Team notebook

Richard Alison

October 1, 2021

Contents

1 Data Structures

1.1 STL PBDS

```cpp
// ost = ordered set
// omp = ordered map
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;

template<class T>
using ost = tree<T, null_type, less<T>, rb_tree_tag,
          tree_order_statistics_node_update>;
template<class T, class U>
using omp = tree<T, U, less<T>, rb_tree_tag,
          tree_order_statistics_node_update>;
```

1.2 Treap

```cpp
// Complexity: O(log N) for split and merge
//
// empty treap: Treap* tr = nullptr;
// insert v at x: [l, r] = split(tr, x), m = Treap(v), merge lmr
```

```cpp
// delete at x: [l, r] = split(tr, x), [m, r] = split(r, 1), merge lr
// lazy prop: propagate every time a node is accessed

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

using Key = int;

struct Treap
{
    Key val;
    Treap* left;
    Treap* right;
    int prio, sz;
    Treap() {}
    Treap(int _val);
};

int size(Treap* tr)
{
    return tr ? tr->sz : 0;
}

void update(Treap* tr)
{
    tr->sz = 1 + size(tr->left) + size(tr->right);
}

Treap::Treap(Key _val) :
    val(_val), left(nullptr), right(nullptr), prio(rng())
{
    update(this);
}

pair<Treap*, Treap*> split(Treap* tr, int sz)
{
    if(!tr) return {nullptr, nullptr};
    int left_sz = size(tr->left);
    if(sz <= left_sz)
    {
        auto [left, mid] = split(tr->left, sz);
        tr->left = mid;
        update(tr);
        return {left, tr};
    }
    else
    {
        auto [mid, right] = split(tr->right, sz - left_sz - 1);
        tr->right = mid;
        update(tr);
        return {tr, right};
    }
}

Treap* merge(Treap* l, Treap* r)
{
    if(!l) return r;
    if(!r) return l;
    if(l->prio < r->prio)
    {
        l->right = merge(l->right, r);
        update(l);
        return l;
    }
    else
    {
        r->left = merge(l, r->left);
        update(r);
        return r;
    }
}
```

2 Geometry

2.1 Smallest Enclosing Circle

```cpp
// Welzl's algorithm to find the smallest circle
// that encloses a group of poins in O(N * ITERS)
// returns {radius, x, y}
const int ITERS = 3e5;
const double INF = 1e12;

tuple<double, double, double> welzl(const vector<pair<int, int>>& points)
{
    double xt = 0, yt = 0;
    for(auto& [x, y] : points)
```

```
    {
        xt += x;
        yt += y;
    }
    xt /= points.size();
    yt /= points.size();
    double p = 0.1;
    double mx_d;
    for(int i = 0; i < ITERS; ++i)
    {
        mx_d = -INF;
        int mx_idx = -1;
        for(int j = 0; j < (int) points.size(); ++j)
        {
            double cx = xt - points[j].first;
            double cy = yt - points[j].second;
            double cur = cx * cx + cy * cy;
            if(cur > mx_d)
            {
                mx_d = cur;
                mx_idx = j;
            }
        }
        xt += (points[mx_idx].first - xt) * p;
        yt += (points[mx_idx].second - yt) * p;
        p *= 0.999;
    }
    return {sqrt(mx_d), xt, yt};
}
```

3 Graphs

3.1 Articulation Point Bridge

```
// gr -> adj list
// vector vis, low -> initialize to -1
// int timer -> initialize to 0
void dfs(int pos, int dad = -1)
{
    vis[pos] = low[pos] = timer++;
    int kids = 0;
    for(auto& i : gr[pos])
    {
        if(i == dad) continue;
        if(vis[i] >= 0)
            low[pos] = min(low[pos], vis[i]);
        else
        {
            dfs(i, pos);
            low[pos] = min(low[pos], low[i]);
            if(low[i] > vis[pos])
                is_bridge(pos, i)
            if(low[i] >= vis[pos] && dad >= 0)
                is_articulation_point(pos)
            ++kids;
        }
    }
    if(dad == -1 && kids > 1)
        is_articulation_point(pos)
}
```

3.2 Dinic's Maximum Flow

```
// O(VE log(max_flow)) if scaling == 1
// O((V + E) sqrt(E)) if unit graph (turn scaling off)
// O((V + E) sqrt(V)) if bipartite matching (turn scaling off)
// indices are 0-based
const ll INF = 1e18;

struct Dinic
{
    struct Edge
    {
        int v;
        ll cap, flow;
        Edge(int _v, ll _cap) : v(_v), cap(_cap), flow(0) {}
    };

    int n;
    ll lim;
    vector<vector<int>> gr;
    vector<Edge> e;
    vector<int> idx, lv;
```

```
    bool has_path(int s, int t)
    {
        queue<int> q;
        q.push(s);
        lv.assign(n, -1);
        lv[s] = 0;
        while(!q.empty())
        {
            int c = q.front();
            q.pop();
            if(c == t) break;
            for(auto& i : gr[c])
            {
                ll cur_flow = e[i].cap - e[i].flow;
                if(lv[e[i].v] == -1 && cur_flow >= lim)
                {
                    lv[e[i].v] = lv[c] + 1;
                    q.push(e[i].v);
                }
            }
        }
        return lv[t] != -1;
    }

    ll get_flow(int s, int t, ll left)
    {
        if(!left || s == t) return left;
        while(idx[s] < (int) gr[s].size())
        {
            int i = gr[s][idx[s]];
            if(lv[e[i].v] == lv[s] + 1)
            {
                ll add = get_flow(
                    e[i].v,
                    t,
                    min(left, e[i].cap - e[i].flow)
                );
                if(add)
                {
                    e[i].flow += add;
                    e[i ^ 1].flow -= add;
                    return add;
                }
            }
            ++idx[s];
        }
        return 0;
    }

    Dinic(int vertices, bool scaling = 1) // toggle scaling here
        : n(vertices), lim(scaling ? 1 << 30 : 1), gr(n) {}

    void add_edge(int from, int to, ll cap, bool directed = 1)
    {
        gr[from].push_back(e.size());
        e.emplace_back(to, cap);
        gr[to].push_back(e.size());
        e.emplace_back(from, directed ? 0 : cap);
    }

    ll get_max_flow(int s, int t) // call this
    {
        ll res = 0;
        while(lim) // scaling
        {
            while(has_path(s, t))
            {
                idx.assign(n, 0);
                while(ll add = get_flow(s, t, INF)) res += add;
            }
            lim >>= 1;
        }
        return res;
    }
};
```

3.3 Edmonds' Blossom

```
// Maximum matching on general graphs in O(V^2 E)
// Indices are 1-based
// Stolen from ko_osaga's cheatsheet
struct Blossom
{
    vector<int> vis, dad, orig, match, aux;
    vector<vector<int>> conn;
```

```
int t, N;
queue<int> Q;

void augment(int u, int v)
{
    int pv = v;
    do
    {
        pv = dad[v];
        int nv = match[pv];
        match[v] = pv;
        match[pv] = v;
        v = nv;
    } while(u != pv);
}

int lca(int v, int w)
{
    ++t;
    while(true)
    {
        if(v)
        {
            if(aux[v] == t) return v;
            aux[v] = t;
            v = orig[dad[match[v]]];
        }
        swap(v, w);
    }
}

void blossom(int v, int w, int a)
{
    while(orig[v] != a)
    {
        dad[v] = w;
        w = match[v];
        if(vis[w] == 1)
        {
            Q.push(w);
            vis[w] = 0;
        }
        orig[v] = orig[w] = a;
        v = dad[w];
    }
}

bool bfs(int u)
{
    fill(vis.begin(), vis.end(), -1);
    iota(orig.begin(), orig.end(), 0);
    Q = queue<int>();
    Q.push(u);
    vis[u] = 0;
    while(!Q.empty())
    {
        int v = Q.front(); Q.pop();
        for(int x : conn[v])
        {
            if(vis[x] == -1)
            {
                dad[x] = v; vis[x] = 1;
                if(!match[x])
                {
                    augment(u, x);
                    return 1;
                }
                Q.push(match[x]);
                vis[match[x]] = 0;
            }
            else if(vis[x] == 0 && orig[v] != orig[x])
            {
                int a = lca(orig[v], orig[x]);
                blossom(x, v, a);
                blossom(v, x, a);
            }
        }
    }
    return false;
}

Blossom(int n) : // n = vertices
    vis(n + 1), dad(n + 1), orig(n + 1), match(n + 1),
    aux(n + 1), conn(n + 1), t(0), N(n)
{
    for(int i = 0; i <= n; ++i)
    {
        conn[i].clear();
```

```cpp
            match[i] = aux[i] = dad[i] = 0;
        }
    }

    void add_edge(int u, int v)
    {
        conn[u].push_back(v);
        conn[v].push_back(u);
    }

    int solve() // call this for answer
    {
        int ans = 0;
        vector<int> V(N - 1);
        iota(V.begin(), V.end(), 1);
        shuffle(V.begin(), V.end(), mt19937(0x94949));
        for(auto x : V)
        {
            if(!match[x])
            {
                for(auto y : conn[x])
                {
                    if(!match[y])
                    {
                        match[x] = y, match[y] = x;
                        ++ans;
                        break;
                    }
                }
            }
        }
        for(int i = 1; i <= N; ++i)
        {
            if(!match[i] && bfs(i)) ++ans;
        }
        return ans;
    }
};
```

3.4 Eulerian Path or Cycle

```cpp
// finds a eulerian path / cycle
// visits each edge only once
// properties:
// - cycle: degrees are even
// - path: degrees are even OR degrees are even except for 2 vertices
// how to use: g = adjacency list g[n] = connected to n, undirected
// if there is a vertex u with an odd degree, call dfs(u)
// else call on any vertex
// ans = path result

vector<set<int>> g;
vector<int> ans;

void dfs(int u)
{
    while(g[u].size())
    {
        int v = *g[u].begin();
        g[u].erase(v);
        g[v].erase(u);
        dfs(v);
    }
    ans.push_back(u);
}
```

3.5 Minimum Cost Maximum Flow

```cpp
// 1-based index
template<class T>
using rpq = priority_queue<T, vector<T>, greater<T>>;

const ll INF = 1e18;

struct MCMF
{
    struct Edge
    {
        int v;
        ll cap, cost;
        int rev;
        Edge(int _v, ll _cap, ll _cost, int _rev) :
            v(_v), cap(_cap), cost(_cost), rev(_rev) {}
    };
```

```cpp
    ll flow, cost;
    int st, ed, n;
    vector<ll> dist, H;
    vector<int> pv, pe;
    vector<vector<Edge>> adj;

    bool dijkstra()
    {
        rpq<pair<ll, int>> pq;
        dist.assign(n + 1, INF);
        dist[st] = 0;
        pq.emplace(0, st);
        while(!pq.empty())
        {
            auto [cst, pos] = pq.top();
            pq.pop();
            if(dist[pos] < cst) continue;
            for(int i = 0; i < (int) adj[pos].size(); ++i)
            {
                auto& e = adj[pos][i];
                int nxt = e.v;
                ll nxt_cst = dist[pos] + e.cost + H[pos] - H[nxt];
                if(e.cap > 0 && nxt_cst < dist[nxt])
                {
                    dist[nxt] = nxt_cst;
                    pe[nxt] = i;
                    pv[nxt] = pos;
                    pq.emplace(nxt_cst, nxt);
                }
            }
        }
        return dist[ed] != INF;
    }

    MCMF(int _n) : n(_n), pv(n + 1), pe(n + 1), adj(n + 1) {}

    void add_edge(int u, int v, ll cap, ll cst)
    {
        adj[u].emplace_back(v, cap, cst, adj[v].size());
        adj[v].emplace_back(u, 0, -cst, adj[u].size() - 1);
    }

    pair<ll, ll> solve(int _st, int _ed)
    {
        st = _st, ed = _ed;
        flow = 0, cost = 0;
        H.assign(n + 1, 0);
        while(dijkstra())
        {
            for(int i = 0; i <= n; ++i)
                H[i] += dist[i];
            ll f = INF;
            for(int i = ed; i != st; i = pv[i])
                f = min(f, adj[pv[i]][pe[i]].cap);
            flow += f;
            cost += f * H[ed];
            for(int i = ed; i != st; i = pv[i])
            {
                auto& e = adj[pv[i]][pe[i]];
                e.cap -= f;
                adj[i][e.rev].cap += f;
            }
        }
        return {flow, cost};
    }
};
```

4 Math

4.1 Euler's Totient

```cpp
// Precompute up to n in O(n log log n)
vector<int> 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;
    return phi;
```

```
}

// Calculate for a single n in O(sqrt(n))
ll totient(ll n)
{
    ll res = 1;
    for(ll i = 2; i * i <= n; ++i)
    {
        if(n % i == 0)
        {
            res *= i - 1;
            n /= i;
        }
        while(n % i == 0)
        {
            res *= i;
            n /= i;
        }
    }
    if(n > 1) res *= n - 1;
    return res;
}
```

4.2 Extended Euclidean GCD

```
// computes x and y such that ax + by = gcd(a, b) in O(log (min(a, b)))
// returns {gcd(a, b), x, y}
tuple<int, int, int> gcd(int a, int b)
{
    if(b == 0) return {a, 1, 0};
    auto [d, x1, y1] = gcd(b, a % b);
    return {d, y1, x1 - y1 * (a / b)};
}
```

4.3 Fibonacci Check

```
bool is_fibonacci(int n)
{
    return is_perfect_square(5 * n * n + 4)
        || is_perfect_square(5 * n * n - 4);
}
```

4.4 Matrix Multiplication

```
using Mat = vector<vector<ll>>;

Mat multiply(const Mat& a, const Mat& b)
{
    assert(a[0].size() == b.size());
    int y = a.size(), x = b[0].size(), n = b.size();
    Mat res(y, vector<ll>(x));
    for(int i = 0; i < y; ++i)
        for(int k = 0; k < n; ++k)
            for(int j = 0; j < x; ++j)
                res[i][j] += a[i][k] * b[k][j];
    return res;
}
```

4.5 Miller-Rabin Pollard's Rho

```
namespace MillerRabin
{
    const vector<ll> primes = { // deterministic up to 2^64 - 1
        2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37
    };
    ll gcd(ll a, ll b)
    {
        return b ? gcd(b, a % b) : a;
    }
    ll powa(ll x, ll y, ll p) // (x ^ y) % p
    {
        if(!y) return 1;
        if(y & 1) return ((__int128) x * powa(x, y - 1, p)) % p;
        ll temp = powa(x, y >> 1, p);
        return ((__int128) temp * temp) % p;
    }
    bool miller_rabin(ll n, ll a, ll d, int s)
    {
        ll x = powa(a, d, n);
        if(x == 1 || x == n - 1) return 0;
```

```cpp
        for(int i = 0; i < s; ++i)
        {
            x = ((__int128) x * x) % n;
            if(x == n - 1) return 0;
        }
        return 1;
    }
    bool is_prime(ll x) // use this
    {
        if(x < 2) return 0;
        int r = 0;
        ll d = x - 1;
        while((d & 1) == 0)
        {
            d >>= 1;
            ++r;
        }
        for(auto& i : primes)
        {
            if(x == i) return 1;
            if(miller_rabin(x, i, d, r)) return 0;
        }
        return 1;
    }
}

namespace PollardRho
{
    mt19937_64 generator(chrono::steady_clock::now()
                    .time_since_epoch().count());
    uniform_int_distribution<ll> rand_ll(0, LLONG_MAX);
    ll f(ll x, ll b, ll n) // (x^2 + b) % n
    {
        return (((__int128) x * x) % n + b) % n;
    }
    ll rho(ll n)
    {
        if(n % 2 == 0) return 2;
        ll b = rand_ll(generator);
        ll x = rand_ll(generator);
        ll y = x;
        while(1)
        {
            x = f(x, b, n);
            y = f(f(y, b, n), b, n);
            ll d = MillerRabin::gcd(abs(x - y), n);
            if(d != 1) return d;
        }
    }
    void pollard_rho(ll n, vector<ll>& res)
    {
        if(n == 1) return;
        if(MillerRabin::is_prime(n))
        {
            res.push_back(n);
            return;
        }
        ll d = rho(n);
        pollard_rho(d, res);
        pollard_rho(n / d, res);
    }
    vector<ll> factorize(ll n, bool sorted = 1) // use this
    {
        vector<ll> res;
        pollard_rho(n, res);
        if(sorted) sort(res.begin(), res.end());
        return res;
    }
}
```

5 Miscellaneous

5.1 Dates

5.1.1 Day of Date

```cpp
// 0-based
const vector<int> T = {
    0, 3, 2, 5, 0, 3,
    5, 1, 4, 6, 2, 4
}

int day(int d, int m, int y)
{
    y -= (m < 3);
```

```
    return (y + y / 4 - y / 100 + y / 400 + T[m - 1] + d) % 7;
}
```

5.1.2 Number of Days since 1-1-1

```
int rdn(int d, int m, int y)
{
    if(m < 3) --y, m += 12;
    return 365 * y + y / 4 - y / 100 + y / 400
        + (153 * m - 457) / 5 + d - 306;
}
```

5.2 Enumerate Subsets of a Bitmask

```
int x = 0;
do
{
    // do stuff with the bitmask here
    x = (x + 1 + ~m) & m;
} while(x != 0);
```

5.3 Int to Roman

```
const string R[] = {
    "M", "CM", "D", "CD", "C", "XC", "L",
    "XL", "X", "IX", "V", "IV", "I"
};

const int N[] = {
    1000, 900, 500, 400, 100, 90,
    50, 40, 10, 9, 5, 4, 1
};

string to_roman(int x)
{
    if (x == 0) return "O"; // Not decimal 0!
    string res = "";
    for (int i = 0; i < 13; ++i)
        while (x >= N[i]) x -= N[i], res += R[i];
    return res;
}
```

5.4 Josephus Problem

```
ll josephus(ll n, ll k) // O(k log n)
{
    if(n == 1) return 0;
    if(k == 1) return n - 1;
    if(k > n) return (josephus(n - 1, k) + k) % n;
    ll cnt = n / k;
    ll res = josephus(n - cnt, k);
    res -= n % k;
    if(res < 0) res += n;
    else res += res / (k - 1);
    return res;
}

int josephus(int n, int k) // O(n)
{
    int res = 0;
    for(int i = 1; i <= n; ++i)
        res = (res + k) % i;
    return res + 1;
}
```

6 Strings

6.1 Knuth-Morris-Pratt

```
// Constructs KMP failure function in O(n)
vector<int> kmp(const string& s)
{
    vector<int> res(s.length());
    int i = 1, j = 0;
    while(i < (int) s.length())
    {
        if(s[i] == s[j]) res[i++] = ++j;
```

```
        else if(j > 0) j = res[j - 1];
        else res[i++] = 0;
    }
    return res;
}
```

6.2 Suffix Array

```
// stores result in sa and lcp
// if lcp is needed, call SuffixArray(str, 1)
struct SuffixArray
{
    int n;
    vector<int> sa, lcp, rnk, cnt;
    vector<pair<int, int>> p;
    SuffixArray(const string& s, bool calc_lcp = 0) :
        n(s.length()), sa(n), lcp(calc_lcp ? n : 0), rnk(n),
        cnt(max(n, 256)), p(n)
    {
        for(int i = 0; i < n; ++i) rnk[i] = s[i];
        iota(sa.begin(), sa.end(), 0);
        for(int i = 1; i < n; i <<= 1) update_sa(i);
        if(!calc_lcp) return;
        vector<int> phi(n), plcp(n);
        phi[sa[0]] = -1;
        for(int i = 1; i < n; ++i) phi[sa[i]] = sa[i - 1];
        int l = 0;
        for(int i = 0; i < n; ++i)
        {
            if(phi[i] == -1) plcp[i] = 0;
            else
            {
                while((i + l < n) && (phi[i] + l < n)
                      && (s[i + l] == s[phi[i] + l])) ++l;
                plcp[i] = l;
                l = max(l - 1, 0);
            }
        }
        for(int i = 0; i < n; ++i) lcp[i] = plcp[sa[i]];
    }
    void update_sa(int len)
    {
        sort_sa(len); sort_sa(0);
        for(int i = 0; i < n; ++i) p[i] = {rnk[i], rnk[(i + len) % n]};
        auto lst = p[sa[0]];
        rnk[sa[0]] = 0;
        int cur = 0;
        for(int i = 1; i < n; ++i)
        {
            if(lst != p[sa[i]])
            {
                lst = p[sa[i]];
                ++cur;
            }
            rnk[sa[i]] = cur;
        }
    }
    void sort_sa(int offset)
    {
        fill(cnt.begin(), cnt.end(), 0);
        for(int i = 0; i < n; ++i) ++cnt[rnk[(i + offset) % n]];
        int sum = 0;
        for(int i = 0; i < (int) cnt.size(); ++i)
        {
            int temp = cnt[i];
            cnt[i] = sum;
            sum += temp;
        }
        vector<int> temp(n);
        for(int i = 0; i < n; ++i)
        {
            int cur = cnt[rnk[(sa[i] + offset) % n]]++;
            temp[cur] = sa[i];
        }
        sa = move(temp);
    }
};
```