)]) return false;
            ans[i] = g.scc[var(i)] > g.scc[neg(i)];
        }
        return true;
    }
};
```

3.2 Hungarian

Min cost assignment $O(n^3)$. Input cost matrix, output (min cost, assignment)

```
pair<int, vi> hungarian(const vvi &a) {
    int n = a.size(), m = a[0].size();
    vi u(n + 1), v(m + 1), p(m + 1), way(m + 1);
    for (int i = 1; i <= n; i++) {
        p[0] = i;
        vi minv(m + 1, 1e9); // INF constant
        vector<bool> used(m + 1, false);
        int j0 = 0;
        do {
            used[j0] = true;
            int i0 = p[j0], delta = 1e9, j1 = 0;
            for (int j = 1; j <= m; j++) if (!used[j]) {
                int cur = a[i0 - 1][j - 1] - u[i0] - v[j];
                if (cur < minv[j]) minv[j] = cur, way[j] = j0;
                if (minv[j] < delta) delta = minv[j], j1 = j;
            }
            for (int j = 0; j <= m; j++) {
                if (used[j]) u[p[j]] += delta, v[j] -= delta;
                else minv[j] -= delta;
            }
            j0 = j1;
        } while (p[j0] != 0);
        do {
            int j1 = way[j0];
            p[j0] = p[j1];
            j0 = j1;
        } while (j0);
    }
    vi match(n);
    for (int j = 1; j <= m; j++) if (p[j]) match[p[j] - 1] = j - 1;
    return {-v[0], match};
}
```

3.3 Iterative dfs

Iterative depth-first search

```
void dfs_iter(int start, vvi& adj, vi &output, vi& visited) {
    stack<int> st; st.push(start); visited[start] = 1;
    while (!st.empty()) {
        int v = st.top(); bool done = true;
        for (auto u : adj[v]) {
            if (!visited[u]) {
                visited[u] = 1; st.push(u);
                done = false; break;
            }
        }
        if (done) {
            st.pop(); output.push_back(v);
        }
    }
}
```

3.4 Max Bipartite Matching

Hopcroft–Karp $O(E\sqrt{V})$ left [0..nL-1], right [0..nR-1] adj from left to right.

```
int nL, nR;
vector<vi>adj;
vi dist, matchL, matchR;
bool bfs() {
    queue<int> q;
    dist.assign(nL, -1);
    for (int u = 0; u < nL; ++u)
        if (matchL[u] == -1) dist[u] = 0, q.push(u);
    bool found = 0;
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int v : adj[u]) {
            int mu = matchR[v];
            if (mu == -1) found = 1;
            else if (dist[mu] == -1) dist[mu] = dist[u] + 1, q.push(mu);
        }
    }
    return found;
}
bool dfs(int u) {
    for (int v : adj[u]) {
        int mu = matchR[v];
        if (mu == -1 || (dist[mu] == dist[u] + 1 && dfs(mu))) {
            matchL[u] = v; matchR[v] = u;
            return 1;
        }
    }
    dist[u] = -1;
    return 0;
}
int hopcroftKarp() {
    matchL.assign(nL, -1);
    matchR.assign(nR, -1);
    int matching = 0;
    while (bfs())
        for (int u = 0; u < nL; ++u)
            if (matchL[u] == -1 && dfs(u))
                ++matching;
    return matching;
}
```

3.5 Max Flow Min Cut

Dinic's max flow $O(V^2E)$, $O(E\sqrt{V})$ for bipartite/unit. Edges from reachable nodes after flow form a min cut.

```
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
};
struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
vector<vi> adj;
int n, m = 0;
int s, t; // source, target
vi level, ptr;
queue<int> q;
Dinic(int n, int s, int t) : n(n), s(s), t(t) {
    adj.resize(n);level.resize(n);ptr.resize(n);
}
void add_edge(int v, int u, long long cap) {
    edges.emplace_back(v, u, cap);
    edges.emplace_back(u, v, 0);
    adj[v].push_back(m);
    adj[u].push_back(m + 1);
    m += 2;
}
bool bfs() {
    while (!q.empty()) {
        int v = q.front(); q.pop();
        for (int id : adj[v]) {
            if (edges[id].cap == edges[id].flow) continue;
            if (level[edges[id].u] != -1) continue;
            level[edges[id].u] = level[v] + 1;
            q.push(edges[id].u);
        }
    }
    return level[t] != -1;
}
long long dfs(int v, long long pushed) {
    if (pushed == 0) return 0;
    if (v == t) return pushed;
    for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
        int id = adj[v][cid]; int u = edges[id].u;
        if (level[v] + 1 != level[u]) continue;
        long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].flow));
        if (tr == 0) continue;
        edges[id].flow += tr;
        edges[id ^ 1].flow -= tr;
        return tr;
    }
    return 0;
}
long long flow() {
    long long f = 0;
    while (true) {
        fill(level.begin(), level.end(), -1);
        level[s] = 0; q.push(s);
        if (!bfs()) break;
        fill(ptr.begin(), ptr.end(), 0);
        while (long long pushed = dfs(s, flow_inf))
            f += pushed;
    }
    return f;
}
vector<pii> min_cut_edges() {
    vector<bool> vis(n, false);
    queue<int> q;
    q.push(s); vis[s] = true;
    while (!q.empty()) {
        int v = q.front(); q.pop();
        for (int id : adj[v]) {
            auto &e = edges[id];
            if (!vis[e.u] && e.cap > e.flow) {
                vis[e.u] = true;
                q.push(e.u);
            }
        }
    }
    vector<pii> cut;
    for (auto &e : edges) {
        if (vis[e.v] && !vis[e.u] && e.cap > 0) {
            cut.push_back({e.v, e.u});
        }
    }
    return cut;
}
};
```

3.6 Min Cost Flow

Finds min cost to send up to F flow (set F=INF for MCMF) $O(FE \log V)$. Handles neg costs (no neg cycles).

```
struct MCF {
    struct E { int v, rev, cap; ll cost; };
    int n; vector<vector<E>> g;
    MCF(int n): n(n), g(n) {}
    void addEdge(int u,int v,int cap,ll cost){
        g[u].push_back({v,(int)g[v].size(),cap,cost});
        g[v].push_back({u,(int)g[u].size()-1,0,-cost});
    }
    pair<ll,ll> minCostFlow(int s,int t,int maxf){
        ll flow=0,cost=0; vll pot(n),dist(n); vi pv(n),pe(n);
        while(flow<maxf){
            fill(dist.begin(),dist.end(),LLONG_MAX);
            priority_queue<pii,vector<pii>,greater<>> pq;
            dist[s]=0; pq.push({0,s});
            while(!pq.empty()){
                auto [d,u]=pq.top(); pq.pop();
                if(d!=dist[u]) continue;
                for(int i=0;i<(int)g[u].size();i++){
                    E &e=g[u][i]; if(!e.cap) continue;
                    ll nd=d+e.cost+pot[u]-pot[e.v];
                    if(nd<dist[e.v]){
                        dist[e.v]=nd; pv[e.v]=u; pe[e.v]=i;
                        pq.push({nd,e.v});
                    }
                }
            }
            if(dist[t]==LLONG_MAX) break;
            for(int i=0;i<n;i++) if(dist[i]<LLONG_MAX) pot[i]+=dist[i];
            int add=maxf-flow;
            for(int v=t;v!=s;v=pv[v])
                add=min(add,g[pv[v]][pe[v]].cap);
            for(int v=t;v!=s;v=pv[v]){
                E &e=g[pv[v]][pe[v]];
                e.cap-=add; g[v][e.rev].cap+=add;
                cost+=add*e.cost;
            }
            flow+=add;
        }
        return {flow,cost};
    }
};
```

3.7 Minimum Spanning Tree

Kruskal $O(E \log E)$ edges list of {u,v,weight}

```
int kruskal(int V, vvi &edges) {
    sort(edges.begin(), edges.end(),[](auto a, auto b){return a[2]<b[2];});
    union_find uf(V);
    int cost = 0, count = 0;
    for (auto &e : edges) {
        if (uf.find(e[0]) != uf.find(e[1])) {
            uf.unite(e[0], e[1]);
            cost += e[2];
            if (++count == V - 1) break;
        }
    }
    return cost;
}
```

Prim's $O((E+V)\log V)$ adj[u] list of (v,weight) adjacent to u.

```
int prim(int V, vector<vpi> &adj) {
    priority_queue<pii, vector<pii>, greater<pii>> q;
    vector<bool> vis(V, false);
    q.push({0, 0});
    int res = 0;
    while(!q.empty()){
        auto [w,u] = q.top();q.pop();
        if(vis[u])continue;
        res += w;
        vis[u] = true;
        for(auto v : adj[u]){
            if(!vis[v.first]){
                q.push({v.second, v.first});
            }          //weight, vertex
        }
    }
    return res;
}
```

3.8 Shortest Path

Find shortest paths from src (no negative weights). O((V+E)logV)

```
vi dijkstra(const vector<vector<pii>>& adj, int src) {
    vi dist(adj.size(), INT_MAX);
    priority_queue<pii, vector<pii>, greater<pii>> q;
    dist[src] = 0; q.push({0, src});
    while (!q.empty()) {
        auto [d, u] = q.top(); q.pop();
        if (d != dist[u]) continue;
        for (auto [v, w] : adj[u]) {
            if (d+w < dist[v]) {
                dist[v] = d+w;
                q.push({d+w, v});
            }
        }
    }
    return dist;
}
```

$O(VE)$ Shortest Path+neg edges; BFS from nodes with dist $-\infty$ for all neg-cycle reachable.

```
vi bellmanFord(int n, vvi& edges, int src) {
  vector<int> dist(n, INT_MAX);
  dist[src] = 0;
  for (int i = 0; i < n; i++) {
    for (auto edge : edges) {
      int u = edge[0];int v = edge[1];int wt = edge[2];
      if (dist[u] != INT_MAX && dist[u] + wt < dist[v]) {
              if(i == n - 1) return {-1};
              dist[v] = dist[u] + wt;
          }
    }
  }
    return dist;
}
```

All-pairs shortest paths (neg edges ok, no neg cycles) $O(V^3)$ $graph[i][i] = 0$, $graph[i][j] = w$ if edge $i- > j$ else INT_MAX

```
vector<vi> floydWarshall(vector<vi> graph) {
    int V = graph.size();
    auto dist = graph;
    for (int k = 0; k < V; ++k)
        for (int i = 0; i < V; ++i)
            for (int j = 0; j < V; ++j)
                if (dist[i][k] < INT_MAX && dist[k][j] < INT_MAX)
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
    return dist;
}
```

3.9 Strongly Connected Components

Kosaraju SCC O(n+m)

```
struct SCC {
    int n, ordn = 0, scc_cnt = 0;
    vvi adj, adjt; vi vis, ord, scc;
    SCC(int n): n(n), adj(n), adjt(n), vis(n,0), ord(n), scc(n) {}
    void add_edge(int u,int v){ adj[u].push_back(v); adjt[v].push_back(u); }
    void dfs(int u){
        vis[u]=1;
        for(int v:adj[u]) if(!vis[v]) dfs(v);
        ord[ordn++]=u;
    }
    void dfst(int u){
        scc[u]=scc_cnt; vis[u]=0;
        for(int v:adjt[u]) if(vis[v]) dfst(v);
    }
    void kosaraju(){
        for(int i=0;i<n;i++) if(!vis[i]) dfs(i);
        for(int i=ordn-1;i>=0;i--) if(vis[ord[i]]){ scc_cnt++; dfst(ord[i]); }
    }
};
```

3.10 TSP

Traveling Salesman Problem $O(n^2 2^n)$

```
int tsp(int n, vvi& dist) {
    int mask_limit = 1 << n;
    vvi dp(mask_limit, vi(n, INT_MAX));
    dp[1][0] = 0;
    for (int mask = 1; mask < mask_limit; mask++) {
        for (int last = 0; last < n; last++) {
            if (dp[mask][last] == INT_MAX) continue;
            for (int next = 0; next < n; next++) {
                if (mask & (1 << next)) continue;
                int new_mask = mask | (1 << next);
                dp[new_mask][next] = min(dp[new_mask][next],
                                        dp[mask][last] + dist[last][next]);
            }
        }
    }
    int ans = INT_MAX;
    for (int last = 1; last < n; last++) {
        if (dp[mask_limit - 1][last] != INT_MAX && dist[last][0] != INT_MAX) {
            ans = min(ans, dp[mask_limit - 1][last] + dist[last][0]);
        }
    }
    return ans;
}
```

3.11 Toposort

TopoSort via DFS $O(V+E)$.

```
void dfs(int u, vector<vi> &adj, vi &vis, vi &res) {
    vis[u] = 1;
    for (int v : adj[u])
        if (!vis[v])
            dfs(v, adj, vis, res);
    res.push_back(u);
}
vi toposort(vector<vi> &adj) {
    int n = adj.size();
    vi vis(n, 0), res;
    for (int i = 0; i < n; i++)
        if (!vis[i]) dfs(i, adj, vis, res);
    reverse(res.begin(), res.end());
    return res;
}
```

4 Arrays

4.1 Inversions

Count pairs where order flips between arrays. $O(n \log n)$

```
ll inversions(vi& a, vi& b) {
    int n = a.size();
    unordered_map<int,int> pos;
    for (int i = 0; i < n; i++) pos[b[i]] = i + 1;
    Fenwick t(n); // C.f. Fenwick tree
    ll inv = 0;
    for (int i = 0; i < n; i++) {
        inv += i - t.sum(pos[a[i]]);
        t.add(pos[a[i]], 1);
    }
    return inv;
}
```