Contents

1 Math

1.1 Power, Little Fermat Theorem, Combination

```
int MOD = 998244353;
int power(int a, int n) {
    int res = 1;
    for (;n; n >>= 1, a = (a * a) % MOD)
        if (n & 1) res = (res * a) % MOD;
    return res;
}
int inv(int a) { return power(a, MOD - 2); }
int C(int n, int r) {
    if (n < r)
        return 0;
    return fact[n] % MOD * inv(fact[n - r]) % MOD * inv(fact[r]) % MOD;
}
```

1.2 Extended Euclidean Algorithm

```
pair<int, int> extended_euclidean(int a, int b) {
    int s = 1, ss = 0, t = 0, tt = 1, r0 = a, r1 = b, q;
    while (r1 > 0) {
        q = r0 / r1;
        tie(r0, r1) = make_tuple(r1, r0 - q * r1);
        tie(s, ss) = make_tuple(ss, s - q * ss);
        tie(t, tt) = make_tuple(tt, t - q * tt);
    }
    return {s, t};
}
```

1.3 Euler's Phi Function

```
int phi(int n) {
    int ret = n;
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            while (n % i == 0)
```

```
                n /= i;
            ret -= ret / i;
        }
    }
    if (n > 1) ret -= ret / n;
    return ret;
}
```

1.4 FFT(Fast Fourier Transform)

```
#define sz(v) ((int)(v).size())
#define all(v) (v).begin(),(v).end()
typedef complex<double> base;
void fft(vector <base> &a, bool invert){
    int n = sz(a);
    for (int i=1,j=0;i<n;i++){
        int bit = n >> 1;
        for (;j>=bit;bit>>=1) j -= bit;
        j += bit;
        if (i < j) swap(a[i],a[j]);
    }
    for (int len=2;len<=n;len<<=1){
        double ang = 2*M_PI/len*(invert?-1:1);
        base wlen(cos(ang),sin(ang));
        for (int i=0;i<n;i+=len){
            base w(1);
            for (int j=0;j<len/2;j++){
                base u = a[i+j], v = a[i+j+len/2]*w;
                a[i+j] = u+v;
                a[i+j+len/2] = u-v;
                w *= wlen;
            }
        }
    }
    if (invert){
        for (int i=0;i<n;i++) a[i] /= n;
    }
}
void multiply(const vector<int> &a,const vector<int> &b,vector<int> &res){
    vector <base> fa(all(a)), fb(all(b));
    int n = 1;
    while (n < max(sz(a),sz(b))) n <<= 1;
    n<<=1;
    fa.resize(n); fb.resize(n);
    fft(fa,false); fft(fb,false);
    for (int i=0;i<n;i++) fa[i] *= fb[i];
    fft(fa,true);
    res.resize(n);
    for (int i=0;i<n;i++) res[i] = (int)(fa[i].real()+(fa[i].real()>0?0.5:-0.5));
}
```

2 String

2.1 Suffix Array

```
// 0-based, Capitalized
#define forn(i, n) for(int i = 0; i < n; i++)
int conv(char c) { return c - 'A'; }
void get_sa(const char *s, int n, vector<int> &sa) {
    sa.resize(n);
    constexpr int m = 26; // 문자의갯수
    vector<int> cnt(max(n, m)), x(n), y(n);
    forn(i, n) cnt[x[i] = conv(s[i])]++;
    forn(i, m - 1) cnt[i + 1] += cnt[i];
    for (int i = n - 1; i >= 0; --i)
        sa[--cnt[x[i]]] = i;
    for (int len = 1, p = 0; p + 1 < n; len <<= 1, m = p + 1) {
        p = 0;
        for (int i = n - len; i < n; ++i) y[p++] = i;
        forn(i, n) if (sa[i] >= len) y[p++] = sa[i] - len;

        forn(i, m) cnt[i] = 0;
        forn(i, n) cnt[x[i]]++;
        forn(i, m - 1) cnt[i + 1] += cnt[i];
        for (int i = n - 1; i >= 0; --i) sa[--cnt[x[y[i]]]] = y[i];
        y = x;
        p = 0;
        x[sa[0]] = 0;
        forn(i, n - 1)
```

```
            x[sa[i + 1]] = sa[i] + len < n && sa[i + 1] + len < n && y[sa[i]] == y[sa[i + 1]] && y[sa[i] + len]
              == y[sa[i + 1] + len] ? p : ++p;
    }
}
void get_lcp(const char *s, int n, vector<int> &sa, vector<int> &lcp) {
    lcp.resize(n);
    vector<int> rank(sa.size()); //상황에 따라도 rank 인자로받아야함
    forn(i, n) rank[sa[i]] = i;
    int k = 0, j;
    for (int i = 0; i < n; lcp[rank[i++]] = k) {
        if (rank[i] - 1 >= 0)
            for (k ? k-- : 0, j = sa[rank[i] - 1]; s[i + k] == s[j + k]; ++k);
    }
}
```

2.2 KMP

```
// 문자열길이  >= 1, xxxa..xxx fail[end of x] => a
// 0-based
void get_fail(const char* p, int len, vector<int>& fail) {
    fail.resize(len);
    fail[0] = 0;
    for (int i = 1, j = 0; i < len; ++i) {
        while (j && p[i] != p[j]) j = fail[j - 1];
        if (p[i] == p[j]) fail[i] = ++j;
    }
}
void KMP(const char* text, int tlen, const char* pattern, int plen, vector<int>& fail, vector<int>& ans) {
    ans.clear();
    for (int i = 0, j = 0; i < tlen; ++i) {
        while (j && text[i] != pattern[j]) j = fail[j - 1];
        if (text[i] == pattern[j]) {
            if (j == plen - 1) {
                ans.push_back(i - j);
                j = fail[j];
            } else
                ++j;
        }
    }
}
```

2.3 Trie

```
struct trie {
    int conv(char c) { return c - 'a'; }
    struct vertex {
        int nxt[26]; // 알파벳갯수
        int finish;  // if not finish, -1
        // 0-를based 가정하는것
        vertex() {
            memset(nxt, -1, sizeof nxt);
            finish = -1;
        }
    };
    vector<vertex> data;
    int head; // data[head]: 의head vertex
    int size; // 단어의총개수
    trie(int cap = 30000) {
        // trie 자체가용량을많이잡아먹기에생성자에서을      cap 잘지정하여매모리를아낀다
        // cap = 단어의( 갯수 * 단어의길이  + 1)
        data.reserve(cap);
        head = 0;
        size = 0;
        data.push_back(vertex());
    }
    void add(const char *str) {
        int v_idx = head;
        for (int idx = 0; str[idx]; ++idx) {
            if (data[v_idx].nxt[conv(str[idx])] == -1) {
                data[v_idx].nxt[conv(str[idx])] = data.size();
                data.push_back(vertex());
            }
            v_idx = data[v_idx].nxt[conv(str[idx])];
        }
        data[v_idx].finish = size++; // 만약중복된단어가들어올수있다면      , 각노드별로   count 변수를만들어야한다   .
    }
    int retrieve(const char *str) { // 찾지못하면  -1, 찾았다면되었던 add 순서를리턴부터 (0 시작)
        int v_idx = head;
        for (int idx = 0; str[idx]; ++idx) {
            if (data[v_idx].nxt[conv(str[idx])] == -1)
```

```
                return -1;
            v_idx = data[v_idx].nxt[conv(str[idx])];
        }
        return data[v_idx].finish;
    }
};
```

2.4 Manacher's Algorithm

```
void manacher(const char* str, int len, vector<int>& a) {  // 사용전문자앞뒤로     # 추가하기
    a.resize(len);
    int p = 0;
    for (int i = 0; i < len; ++i) {
        if (i <= p + a[p])
            a[i] = min(a[2 * p - i], p + a[p] - i);  // 점를 p 기준으로대칭
        else
            a[i] = 0;
        while (i - a[i] - 1 >= 0 && i + a[i] + 1 < len && str[i - a[i] - 1] == str[i + a[i] + 1])  // 확장
            ++a[i];
        if (p + a[p] < i + a[i]) p = i;  // p = 최대커버
    }
}
```

3 Data Structure

3.1 Segment Tree with Lazy Propagation

```
struct SegTree {
    vector<int> tree, lazy;
    int base;
    SegTree(int a) {
        base = 1;
        while (base < a) base <<= 1;
        tree.resize((base + 1) << 1);
        lazy.resize((base + 1) << 1); // FOR LAZY-PROPAGATION
        base--;
    }
    void update(int idx, int val) {
        idx += base;
        tree[idx] = val;
        idx >>= 1;
        while (idx != 0) {
            tree[idx] = tree[idx * 2] + tree[idx * 2 + 1];
            idx >>= 1;
        }
    }
    int query(int l, int r, int s = 1, int e = -1, int node = 1) {
        if (e == -1) e = base + 1;
        propagate(s, e, node); // FOR LAZY-PROPAGATION
        if (l > e || r < s) return 0;
        if (l <= s && e <= r) return tree[node];
        return query(l, r, s, (s + e) / 2, node * 2) +
               query(l, r, (s + e) / 2 + 1, e, node * 2 + 1);
    }
    ////////// UNDERNEATH HERE IS FOR LAZY-PROPAGATION //////////
    void propagate(int s, int e, int node) {
        if (lazy[node] == 0) return;
        if (s < e) {
            lazy[node * 2] += lazy[node];
            lazy[node * 2 + 1] += lazy[node];
        }
        tree[node] += lazy[node] * (e - s + 1);
        lazy[node] = 0;
    }
    void add_val(int val, int l, int r, int s = 1, int e = -1, int node = 1) {
        if (e == -1) e = base + 1;
        propagate(s, e, node);
        if (l > e || r < s) return;
        if (l <= s && e <= r) {
            lazy[node] += val;
            propagate(s, e, node);
        } else {
            add_val(val, l, r, s, (s + e) / 2, node * 2);
            add_val(val, l, r, (s + e) / 2 + 1, e, node * 2 + 1);
            tree[node] = tree[node * 2] + tree[node * 2 + 1];
        }
    }
};
```

3.2 Ordered Set

```
#define ordered_set tree<int, null_type,less<int>, rb_tree_tag,tree_order_statistics_node_update>
```

3.3 Merge Sort Tree

```
#define all(v) (v).begin(),(v).end()
struct merge_sort_tree{
    vector<vector<int>> tree;
    vector<int> a;
    int n;
    int base;
    merge_sort_tree(int n): n(n) {
        base = 1;
        while(base < n) base <<= 1;
        tree.resize(base*2 - 1);
        --base;
        a.resize(n);
    }
    void init() {
        for (int i = base; i<base+n;++i) {
            tree[i].resize(1);
            tree[i][0] = a[i-base];
        }
        for (int i=base-1; i>=0; --i) {
            tree[i].resize(tree[i*2+1].size() + tree[i*2+2].size());
            merge(all(tree[2*i+1]), all(tree[2*i+2]), tree[i].begin());
        }
    }
    //[qs, qe에서] 이하val 몇개?
    int qry_ub(int qs, int qe, int val) {
        int ret = 0;
        qs += base;
        qe += base+1;
        for(; qs < qe; qs = qs / 2, qe = qe / 2){
            if (qs%2 == 0){
                ret += upper_bound(all(tree[qs]), val) - tree[qs].begin();
                ++qs;
            }
            if (qe%2 == 0){
                --qe;
                ret += upper_bound(all(tree[qe]), val) - tree[qe].begin();
            }
        }
        return ret;
    }
    // [qs, qe에서] 미만val 몇개?
    int qry_lb(int qs, int qe, int val){
        int ret = 0;
        qs += base;
        qe += base+1;
        for(; qs < qe; qs = qs / 2, qe = qe / 2){
            if (qs%2 == 0){
                ret += lower_bound(all(tree[qs]), val) - tree[qs].begin();
                ++qs;
            }
            if (qe%2 == 0){
                --qe;
                ret += lower_bound(all(tree[qe]), val) - tree[qe].begin();
            }
        }
        return ret;
    }
    //배열 에서a, [l,r에] 속하는값들만 '' 고려했을때  , 번째k 원소의를 index 반환
    int qry_kth(int l, int r, int k){
        int node = 0;
        while(node < base){
            int c = upper_bound(all(tree[node*2+1]), r) -
                    lower_bound(all(tree[node*2+1]), l);
            if (c >= k){
                node = 2*node + 1;
            }
            else{
                node = 2*node + 2;
                k -= c;
            }
        }
        return node - base;
    }
};
```

3.4 Heavy Light Decomposition

```
constexpr int MAXN = 111111;

vector<int> graph[MAXN];
int par[MAXN], sub[MAXN], dep[MAXN];
void dfs(int now, int before) {
    par[now] = before;
    sub[now] = 1;
    for(int nxt: graph[now]) if (nxt != before)
        {
            dep[nxt] = dep[now] + 1;
            dfs(nxt, now);
            sub[now] += sub[nxt];
        }
}
vector<int> chain[MAXN];
int wt_chain[MAXN], in_chain[MAXN], ps[MAXN], occ = 0;
void hld(int now, int before, int cur_chain) {
    wt_chain[now] = cur_chain;
    in_chain[now] = chain[cur_chain].size();
    chain[cur_chain].push_back(now);
    int hvy = -1;
    for (int nxt: graph[now]) if (nxt != before)
        {
            if (hvy==-1 || sub[nxt] > sub[hvy]) hvy = nxt;
        }
    if (hvy != -1) hld(hvy, now,  cur_chain);
    for (int nxt: graph[now]) if (nxt != before)
        {
            if (nxt != hvy) hld(nxt, now, nxt);
        }
    if (cur_chain == now)
    {
        ps[cur_chain] = occ;
        occ += chain[cur_chain].size();
    }
}
//위치 공식 : ps[wt_chain[a]]+in_chain[a]
int get_lca(int a, int b)
{
    while(wt_chain[a] != wt_chain[b])
    {
        if (dep[wt_chain[a]] > dep[wt_chain[b]])
            a = par[wt_chain[a]];
        else
            b = par[wt_chain[b]];
    }
    if (in_chain[a] > in_chain[b])
        return b;
    else
        return a;
}
```

4 Graph

4.1 SCC: Kosaraju's Algorithm

```
int n;
stack<int> st;
vector<bool> check;
vector<vector<int>> adj, adj_r;

void dfs(int curr) {
    check[curr] = true;
    for (auto& next : adj[curr])
        if (!check[next]) dfs(next);
    st.push(curr);
}
vector<vector<int>> kosaraju() {
    check.resize(n);
    for (int i = 0; i < n; i++)
        if (!check[i]) dfs(i);
    check = vector<bool>(n);
    vector<vector<int>> ret;
    stack<int> sk;
    vector<int> comp;
    while (!st.empty()) {
        int node = st.top();
```

```
        st.pop();
        if (check[node]) continue;
        sk.push(node);
        comp.clear();
        while (!sk.empty()) {
            int curr = sk.top();
            sk.pop();
            check[curr] = true;
            comp.push_back(curr);
            for (auto next : adj_r[curr])
                if (!check[next]) sk.push(next);
        }
        ret.push_back(comp);
    }
    return ret;
}
```

5 Flow

5.1 Dinic

```
struct dinic {
    typedef int flow_t;

    struct edge {
        int nxt;
        flow_t res;  // residual flow
        size_t inv;  // inverse edge index
        edge(int n, flow_t r, size_t v) : nxt(n), res(r), inv(v) {}
    };

    int n;
    vector<vector<edge>> adj;
    vector<int> q, lvl, start;

    dinic(int _n) {
        n = _n;
        adj.resize(n);
    }
    void add_edge(int a, int b, flow_t cap, flow_t cap_inv=0) {
        adj[a].emplace_back(b, cap, adj[b].size());
        adj[b].emplace_back(a, cap_inv, adj[a].size() - 1);
    }
    bool assign_level(int src, int sink) {
        int t = 0;
        memset(&lvl[0], 0, sizeof(lvl[0]) * n);
        lvl[src] = 1;
        q[t++] = src;
        for (int h = 0; h < t && !lvl[sink]; h++) {
            int cur = q[h];
            for (auto& e : adj[cur]) {
                if (lvl[e.nxt] == 0 && e.res > 0) {
                    lvl[e.nxt] = lvl[cur] + 1;
                    q[t++] = e.nxt;
                }
            }
        }
        return lvl[sink] != 0;
    }
    flow_t block_flow(int cur, int sink, flow_t cur_flow) {
        if (cur == sink) return cur_flow;
        for (int& i = start[cur]; i < adj[cur].size(); i++) {
            auto& e = adj[cur][i];
            if (e.res > 0 && lvl[e.nxt] == lvl[cur] + 1) {
                if (flow_t res = block_flow(e.nxt, sink, min(e.res, cur_flow))) {
                    e.res -= res;
                    adj[e.nxt][e.inv].res += res;
                    return res;
                }
            }
        }
        return 0;
    }
    flow_t solve(int src, int sink) {
        q.resize(n);
        lvl.resize(n);
        start.resize(n);
        flow_t ans = 0;
        while (assign_level(src, sink)) {
            memset(&start[0], 0, sizeof(start[0]) * n);
```

```
            while (flow_t flow = block_flow(src, sink, numeric_limits<flow_t>::max()))
                ans += flow;
        }
        return ans;
    }
};
```

5.2 Hopcroft-Karp

```
struct hopcroft_karp{
    //0based-index
    int nx, ny;
    vector<int> X;//X[x] = 집합의 X 원소가 x 어떤에 y 대응되어있는가 ?
    vector<int> Y;//Y[y] = 집합의 Y 원소가 y 어떤에 x 대응되어있는가 ?
    vector<vector<int>> adj; // X -> Y 형태의이분그래프 , 를X 로Y 대응한다.
    vector<int> dist;
    hopcroft_karp(int nx, int ny) : nx(nx), ny(ny){ //집합 와X 의Y 크기
        X.resize(nx);
        Y.resize(ny);
        adj.resize(nx);
        dist.resize(nx);
    }
    void add_edge(int x, int y){
        adj[x].push_back(y);
    }

    void assign_level(){
        queue<int> q;
        for(int i=0;i<nx;++i){
            if (X[i] == -1){
                dist[i] = 0;
                q.push(i);
            } else dist[i] = -1;
        }
        while(!q.empty()){
            int now = q.front(); q.pop();
            for (int nxt: adj[now]){
                if (Y[nxt] != -1 && dist[Y[nxt]] == -1){
                    dist[Y[nxt]] = dist[now] + 1;
                    q.push(Y[nxt]);
                }
            }
        }
    }
    bool find_match(int now){
        for (int nxt: adj[now]){
            if (Y[nxt] == -1 || dist[Y[nxt]] == dist[now] + 1 && find_match(Y[nxt])){
                Y[nxt] = now;
                X[now] = nxt;
                return true;
            }
        }
        return false;
    }
    int operator()(){
        X.assign(nx, -1);
        Y.assign(ny, -1);
        int ret = 0;
        int flow = 0;
        do{
            assign_level();
            flow = 0;
            for(int i=0;i<nx;++i)
                flow += X[i] == -1 && find_match(i);
            ret += flow;
        }while(flow);
        return ret;
    }
};
```

6 DP

6.1 Convex Hull Trick

```
typedef long long ll;
struct convexhull_trick {
    //일단 binary 와search linear 둘다구현해둠
    struct line {
        ll a, b;
```

```
        pair<ll, ll> start_x; // first/second
        ll y(ll x) { return a * x + b; }
        static bool compare_x(line &l, const line &l1, const line &l2) {
            // intersect_x(l, l1) < intersect_x(l, l2)
            ll t1 = (l1.b - l.b) * (l.a - l2.a);
            ll t2 = (l2.b - l.b) * (l.a - l1.a);
            bool op = (l.a - l2.a < 0) ^ (l.a - l1.a < 0);
            return op ? t1 > t2 : t1 < t2;
        }
        bool not_start(ll x) { // start_x > x
            if (start_x.second < 0)
                return start_x.first < x * start_x.second;
            return start_x.first > x * start_x.second;
        }
    };
    vector<line> stk;
    int qptr;
    convexhull_trick(int capacity = 100010) : qptr(0) {
        stk.reserve(capacity);
    }
    void init(int capacity = 100010) {
        stk.clear();
        stk.reserve(capacity);
        qptr = 0;
    }
    void push_line(line l) {
        while (stk.size() > 1) {
            line &l1 = stk[stk.size() - 1];
            line &l2 = stk[stk.size() - 2];
            if (line::compare_x(l, l2, l1)) break;
            stk.pop_back();
        }
        if (qptr >= stk.size())
            qptr = stk.size() - 1;
        if (!stk.empty()) l.start_x = make_pair(stk.back().b - l.b, l.a - stk.back().a);
        else l.start_x = make_pair(0LL, 0LL);
        stk.push_back(l);
    }
    void push_line(ll a, ll b) { push_line({a, b}); }
    ll query_bs(ll x) {
        int lf = 0, rg = stk.size() - 1;
        while (lf < rg) {
            int mid = (lf + rg + 1) / 2;
            if (stk[mid].not_start(x)) rg = mid - 1;
            else lf = mid;
        }
        return stk[lf].y(x);
    }
    ll query_linear(ll x) { // 값이x 증가하는순서대로들어와야함
        while (qptr + 1 < stk.size() && !stk[qptr + 1].not_start(x)) ++qptr;
        return stk[qptr].y(x);
    }
};
```

7 Geometry

7.1 CCW(CounterClockWise)

```
struct P {
    int x, y;
    int s() { return this->x + this->y; }
};

struct L {
    L() {}
    L(const P &a, const P &b) : a(a), b(b) {}
    P a, b;
};

int ccw(const P &a, const P &b, const P &c) {
    int k = (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a.y);
    if (k > 0) return 1;
    if (k) return -1;
    return 0;
}

int insc(L p, L q) {
    if (!ccw(p.a, p.b, q.a) && !ccw(p.a, p.b, q.b)) {
        if (p.a.s() > p.b.s()) swap(p.a, p.b);
        if (q.a.s() > q.b.s()) swap(q.a, q.b);
```

```
        int a = p.a.s(), b = p.b.s(), c = q.a.s(), d = q.b.s();
        if (b < c || d < a) return 0;
        if (b == c || a == d) return 1;
        return -1;
    }
    if (ccw(p.a, p.b, q.a) == ccw(p.a, p.b, q.b) ||
        ccw(q.a, q.b, p.a) == ccw(q.a, q.b, p.b))
        return 0;
    return 1;
}
```