Team notebook

fightFight

August 19, 2021

Contents

1 DP

1.1 SOS DP

```
// iterate over all the masks
// 3 ^ N
for (int mask = 0; mask < (1<<n); mask++){
```

```
        F[mask] = A[0];
    // iterate over all the subsets of the mask
    for(int i = mask; i > 0; i = (i-1) & mask){
        F[mask] += A[i];
    }
}

// N 2^ N
//iterative version
for(int mask = 0; mask < (1<<N); ++mask){
        dp[mask][-1] = A[mask]; //handle base case separately (leaf states)
        for(int i = 0;i < N; ++i){
                if(mask & (1<<i))
                        dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
                else
                        dp[mask][i] = dp[mask][i-1];
        }
        F[mask] = dp[mask][N-1];
}
//memory optimized, super easy to code.
for(int i = 0; i<(1<<N); ++i)
        F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
        if(mask & (1<<i))
                F[mask] += F[mask^(1<<i)];
}
```

2 Data Structures

2.1 Lazy Propagation

```
void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = 0;
    }
}
```

```
void update(int v, int tl, int tr, int l, int r, int add) {
    if (l > r)
        return;
    if (l == tl && r == tr) {
        t[v] += add;
    } else {
        int tm = (tl + tr) / 2;
        update(v*2, tl, tm, l, min(r, tm), add);
        update(v*2+1, tm+1, tr, max(l, tm+1), r, add);
    }
}

int get(int v, int tl, int tr, int pos) {
    if (tl == tr)
        return t[v];
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        return t[v] + get(v*2, tl, tm, pos);
    else
        return t[v] + get(v*2+1, tm+1, tr, pos);
}
```

2.2 Range Update Segment Tree

```
template <typename T>
class RangeUpdateTree{
private:
    int n;
    vector<T> tree;
    T identity;

    T merge(T a, T b){
        return a + b;
    }

    T __query(int q){
        T accum = identity;
        for(int i=n+q; i > 0; i /= 2){
            accum = merge(tree[i], accum);
        }
        return accum;
    }
```

```
    void __update(int node, int seg_l, int seg_r, int q_l, int q_r, T
        val){
        if(seg_l > q_r || seg_r < q_l) return;
        if(seg_l >= q_l && seg_r <= q_r) {
            tree[node] += val;
            return;
        }
        int mid = seg_l + (seg_r-seg_l)/2;
        __update(2*node, seg_l, mid, q_l, q_r, val);
        __update(2*node+1, mid+1, seg_r, q_l, q_r, val);
    }

    void build(vector<T> &arr){
        n = 1 << (32 - __builtin_clz ((int)arr.size() - 1));
        tree.resize(2*n);
        for(int i=0; i<arr.size(); i++)
            tree[n+i] = arr[i];
    }

public:
    RangeUpdateTree(vector<T> &arr, T id){
        identity = id;
        build(arr);
    }

    T query(int pos){
        return __query(pos);
    }

    void update(int l, int r, T value){
        __update(1, 0, n-1, l, r, value);
    }
};
```

2.3 Segment Tree

```
template <typename T>
class Segtree{
private:
    int n;
    vector<T> tree;
    T identity;

    T merge(T a, T b){
        return a + b;
    }

    T __query(int node, int seg_l, int seg_r, int q_l, int q_r){
        if(seg_l >= q_l && seg_r <= q_r) return tree[node];
        if(seg_l > q_r || seg_r < q_l) return identity;
        int mid = seg_l + (seg_r-seg_l)/2;
        return merge(__query(2*node, seg_l, mid, q_l, q_r),
            __query(2*node+1, mid+1, seg_r, q_l, q_r));
    }

    void build(vector<T> &arr){
        int sz = (int) arr.size();
        n = 1 << (32 - __builtin_clz (sz - 1));
        tree.resize(2*n);
        for(int i=0; i<sz; i++) tree[n+i] = arr[i];
        for(int i=sz; i<n; i++) tree[n+i] = identity;

        for(int i=n-1; i>=1; i--)
            tree[i] = merge(tree[2*i], tree[2*i+1]);
    }

public:
    Segtree(vector<T> &arr, T id){
        identity = id;
        build(arr);
    }

    T query(int l, int r){
        return __query(1, 0, n-1, l, r);
    }

    void update(int node, T value){
        tree[n+node] = value;
        for(int i=(n+node)/2; i>=1; i/=2)
            tree[i] = merge(tree[2*i], tree[2*i+1]);
    }
};
```

2.4 Tries (Array Implementation)

```
class Trie{
```

```
private:
    int arr[int(1e6)][26];
    int root;
    int lastocc;
public:
    Trie(){
        root = 0;
        lastocc = 0;
        memset(arr, 0, sizeof(int)*1e6*26);
    }

    void insert(const string &x){
        int curptr = root;

        for(auto ch:x){
            if(arr[curptr][ch-'a']==0)
                arr[curptr][ch-'a'] = ++lastocc;
            curptr = arr[curptr][ch-'a'];
        }
    }

    int search(const string &x){
        int curptr = root;
        for(auto ch:x){
            if(arr[curptr][ch-'a']==0)
                return 0;
            else
                curptr = arr[curptr][ch-'a'];
        }
        return 1;
    }
};
```

2.5 Tries (Pointer Implementation)

```
class Trie{

private:
    struct Node{
        int val;
        Node *arr[26];
    } *root;
public:
    Trie(){
        root = new Node();
    }

    void insert(const string &x){
        Node *curptr = root;
        for(auto ch:x){
            if(curptr->arr[ch-'a']==NULL)
                curptr->arr[ch-'a'] = new Node();
            curptr = curptr->arr[ch-'a'];
        }
    }

    int search(const string &x){
        Node *curptr = root;
        for(auto ch:x){
            if(curptr->arr[ch-'a']==NULL)
                return 0;
            else
                curptr = curptr->arr[ch-'a'];
        }
        return 1;
    }
};
```

3 Graphs

3.1 2 SAT

```
int n;
vector<vector<int>> g, gt;
vector<bool> used;
vector<int> order, comp;
vector<bool> assignment;

void dfs1(int v) {
    used[v] = true;
    for (int u : g[v]) {
        if (!used[u])
            dfs1(u);
```

```
    }
    order.push_back(v);
}

void dfs2(int v, int cl) {
    comp[v] = cl;
    for (int u : gt[v]) {
        if (comp[u] == -1)
            dfs2(u, cl);
    }
}

bool solve_2SAT() {
    order.clear();
    used.assign(n, false);
    for (int i = 0; i < n; ++i) {
        if (!used[i])
            dfs1(i);
    }

    comp.assign(n, -1);
    for (int i = 0, j = 0; i < n; ++i) {
        int v = order[n - i - 1];
        if (comp[v] == -1)
            dfs2(v, j++);
    }

    assignment.assign(n / 2, false);
    for (int i = 0; i < n; i += 2) {
        if (comp[i] == comp[i + 1])
            return false;
        assignment[i / 2] = comp[i] > comp[i + 1];
    }
    return true;
}
```

3.2 Articulation Points

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}
```

3.3 Bellman Ford

```
void solve()
{
    vector<int> d (n, INF);
    d[v] = 0;
    vector<int> p (n, -1);

    for (;;)
    {
```

```
        bool any = false;
        for (int j = 0; j < m; ++j)
            if (d[e[j].a] < INF)
                if (d[e[j].b] > d[e[j].a] + e[j].cost)
                {
                    d[e[j].b] = d[e[j].a] + e[j].cost;
                    p[e[j].b] = e[j].a;
                    any = true;
                }
        if (!any) break;
    }

    if (d[t] == INF)
        cout << "No path from " << v << " to " << t << ".";
    else
    {
        vector<int> path;
        for (int cur = t; cur != -1; cur = p[cur])
            path.push_back (cur);
        reverse (path.begin(), path.end());

        cout << "Path from " << v << " to " << t << ": ";
        for (size_t i=0; i<path.size(); ++i)
            cout << path[i] << ' ';
    }
}
```

3.4 Bridges

```
void tarjans_b(int node, vvi &adj, vi &disc, vi &low, int p, DSU &dsu){
    static int timer = 1;
    if(disc[node]) return;
    low[node] = disc[node] = timer++;

    for(auto &x:adj[node]){
        if(x==p) continue;
        tarjans_b(x, adj, disc, low, node, dsu);
        low[node] = min(low[node], low[x]);
        if(low[x] <= disc[node])
            dsu.unify(node, x);
    }
}
```

```
void bridgeCutTree(vvi &adj, vvi &tree, DSU &dsu){
    int n = (int) adj.size();
    vi disc(n);
    vi low(n);
    dsu.make(n);
    tree.resize(n);

    tarjans_b(0, adj, disc, low, -1, dsu);
    for(int i=0; i<n; i++){
        for(auto j:adj[i]){
            int ip = dsu[i];
            int jp = dsu[j];
            if(ip==jp) continue;
            tree[ip].pb(jp);
        }
    }
}
```

3.5 Condensation Graph

```
vector<int> roots(n, 0);
vector<int> root_nodes;
vector<vector<int>> adj_scc(n);

for (auto v : order)
    if (!used[v]) {
        dfs2(v);

        int root = component.front();
        for (auto u : component) roots[u] = root;
        root_nodes.push_back(root);

        component.clear();
    }


for (int v = 0; v < n; v++)
    for (auto u : adj[v]) {
        int root_v = roots[v],
            root_u = roots[u];

        if (root_u != root_v)
            adj_scc[root_v].push_back(root_u);
```

```
    }
```

3.6 DSU

```
class DSU{
private:
    vector<int> dsu;
    vector<int> rank;

    void __makedsu(int n){
        dsu.resize(n);
        rank.resize(n);
        for(int i=0; i<n; i++) dsu[i] = i;
    }

public:
    DSU(){}
    DSU(int n){ __makedsu(n); }
    void make(int n){ __makedsu(n); }

    int parent(int i){
        if(dsu[i]==i) return i;
        else return dsu[i] = parent(dsu[i]);
    }

    int operator[](int i){
        return parent(i);
    }

    void unify(int a, int b){
        a = parent(a);
        b = parent(b);
        if(rank[a] < rank[b])
            swap(a, b);
        dsu[b] = a;
        if(a!=b && rank[a] == rank[b])
            rank[a]++;
    }
};
```

3.7 Floyd Warshall

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

3.8 Kosaraju SCC

```
vector<vector<int>> adj, adj_rev;
vector<bool> used;
vector<int> order, component;

void dfs1(int v) {
    used[v] = true;

    for (auto u : adj[v])
        if (!used[u])
            dfs1(u);

    order.push_back(v);
}

void dfs2(int v) {
    used[v] = true;
    component.push_back(v);

    for (auto u : adj_rev[v])
        if (!used[u])
            dfs2(u);
}

int main() {
    int n;
    // ... read n ...

    for (;;) {
        int a, b;
        // ... read next directed edge (a,b) ...
        adj[a].push_back(b);
```

```
        adj_rev[b].push_back(a);
    }

    used.assign(n, false);

    for (int i = 0; i < n; i++)
        if (!used[i])
            dfs1(i);

    used.assign(n, false);
    reverse(order.begin(), order.end());

    for (auto v : order)
        if (!used[v]) {
            dfs2 (v);

            // ... processing next component ...

            component.clear();
        }
}
```

3.9 LCA

```
class LCA{
private:
    int n;
    vvi lift;
    vi dep;

    void mark_parents(int node, vvi &adj, int p){
        for(auto x:adj[node]){
            if(x==p) continue;
            dep[x] = dep[node]+1;
            mark_parents(x, adj, node);
            lift[x][0] = node;
        }
    }

    void precomp_LCA(vvi &tree, int root){
        int n = (int) tree.size();
        lift.assign(n, vi(32, root));
        dep.assign(n, 0);
        dep[root] = 0;
        mark_parents(root, tree, -1);
        for(int i=1; i<32; i++)
            for(int j=0; j<n; j++)
                lift[j][i] = lift[lift[j][i-1]][i-1];
    }

public:

    LCA() = default;

    LCA(vvi &tree, int root){
        precomp_LCA(tree, root);
    }

    void make(vvi &tree, int root) {
        precomp_LCA(tree, root);
    }

    int getLCA(int a, int b){
        if(dep[a]<dep[b]) swap(a, b);
        int diff = dep[a] - dep[b];

        for(int i=31; i>=0; i--)
            if(diff & (1<<i))
                a = lift[a][i];

        if(a==b) return a;

        for(int i=31; i>=0; i--)
            if(lift[a][i] != lift[b][i])
                a = lift[a][i], b = lift[b][i];

        return lift[a][0];
    }

    int getAncestor(int a, int k){
        for(int i=0; i<32; i++)
            if(k & (1<<i))
                a = lift[a][i];
        return a;
    }

    int depth(int a) { return dep[a]; }
```

```
};
```

3.10 MCMF

```
// After running max-flow, the left side of a min-cut from $s$ to $t$ is
    given by all vertices reachable from $s$, only traversing edges with
    positive residual capacity
const ll INF = numeric_limits<ll>::max() / 4;
typedef vector<ll> VL;

struct MCMF {
	int N;
	vector<vi> ed, red;
	vector<VL> cap, flow, cost;
	vi seen;
	VL dist, pi;
	vector<pii> par;

	MCMF(int N) :
		N(N), ed(N), red(N), cap(N, VL(N)), flow(cap), cost(cap),
		seen(N), dist(N), pi(N), par(N) {}

	void addEdge(int from, int to, ll cap, ll cost) {
		this->cap[from][to] = cap;
		this->cost[from][to] = cost;
		ed[from].push_back(to);
		red[to].push_back(from);
	}

	void path(int s) {
		fill(all(seen), 0);
		fill(all(dist), INF);
		dist[s] = 0; ll di;

		__gnu_pbds::priority_queue<pair<ll, int>> q;
		vector<decltype(q)::point_iterator> its(N);
		q.push({0, s});

		auto relax = [&](int i, ll cap, ll cost, int dir) {
			ll val = di - pi[i] + cost;
			if (cap && val < dist[i]) {
				dist[i] = val;
				par[i] = {s, dir};
				if (its[i] == q.end()) its[i] =
				    q.push({-dist[i], i});
				else q.modify(its[i], {-dist[i], i});
			}
		};

		while (!q.empty()) {
			s = q.top().second; q.pop();
			seen[s] = 1; di = dist[s] + pi[s];
			for (int i : ed[s]) if (!seen[i])
				relax(i, cap[s][i] - flow[s][i], cost[s][i],
				    1);
			for (int i : red[s]) if (!seen[i])
				relax(i, flow[i][s], -cost[i][s], 0);
		}
		rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
	}

	pair<ll, ll> maxflow(int s, int t) {
		ll totflow = 0, totcost = 0;
		while (path(s), seen[t]) {
			ll fl = INF;
			for (int p,r,x = t; tie(p,r) = par[x], x != s; x =
			    p)
				fl = min(fl, r ? cap[p][x] - flow[p][x] :
				    flow[x][p]);
			totflow += fl;
			for (int p,r,x = t; tie(p,r) = par[x], x != s; x =
			    p)
				if (r) flow[p][x] += fl;
				else flow[x][p] -= fl;
		}
		rep(i,0,N) rep(j,0,N) totcost += cost[i][j] * flow[i][j];
		return {totflow, totcost};
	}

	// If some costs can be negative, call this before maxflow:
	void setpi(int s) { // (otherwise, leave this out)
		fill(all(pi), INF); pi[s] = 0;
		int it = N, ch = 1; ll v;
		while (ch-- && it--)
			rep(i,0,N) if (pi[i] != INF)
				for (int to : ed[i]) if (cap[i][to])
					if ((v = pi[i] + cost[i][to]) <
					    pi[to])
```

```
                                    pi[to] = v, ch = 1;
            assert(it >= 0); // negative cost cycle
        }
};
```

3.11 MST Kruskals

```
vector<int> parent, rank;

void make_set(int v) {
    parent[v] = v;
    rank[v] = 0;
}

int find_set(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}

void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (rank[a] < rank[b])
            swap(a, b);
        parent[b] = a;
        if (rank[a] == rank[b])
            rank[a]++;
    }
}

struct Edge {
    int u, v, weight;
    bool operator<(Edge const& other) {
        return weight < other.weight;
    }
};

int n;
vector<Edge> edges;

int cost = 0;
vector<Edge> result;
parent.resize(n);
rank.resize(n);
for (int i = 0; i < n; i++)
    make_set(i);

sort(edges.begin(), edges.end());

for (Edge e : edges) {
    if (find_set(e.u) != find_set(e.v)) {
        cost += e.weight;
        result.push_back(e);
        union_sets(e.u, e.v);
    }
}
```

3.12 MST Prims

```
const int INF = 1000000000;

struct Edge {
    int w = INF, to = -1;
    bool operator<(Edge const& other) const {
        return make_pair(w, to) < make_pair(other.w, other.to);
    }
};

int n;
vector<vector<Edge>> adj;

void prim() {
    int total_weight = 0;
    vector<Edge> min_e(n);
    min_e[0].w = 0;
    set<Edge> q;
    q.insert({0, 0});
    vector<bool> selected(n, false);
    for (int i = 0; i < n; ++i) {
        if (q.empty()) {
            cout << "No MST!" << endl;
            exit(0);
        }
```

```
        int v = q.begin()->to;
        selected[v] = true;
        total_weight += q.begin()->w;
        q.erase(q.begin());

        if (min_e[v].to != -1)
            cout << v << " " << min_e[v].to << endl;

        for (Edge e : adj[v]) {
            if (!selected[e.to] && e.w < min_e[e.to].w) {
                q.erase({min_e[e.to].w, e.to});
                min_e[e.to] = {e.w, v};
                q.insert({e.w, e.to});
            }
        }
    }

    cout << total_weight << endl;
}
```

3.13 Maximum Flow Dinics

```
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<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

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

    void add_edge(int v, int u, 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 < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    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] || edges[id].cap - edges[id].flow
                < 1)
                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;
    }
};
```

4 Linear Algebra

4.1 Gauss Determinant

```
const double EPS = 1E-9;
int n;
vector < vector<double> > a (n, vector<double> (n));

double det = 1;
for (int i=0; i<n; ++i) {
    int k = i;
    for (int j=i+1; j<n; ++j)
        if (abs (a[j][i]) > abs (a[k][i]))
            k = j;
    if (abs (a[k][i]) < EPS) {
        det = 0;
        break;
    }
    swap (a[i], a[k]);
    if (i != k)
        det = -det;
    det *= a[i][i];
    for (int j=i+1; j<n; ++j)
        a[i][j] /= a[i][i];
    for (int j=0; j<n; ++j)
        if (j != i && abs (a[j][i]) > EPS)
            for (int k=i+1; k<n; ++k)
                a[j][k] -= a[i][k] * a[j][i];
}

cout << det;
```

4.2 Gauss Linear Equations

```
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be infinity or a big
    number

int gauss (vector < vector<double> > a, vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
```

```
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

4.3 Rank of Matrix

```
const double EPS = 1E-9;

int compute_rank(vector<vector<double>> A) {
    int n = A.size();
    int m = A[0].size();

    int rank = 0;
    vector<bool> row_selected(n, false);
    for (int i = 0; i < m; ++i) {
        int j;
        for (j = 0; j < n; ++j) {
            if (!row_selected[j] && abs(A[j][i]) > EPS)
                break;
        }

        if (j != n) {
            ++rank;
            row_selected[j] = true;
            for (int p = i + 1; p < m; ++p)
                A[j][p] /= A[j][i];
            for (int k = 0; k < n; ++k) {
                if (k != j && abs(A[k][i]) > EPS) {
                    for (int p = i + 1; p < m; ++p)
                        A[k][p] -= A[j][p] * A[k][i];
                }
            }
        }
    }
    return rank;
}
```

5 Number Theory

5.1 Chinese Remainder Theorem

```
for (int i = 0; i < k; ++i) {
    x[i] = a[i];
    for (int j = 0; j < i; ++j) {
        x[i] = r[j][i] * (x[i] - x[j]);

        x[i] = x[i] % p[i];
        if (x[i] < 0)
            x[i] += p[i];
    }
}
```

5.2 Counting Divisors

```
// C++ program to count distinct divisors
// of a given number n
#include <bits/stdc++.h>
using namespace std;

void SieveOfEratosthenes(int n, bool prime[],
                         bool primesquare[], int a[])
{
    // Create a boolean array "prime[0..n]" and
    // initialize all entries it as true. A value
    // in prime[i] will finally be false if i is
    // Not a prime, else true.
    for (int i = 2; i <= n; i++)
        prime[i] = true;

    // Create a boolean array "primesquare[0..n*n+1]"
    // and initialize all entries it as false. A value
    // in squareprime[i] will finally be true if i is
    // square of prime, else false.
```

```
    for (int i = 0; i <= (n * n + 1); i++)
        primesquare[i] = false;

    // 1 is not a prime number
    prime[1] = false;

    for (int p = 2; p * p <= n; p++) {
        // If prime[p] is not changed, then
        // it is a prime
        if (prime[p] == true) {
            // Update all multiples of p
            for (int i = p * 2; i <= n; i += p)
                prime[i] = false;
        }
    }

    int j = 0;
    for (int p = 2; p <= n; p++) {
        if (prime[p]) {
            // Storing primes in an array
            a[j] = p;

            // Update value in primesquare[p*p],
            // if p is prime.
            primesquare[p * p] = true;
            j++;
        }
    }
}

// Function to count divisors
int countDivisors(int n)
{
    // If number is 1, then it will have only 1
    // as a factor. So, total factors will be 1.
    if (n == 1)
        return 1;

    bool prime[n + 1], primesquare[n * n + 1];

    int a[n]; // for storing primes upto n

    // Calling SieveOfEratosthenes to store prime
    // factors of n and to store square of prime
    // factors of n
    SieveOfEratosthenes(n, prime, primesquare, a);

    // ans will contain total number of distinct
    // divisors
    int ans = 1;

    // Loop for counting factors of n
    for (int i = 0;; i++) {
        // a[i] is not less than cube root n
        if (a[i] * a[i] * a[i] > n)
            break;

        // Calculating power of a[i] in n.
        int cnt = 1; // cnt is power of prime a[i] in n.
        while (n % a[i] == 0) // if a[i] is a factor of n
        {
            n = n / a[i];
            cnt = cnt + 1; // incrementing power
        }

        // Calculating the number of divisors
        // If n = a^p * b^q then total divisors of n
        // are (p+1)*(q+1)
        ans = ans * cnt;
    }

    // if a[i] is greater than cube root of n

    // First case
    if (prime[n])
        ans = ans * 2;

    // Second case
    else if (primesquare[n])
        ans = ans * 3;

    // Third case
    else if (n != 1)
        ans = ans * 4;

    return ans; // Total divisors
}

// Driver Program
int main()
```

```
{
    cout << "Total distinct divisors of 100 are : "
         << countDivisors(100) << endl;
    return 0;
}
```

5.3 Eulers Totient Function

```
// Totient of n
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}
// Totient of 1 to n
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    phi[0] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

5.4 Extended Euclidean

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}
```

5.5 Fraction

```
class Fraction {
private:
ll numerator, denominator;
ll binpow(ll a, ll b) {
        a %= modulo;
        b %= modulo - 1;
        ll res = 1;
        while (b > 0) {
                if (b & 1)
                        res = res * a % modulo;
                a = a * a % modulo;
                b >>= 1;
        }
        return res;
}
public:
static ll modulo;
Fraction(ll n = 0, ll d = 1) { numerator = n; denominator = d; }

void reduce() {
        ll x = __gcd(numerator, denominator);
        numerator /= x;
        denominator /= x;

        if (modulo == -1) return;

        numerator %= modulo;
        denominator %= modulo;
}
```

```
Fraction operator + (Fraction const f) {
        Fraction ans;
        ans.numerator = numerator * f.denominator + denominator *
            f.numerator;
        ans.denominator = denominator * f.denominator;
        ans.reduce();
        return ans;
}

Fraction operator * (Fraction const f) {
        Fraction ans;
        ans.numerator = numerator * f.numerator;
        ans.denominator = denominator * f.denominator;
        ans.reduce();
        return ans;
}
Fraction operator / (Fraction const f) {
        Fraction ans;
        ans.numerator = numerator * f.denominator;
        ans.denominator = denominator * f.numerator;
        ans.reduce();
        return ans;
}

ll inverseNotation() {
        ll inverse = binpow(denominator, modulo - 2);
        return numerator * inverse % modulo;
}
operator float() const { return float(numerator) / float(denominator); }
operator double() const { return double(numerator) / double(denominator);
    }
};
// Make modulo -1 if you don't want it to apply modulo
ll Fraction::modulo = 1000000007;
```

5.6 Garners Algorithm

```
final int SZ = 100;
int pr[] = new int[SZ];
int r[][] = new int[SZ][SZ];

void init() {
    for (int x = 1000 * 1000 * 1000, i = 0; i < SZ; ++x)
        if (BigInteger.valueOf(x).isProbablePrime(100))
            pr[i++] = x;

    for (int i = 0; i < SZ; ++i)
        for (int j = i + 1; j < SZ; ++j)
            r[i][j] =
                BigInteger.valueOf(pr[i]).modInverse(BigInteger.valueOf(pr[j])).in
}

class Number {
    int a[] = new int[SZ];

    public Number() {
    }

    public Number(int n) {
        for (int i = 0; i < SZ; ++i)
            a[i] = n % pr[i];
    }

    public Number(BigInteger n) {
        for (int i = 0; i < SZ; ++i)
            a[i] = n.mod(BigInteger.valueOf(pr[i])).intValue();
    }

    public Number add(Number n) {
        Number result = new Number();
        for (int i = 0; i < SZ; ++i)
            result.a[i] = (a[i] + n.a[i]) % pr[i];
        return result;
    }

    public Number subtract(Number n) {
        Number result = new Number();
        for (int i = 0; i < SZ; ++i)
            result.a[i] = (a[i] - n.a[i] + pr[i]) % pr[i];
        return result;
    }

    public Number multiply(Number n) {
        Number result = new Number();
        for (int i = 0; i < SZ; ++i)
            result.a[i] = (int)((a[i] * 1l * n.a[i]) % pr[i]);
        return result;
    }
```

```
    public BigInteger bigIntegerValue(boolean can_be_negative) {
        BigInteger result = BigInteger.ZERO, mult = BigInteger.ONE;
        int x[] = new int[SZ];
        for (int i = 0; i < SZ; ++i) {
            x[i] = a[i];
            for (int j = 0; j < i; ++j) {
                long cur = (x[i] - x[j]) * 1l * r[j][i];
                x[i] = (int)((cur % pr[i] + pr[i]) % pr[i]);
            }
            result = result.add(mult.multiply(BigInteger.valueOf(x[i])));
            mult = mult.multiply(BigInteger.valueOf(pr[i]));
        }

        if (can_be_negative)
            if (result.compareTo(mult.shiftRight(1)) >= 0)
                result = result.subtract(mult);

        return result;
    }
}
```

5.7 Linear Diophantine Equations

```
// Is there a solution
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }
    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

// Number of solutions / solutions in a given interval
void shift_solution(int & x, int & y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}

int find_all_solutions(int a, int b, int c, int minx, int maxx, int miny,
     int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
```

```
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

5.8 Linear Sieve

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

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

5.9 Matrix Exponentiation

```
class Matrix {
private:

public:
vvi matrix;
static ll modulo;
Matrix(int n, int identity = 0) {
        matrix = vvi(n, vi(n,0));

        if (identity) {
                for (int i = 0; i < n; i++) {
                        matrix[i][i] = 1;
                }
        }

}
Matrix(vvi m) { matrix = m; }

int size() {
        return matrix.size();
}
Matrix operator * (Matrix const m) {
        int sz = size();
        Matrix product(sz);
        for (int i = 0; i < sz; i++) {
                for (int j = 0; j < sz; j++) {
                        for (int k = 0; k < sz; k++) {
                                if (modulo == -1) {
                                        product.matrix[i][j] += matrix[i][k]
                                            * m.matrix[k][j];
                                }
                                else {
                                        product.matrix[i][j] += matrix[i][k]
                                            * m.matrix[k][j] % modulo;
                                        product.matrix[i][j] %= modulo;
                                }
                        }
                }
        }
        return product;
}

Matrix operator + (Matrix const m) {
        int sz = size();
        Matrix sum(sz);
        for (int i = 0; i < sz; i++) {
                for (int j = 0; j < sz; j++) {
                        sum.matrix[i][j] = matrix[i][j] + m.matrix[i][j];
                        if (modulo != -1) sum.matrix[i][j] %= modulo;
                }
        }
        return sum;
}

Matrix operator - (Matrix const m) {
```

```
        int sz = size();
        Matrix sum(sz);
        for (int i = 0; i < sz; i++) {
                for (int j = 0; j < sz; j++) {
                        sum.matrix[i][j] = matrix[i][j] - m.matrix[i][j];
                        if (modulo != -1) sum.matrix[i][j] %= modulo;
                }
        }
        return sum;
}

Matrix binpow(Matrix a, ll b) {
        b %= modulo - 1;
        Matrix res(a.size(), 1);
        while (b > 0) {
                if (b & 1)
                        res = res * a;
                a = a * a;
                b >>= 1;
        }
        return res;
}
};
// Make modulo -1 if you don't want it to apply modulo
ll Matrix::modulo = 1000000007;
```

5.10 Totient Function Theory

Application in Euler's theorem

The most famous and important property of Euler's totient function is expressed in **Euler's theorem**:

$$a^{\phi(m)} \equiv 1 \pmod m$$

if a and m are relatively prime.

In the particular case when m is prime, Euler's theorem turns into **Fermat's little theorem**:

$$a^{m-1} \equiv 1 \pmod m$$

Euler's theorem and Euler's totient function occur quite often in practical applications, for example both are used to compute the modular multiplicative inverse

As immediate consequence we also get the equivalence:

$$a^n \equiv a^{n \bmod \phi(m)} \pmod m$$

This allows computing $x^n \bmod m$ for very big n, especially if n is the result of another computation, as it allows to compute n under a modulo.

Generalization

There is a less known version of the last equivalence, that allows computing $x^n \bmod m$ efficiently for not coprime x and m.
For arbitrary x, m and $n \geq \log_2 m$:

$$x^n \equiv x^{\phi(m)+[n \bmod \phi(m)]} \mod m$$

Proof:

Let $p_1, \ldots, p_t$ be common prime divisors of x and m, and k_i their exponents in m.
With those we define $a = p_1^{k_1} \ldots p_t^{k_t}$, which makes $\frac{m}{a}$ coprime to x.
And let k be the smallest number such that a divides x^k.
Assuming $n \geq k$, we can write:

$$x^n \bmod m = \frac{x^k}{a} a x^{n-k} \bmod m$$

$$= \frac{x^k}{a} \left(a x^{n-k} \bmod m \right) \bmod m$$

$$= \frac{x^k}{a} \left(a x^{n-k} \bmod a \frac{m}{a} \right) \bmod m$$

$$= \frac{x^k}{a} a \left(x^{n-k} \bmod \frac{m}{a} \right) \bmod m$$

$$= x^k \left(x^{n-k} \bmod \frac{m}{a} \right) \bmod m$$

The equivalence between the third and forth line follows from the fact that $ab \bmod ac = a(b \bmod c)$. Indeed if $b = cd + r$ with $r < c$, then $ab = acd + ar$ with $ar < ac$.

Since x and $\frac{m}{a}$ are coprime, we can apply Euler's theorem and get the efficient (since k is very small; in fact $k \le \log_2 m$) formula:

$$x^n \bmod m = x^k \left(x^{n-k \bmod \phi(\frac{m}{a})} \bmod \frac{m}{a} \right) \bmod m.$$

This formula is difficult to apply, but we can use it to analyze the behavior of $x^n \bmod m$. We can see that the sequence of powers $(x^1 \bmod m, x^2 \bmod m, x^3 \bmod m, \ldots)$ enters a cycle of length $\phi\left(\frac{m}{a}\right)$ after the first k (or less) elements.
$\phi\left(\frac{m}{a}\right)$ divides $\phi(m)$ (because a and $\frac{m}{a}$ are coprime we have $\phi(a) \cdot \phi\left(\frac{m}{a}\right) = \phi(m)$), therefore we can also say that the period has length $\phi(m)$.
And since $\phi(m) \ge \log_2 m \ge k$, we can conclude the desired, much simpler, formula:

$$x^n \equiv x^{\phi(m)} x^{(n-\phi(m)) \bmod \phi(m)} \bmod m \equiv x^{\phi(m)+[n \bmod \phi(m)]} \quad \bmod m.$$

5.11 nCr in O(1)

```
// C++ program to answer queries
// of nCr in O(1) time.
#include <bits/stdc++.h>
#define ll long long
const int N = 1000001;
using namespace std;

// array to store inverse of 1 to N
ll factorialNumInverse[N + 1];

// array to precompute inverse of 1! to N!
ll naturalNumInverse[N + 1];

// array to store factorial of first N numbers
ll fact[N + 1];

// Function to precompute inverse of numbers
void InverseofNumber(ll p)
{
    naturalNumInverse[0] = naturalNumInverse[1] = 1;
    for (int i = 2; i <= N; i++)
        naturalNumInverse[i] = naturalNumInverse[p % i] * (p - p / i) % p;
}
// Function to precompute inverse of factorials
void InverseofFactorial(ll p)
{
    factorialNumInverse[0] = factorialNumInverse[1] = 1;

    // precompute inverse of natural numbers
    for (int i = 2; i <= N; i++)
        factorialNumInverse[i] = (naturalNumInverse[i] *
            factorialNumInverse[i - 1]) % p;
}

// Function to calculate factorial of 1 to N
void factorial(ll p)
{
    fact[0] = 1;

    // precompute factorials
    for (int i = 1; i <= N; i++) {
        fact[i] = (fact[i - 1] * i) % p;
    }
}

// Function to return nCr % p in O(1) time
ll Binomial(ll N, ll R, ll p)
{
```

```
    // n C r = n!*inverse(r!)*inverse((n-r)!)
    ll ans = ((fact[N] * factorialNumInverse[R])
              % p * factorialNumInverse[N - R])
             % p;
    return ans;
}

// Driver Code
int main()
{
    // Calling functions to precompute the
    // required arrays which will be required
    // to answer every query in O(1)
    ll p = 1000000007;
    InverseofNumber(p);
    InverseofFactorial(p);
    factorial(p);

    // 1st query
    ll N = 15;
    ll R = 4;
    cout << Binomial(N, R, p) << endl;

    // 2nd query
    N = 20;
    R = 3;
    cout << Binomial(N, R, p) << endl;

    return 0;
}
```

6 Numerical Methods

6.1 Ternary Search

```
double ternary_search(double l, double r) {
    double eps = 1e-9;              //set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);    //evaluates the function at m1
        double f2 = f(m2);    //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);                    //return the maximum of f(x) in [l, r]
}
```

7 Strings

7.1 KMP

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

7.2 Rabin Karp

```
vector<int> rabin_karp(string const& s, string const& t) {
    const int p = 31;
    const int m = 1e9 + 9;
    int S = s.size(), T = t.size();

    vector<long long> p_pow(max(S, T));
    p_pow[0] = 1;
    for (int i = 1; i < (int)p_pow.size(); i++)
        p_pow[i] = (p_pow[i-1] * p) % m;

    vector<long long> h(T + 1, 0);
    for (int i = 0; i < T; i++)
```

```
        h[i+1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
    long long h_s = 0;
    for (int i = 0; i < S; i++)
        h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;

    vector<int> occurences;
    for (int i = 0; i + S - 1 < T; i++) {
        long long cur_h = (h[i+S] + m - h[i]) % m;
        if (cur_h == h_s * p_pow[i] % m)
            occurences.push_back(i);
    }
    return occurences;
}
```

8 Template

8.1 C++ Snippet

```
#include <bits/stdc++.h>
using namespace std;
#define MOD 1000000007
typedef long long ll;
//#define int ll

typedef pair<ll, ll> ii;
typedef vector<ll> vi;
typedef vector<bool> vb;
typedef vector<vi> vvi;
typedef vector<ii> vii;
typedef vector<vii> vvii;
#define ff first
#define ss second
#define pb push_back
#define all(s) s.begin(), s.end()
#define tc int t; cin>>t; while(t--)
#define file_read(x,y) freopen(x, "r", stdin); \
                                           freopen(y, "w", stdout);
#define fightFight cin.tie(0) -> sync_with_stdio(0)

int main(){
       fightFight;

}
```