

Team Note of (교내)우승후보

overnap

Compiled on October 8, 2022

Contents

1 Data Structures For Range Query	2	5 String	9
1.1 Segment Tree w/ Lazy Propagation	2	5.1 Knuth-Moris-Pratt	9
1.2 Sparse Table	2	5.2 Rabin-Karp	9
1.3 Merge Sort Tree	2	5.3 Manacher	10
1.4 Binray Search In Segment Tree	3	5.4 Suffix Array and LCP Array	10
1.5 Persistence Segment Tree	3	5.5 Aho-Corasick	11
2 Graph	4	6 Offline Query	11
2.1 BipartiteMatching	4	6.1 Mo's	11
2.2 Max Flow	4	6.2 Parallel Binary Search	12
2.3 Min Cost Max Flow	4	7 DP Optimization	12
2.4 Dinic's	5	7.1 Convex Hull Trick w/ Stack	12
2.5 Strongly Connected Component	6	7.2 Convex Hull Trick w/ Li-Chao Tree	13
2.6 Biconnected Component	6	7.3 Knuth Optimization	13
2.7 Lowest Common Ancestor	6	7.4 Slope Trick	13
2.8 Heavy-Light Decomposition	6	8 Number Theory	14
3 Geometry	7	8.1 Modular Operator	14
3.1 Counter Clockwise	7	8.2 Modular Inverse in $\mathcal{O}(N)$	14
3.2 Line intersection	7	8.3 Extended Euclidean	14
3.3 Graham Scan	7	8.4 Miller-Rabin	14
3.4 Monotone Chain	7	8.5 Chinese Remainder Theorem	15
3.5 Rotating Calipers	8	8.6 Pollard Rho	15
4 Fast Fourier Transform	8	9 ETC	15
4.1 Fast Fourier Transform	8	9.1 Ternary Search	15
4.2 Number Theoretic Transform	9	9.2 Randomized Meldable Heap	15
4.3 Fast Walsh Hadamard Transform	9	9.3 Berlekamp-Massey	16
		9.4 Splay Tree w/ Rotate	18

9.5	Useful Stuff	18
9.6	Template	18
9.7	제출하기 전 생각해볼 것	19
9.8	자주 쓰이는 문제 접근법	19

1 Data Structures For Range Query

1.1 Segment Tree w/ Lazy Propagation

Usage: update(1, 0, n-1, l, r, v)

Time Complexity: $\mathcal{O}(\log N)$

```
struct lazySeg {
    vector<ll> tree, lazy;
    void push(int n, int s, int e) {
        tree[n] += lazy[n] * (e - s + 1);
        if (s != e) {
            lazy[n*2] += lazy[n];
            lazy[n*2+1] += lazy[n];
        }
        lazy[n] = 0;
    }
    void update(int n, int s, int e, int l, int r, int v) {
        push(n, s, e);
        if (e < l || r < s)
            return;
        if (l <= s && e <= r) {
            lazy[n] += v;
            push(n, s, e);
        } else {
            int m = (s + e) / 2;
            update(n*2, s, m, l, r, v);
            update(n*2+1, m+1, e, l, r, v);
            tree[n] = tree[n*2] + tree[n*2+1];
        }
    }
    ll query(int n, int s, int e, int l, int r) {
        push(n, s, e);
        if (e < l || r < s)
            return 0;
```

```
        if (l <= s && e <= r)
            return tree[n];
        int m = (s + e) / 2;
        return query(n*2, s, m, l, r) + query(n*2+1, m+1, e, l, r);
    }
};
```

1.2 Sparse Table

Usage: RMQ l r: min(lift[l][len], lift[r-(1<<len)+1][len])

Time Complexity: $\mathcal{O}(N) - \mathcal{O}(1)$

```
int k = ceil(log2(n));
vector<vector<int>> lift(n, vector<int>(k));
for (int i=0; i<n; ++i)
    lift[i][0] = lcp[i];
for (int i=1; i<k; ++i) {
    for (int j=0; j<=n-(1<<i); ++j)
        lift[j][i] = min(lift[j][i-1], lift[j+(1<<(i-1))][i-1]);
}
vector<int> bits(n+1);
for (int i=2; i<=n; ++i) {
    bits[i] = bits[i-1];
    while (1 << bits[i] < i)
        bits[i]++;
    bits[i]--;
}
```

1.3 Merge Sort Tree

Time Complexity: $\mathcal{O}(N \log N) - \mathcal{O}(\log^2 N)$

```
struct mst {
    int n;
    vector<vector<int>> tree;
    void init(vector<int> &arr) {
        n = 1 << (int)ceil(log2(arr.size()));
        tree.resize(n*2);
        for (int i=0; i<arr.size(); ++i)
            tree[n+i].push_back(arr[i]);
        for (int i=n-1; i>0; --i) {
```

```

    tree[i].resize(tree[i*2].size() + tree[i*2+1].size());
    merge(tree[i*2].begin(), tree[i*2].end(),
          tree[i*2+1].begin(), tree[i*2+1].end(), tree[i].begin());
}
}
int sum(int l, int r, int k) {
    int ret = 0;
    for (l+=n, r+=n; l<=r; l/=2, r/=2) {
        if (l%2)
            ret += upper_bound(tree[l].begin(), tree[l].end(), k) -
tree[l].begin(), l++;
        if (r%2 == 0)
            ret += upper_bound(tree[r].begin(), tree[r].end(), k) -
tree[r].begin(), r--;
    }
    return ret;
}
};

```

1.4 Binray Search In Segment Tree

Time Complexity: $\mathcal{O}(\log N)$

```

int query(int x) {
    int acc, i;
    for (acc=0, i=1; i<n; i++) {
        if (acc + tree[i*2] < x) {
            acc += tree[i*2];
            i = i*2+1;
        } else
            i = i*2;
    }
    return i - n;
}

```

1.5 Persistence Segment Tree

Time Complexity: $\mathcal{O}(\log^2 N)$

```

int node_cnt, cp, n, m, cnt, root[MN+3];
struct data{

```

```

    int x,y;
}point[M];
struct pst{
    int l,r,v;
};
vector<pst> tree(MN*30);
inline bool cmp(const data a, const data b){
    return a.y<b.y;
}
void mk_tree(){
    int i;
    root[0] = 1;
    node_cnt = MN<<1;
    for(i=1; i<MN; i++){
        tree[i].l = i<<1;
        tree[i].r = i<<1|1;
    }
}
void update(int s, int e, int now, int idx){
    tree[now].v++;
    if(s!=e){
        int m = (s+e)/2;
        int L = tree[now].l, R = tree[now].r;
        if(idx<=m){
            tree[now].l = node_cnt;
            tree[node_cnt++] = tree[L];
            update(s,m,tree[now].l,idx);
        }
        else{
            tree[now].r = node_cnt;
            tree[node_cnt++] = tree[R];
            update(m+1,e,tree[now].r, idx);
        }
    }
}
int find(int s, int e, int l, int r, int p_node, int n_node){
    if(l<=s&&e<=r) return tree[n_node].v-tree[p_node].v;
    if(r<s||l>e) return 0;
    int m = s+e>>1;
    int P = p_node, N = n_node;

```

```

    return fnd(s,m,l,r,tree[P].l,
tree[N].l)+fnd(m+1,e,l,r,tree[P].r, tree[N].r);
}
void Reset(){
    int i;
    for(i=1;i<=n;i++)
        point[i].x = point[i].y = 0;
    for(i=1;i<=MN;i++)
        root[i] = tree[i].l = tree[i].r = tree[i].v = 0;
}

```

2 Graph

2.1 BipartiteMatching

Usage: Run dfs for all left nodes. The count of return value true equal to count of max possible matches.

Time Complexity: $\mathcal{O}(VE)$

```

vector<int> from(n, -1);
vector<bool> visited(n, false);
bool dfs(vector<vector<int>>& adjList, vector<bool>& visited,
vector<int>& from, int curNode) {
    for (int nextNode : adjList[curNode]) {
        if (from[nextNode] == -1) {
            from[nextNode] = curNode;
            return true;
        }
    }
    for (int nextNode : adjList[curNode]) {
        if (visited[nextNode]) continue;
        visited[nextNode] = true;
        if (dfs(adjList, visited, from, from[nextNode])) {
            from[nextNode] = curNode;
            return true;
        }
    }
    return false;
}

```

2.2 Max Flow

Time Complexity: $\mathcal{O}(VE^2)$

```

while (true) {
    vector<int> prev(n, -1);
    queue<int> q;
    q.push(0);
    while (!q.empty() && prev[n-1] == -1) {
        const int x = q.front();
        q.pop();

        for (int next : e[x]) {
            if (cap[x][next] - flow[x][next] > 0 && prev[next] ==
-1) {
                prev[next] = x;
                q.push(next);
            }
        }
    }
    if (prev[n-1] == -1)
        break;
    int bot = 1e9+7;
    for (int i=n-1; i!=0; i=prev[i])
        bot = min(bot, cap[prev[i]][i] - flow[prev[i]][i]);
    for (int i=n-1; i!=0; i=prev[i]) {
        flow[prev[i]][i] += bot;
        flow[i][prev[i]] -= bot;
    }
}

```

2.3 Min Cost Max Flow

Time Complexity: $\mathcal{O}(VEf)$

```

void mcmf(){
    int cp = 0;
    while(cp<2){
        int prev[MN],dist[MN],inq[MN]={0};
        queue <int> Q;
        fill(prev, prev+MN, -1);
    }
}

```

```

fill(dist, dist+MN, INF);
dist[S] = 0; inq[S] = 1;
Q.push(S);
while(!Q.empty()){
    int cur= Q.front();
    Q.pop();
    inq[cur] = 0;
    for(int nxt: adj[cur]){
        if(cap[cur][nxt]-flow[cur][nxt]>0&&dist[nxt]>dist[cur]+cst[cur][nxt]){
            dist[nxt] = dist[cur] + cst[cur][nxt];
            prev[nxt] = cur;
            if(!inq[nxt]){
                Q.push(nxt);
                inq[nxt] = 1;
            }
        }
    }
}
if(prev[E]==-1) break;
int tmp = INF;
for(int i=E;i!=S;i=prev[i])
    tmp = min(tmp, cap[prev[i]][i]-flow[prev[i]][i]);

for(int i=E;i!=S;i=prev[i]){
    ans += tmp * cst[prev[i]][i];
    flow[prev[i]][i] += tmp;
    flow[i][prev[i]] -= tmp;
}
cp++;
}
}

```

2.4 Dinic's

Time Complexity: $\mathcal{O}(V^2E)$

```

int din_bfs(){ //level graph
    fill(level, level+MN, -1);
    level[S] = 0;
    queue<int> Q;
    Q.push(S);

```

```

while(!Q.empty()){
    int cur = Q.front();
    Q.pop();
    for(int nxt: adj[cur]){
        if(level[nxt]==-1&&cap[cur][nxt]-flow[cur][nxt]>0){
            level[nxt] = level[cur]+1;
            Q.push(nxt);
        }
    }
    return level[E] != -1;
}

int din_dfs(int cur, int des, int fl){ //flow
    if(cur==des) return fl;
    for(int &i=work[cur]; i<adj[cur].size(); i++){
        int nxt = adj[cur][i];
        if(level[nxt]==level[cur]+1&&cap[cur][nxt]-flow[cur][nxt]>0){
            int df = din_dfs(nxt, des,
min(cap[cur][nxt]-flow[cur][nxt], fl));
            if(df>0){
                flow[cur][nxt] += df;
                flow[nxt][cur] -= df;
                return df;
            }
        }
    }
    return 0;
}

void dinic(){
    int tot=0;
    while(din_bfs()){
        fill(work, work+MN, 0);
        while(1){
            int fl = din_dfs(S,E,INF);
            if(!fl) break;
            tot += fl;
        }
    }
    ans = tot;
}

```

2.5 Strongly Connected Component

Time Complexity: $\mathcal{O}(N)$

```
int idx = 0, scnt = 0;
vector<int> scc(n, -1), vis(n, -1), st;
function<int (int)> dfs = [&] (int x) {
    int ret = vis[x] = idx++;
    st.push_back(x);
    for (int next : e[x]) {
        if (vis[next] == -1)
            ret = min(ret, dfs(next));
        else if (scc[next] == -1)
            ret = min(ret, vis[next]);
    }
    if (ret == vis[x]) {
        while (!st.empty()) {
            const int t = st.back();
            st.pop_back();
            scc[t] = scnt;
            if (t == x)
                break;
        }
        scnt++;
    }
    return ret;
};
```

2.6 Biconnected Component

Time Complexity: $\mathcal{O}(N)$

```
int idx = 0;
vector<int> vis(n, -1);
vector<pii> st;
vector<vector<pii>> bcc;
vector<bool> cut(n); // articulation point
function<int (int, int)> dfs = [&] (int x, int p) {
    int ret = vis[x] = idx++;
    int child = 0;
    for (int next : e[x]) {
```

```
        if (next == p)
            continue;
        if (vis[next] < vis[x])
            st.emplace_back(x, next);
        if (vis[next] != -1)
            ret = min(ret, vis[next]);
        else {
            int res = dfs(next, x);
            ret = min(ret, res);
            child++;
            if (vis[x] <= res) {
                if (p != -1)
                    cut[x] = true;
                bcc.emplace_back();
                while (st.back() != pii{x, next}) {
                    bcc.back().push_back(st.back());
                    st.pop_back();
                }
                bcc.back().push_back(st.back());
                st.pop_back();
            } // vis[x] < res to find bridges
        }
    }
    if (p == -1 && child > 1)
        cut[x] = true;
    return ret;
};
```

2.7 Lowest Common Ancestor

Usage: Query with the sparse table

Time Complexity: $\mathcal{O}(N \log N) - \mathcal{O}(\log N)$

```
for (int i=1; i<16; ++i) {
    for (int j=0; j<n; ++j)
        par[j][i] = par[par[j][i-1]][i-1];
}
```

2.8 Heavy-Light Decomposition

Usage: Query with the ETT number and it's root node

Time Complexity: $\mathcal{O}(N) - \mathcal{O}(\log N)$

```
vector<int> par(n), ett(n), root(n), depth(n), sz(n);
function<void (int)> dfs1 = [&] (int x) {
    sz[x] = 1;
    for (int &next : e[x]) {
        if (next == par[x])
            continue;
        depth[next] = depth[x]+1;
        par[next] = x;
        dfs1(next);
        sz[x] += sz[next];
        if (e[x][0] == par[x] || sz[e[x][0]] < sz[next])
            swap(e[x][0], next);
    }
};
int idx = 1;
function<void (int)> dfs2 = [&] (int x) {
    ett[x] = idx++;
    for (int next : e[x]) {
        if (next == par[x])
            continue;
        root[next] = next == e[x][0] ? root[x] : next;
        dfs2(next);
    }
};
```

3 Geometry

3.1 Counter Clockwise

Usage: It returns $\{-1, 0, 1\}$ - the ccw of $b - a$ and $c - b$

Time Complexity: $\mathcal{O}(1)$

```
auto ccw = [] (const pii &a, const pii &b, const pii &c) {
    pii x = { b.first - a.first, b.second - a.second };
    pii y = { c.first - b.first, c.second - b.second };
    ll ret = 1LL * x.first * y.second - 1LL * x.second * y.first;
    return ret == 0 ? 0 : (ret > 0 ? 1 : -1);
};
```

3.2 Line intersection

Usage: Check the intersection of (x_1, x_2) and (y_1, y_2) . It requires an additional condition when they are parallel

Time Complexity: $\mathcal{O}(1)$

```
ccw(x1, x2, y1) != ccw(x1, x1, y2) && ccw(y1, y2, x1) != ccw(y1, y2, x2)
```

3.3 Graham Scan

Time Complexity: $\mathcal{O}(N \log N)$

```
struct point {
    int x, y, p, q;
    point() { x = y = p = q = 0; }
    bool operator < (const point& other) {
        if (1LL * other.p * q != 1LL * p * other.q)
            return 1LL * other.p * q < 1LL * p * other.q;
        else if (y != other.y)
            return y < other.y;
        else
            return x < other.x;
    }
};
swap(points[0], *min_element(points.begin(), points.end()));
for (int i=1; i<points.size(); ++i) {
    points[i].p = points[i].x - points[0].x;
    points[i].q = points[i].y - points[0].y;
}
sort(points.begin()+1, points.end());
vector<int> hull;
for (int i=0; i<points.size(); ++i) {
    while (hull.size() >= 2 && ccw(points[hull[hull.size()-2]],
    points[hull.back()], points[i]) < 1)
        hull.pop_back();
    hull.push_back(i);
}
```

3.4 Monotone Chain

Usage: Get the upper and lower hull of the convex hull

Time Complexity: $\mathcal{O}(N \log N)$

```
pair<vector<pii>, vector<pii>> getConvexHull(vector<pii> pt){
    sort(pt.begin(), pt.end());
    vector<pii> uh, dh;
    int un=0, dn=0; // for easy coding
    for (auto &tmp : pt) {
        while(un >= 2 && ccw(uh[un-2], uh[un-1], tmp))
            uh.pop_back(), --un;
        uh.push_back(tmp); ++un;
    }
    reverse(pt.begin(), pt.end());
    for (auto &tmp : pt) {
        while(dn >= 2 && ccw(dh[dn-2], dh[dn-1], tmp))
            dh.pop_back(), --dn;
        dh.push_back(tmp); ++dn;
    }
    return {uh, dh};
} // ref: https://namnamseo.tistory.com
```

3.5 Rotating Calipers

Usage: Get the maximum distance of the convex hull

Time Complexity: $\mathcal{O}(N)$

```
auto ccw4 = [&] (point& a1, point& a2, point& b1, point& b2) {
    return 1LL * (a2.x - a1.x) * (b2.y - b1.y) > 1LL * (a2.y - a1.y)
    * (b2.x - b1.x);
};
auto dist = [] (point& a, point& b) {
    return 1LL * (a.x - b.x) * (a.x - b.x) + 1LL * (a.y - b.y) *
    (a.y - b.y);
};
ll maxi = 0;
for (int i=0, j=1; i<hull.size(); i++) {
    maxi = max(maxi, dist(hull[i], hull[j]));
    if (j < hull.size()-1 && ccw4(hull[i], hull[i+1], hull[j],
    hull[j+1]))
        j++;
    else
```

```
        j++;
    }
}
```

4 Fast Fourier Transform

4.1 Fast Fourier Transform

Usage: FFT and multiply polynomials

Time Complexity: $\mathcal{O}(N \log N)$

```
using cd = complex<double>;
void fft(vector<cd> &f, cd w) {
    int n = f.size();
    if (n == 1)
        return;
    vector<cd> odd(n/2), even(n/2);
    for (int i=0; i<n; ++i)
        (i%2 ? odd : even)[i/2] = f[i];
    fft(odd, w*w);
    fft(even, w*w);
    cd x(1, 0);
    for (int i=0; i<n/2; ++i) {
        f[i] = even[i] + x * odd[i];
        f[i+n/2] = even[i] - x * odd[i];
        x *= w; // get through power to better accuracy
    }
}
vector<cd> mult(vector<cd> a, vector<cd> b) {
    int n;
    for (n=1; n<a.size() || n<b.size(); n*=2);
    n *= 2;
    vector<cd> ret(n);
    a.resize(n);
    b.resize(n);
    static constexpr double PI = 3.1415926535897932384;
    cd w(cos(PI*2/n), sin(PI*2/n));
    fft(a, w);
    fft(b, w);
    for (int i=0; i<n; ++i)
        ret[i] = a[i] * b[i];
}
```



```
fft(ret, cd(1, 0)/w);
for (int i=0; i<n; ++i) {
    ret[i] /= cd(n, 0);
    ret[i] = cd(round(ret[i].real()), round(ret[i].imag()));
}
return ret;
}
```

4.2 Number Theoretic Transform

Usage: FFT with integer - to get better accuracy

Time Complexity: $\mathcal{O}(N \log N)$

```
// w is the root of mod e.g. 3/998244353 and 5/1012924417
void ntt(vector<ll> &f, const ll w, const ll mod) {
    const int n = f.size();
    if (n == 1)
        return;
    vector<ll> odd(n/2), even(n/2);
    for (int i=0; i<n; ++i)
        (i&1 ? odd : even)[i/2] = f[i];
    ntt(odd, w*w%mod, mod);
    ntt(even, w*w%mod, mod);
    ll x = 1;
    for (int i=0; i<n/2; ++i) {
        f[i] = (even[i] + x * odd[i] % mod) % mod;
        f[i+n/2] = (even[i] - x * odd[i] % mod + mod) % mod;
        x = x*w%mod;
    }
}
```

4.3 Fast Walsh Hadamard Transform

Usage: XOR convolution

Time Complexity: $\mathcal{O}(N \log N)$

```
void fwht(vector<ll> &f) {
    const int n = f.size();
    if (n == 1)
        return;
    vector<ll> odd(n/2), even(n/2);
```

```
for (int i=0; i<n; ++i)
    (i&1 ? odd : even)[i/2] = f[i];
    fwht(odd);
    fwht(even);
    for (int i=0; i<n/2; ++i) {
        f[i*2] = even[i] + odd[i];
        f[i*2+1] = even[i] - odd[i];
    }
}
```

5 String

5.1 Knuth-Morris-Pratt

Time Complexity: $\mathcal{O}(N)$

```
vector<int> fail(m);
for (int i=1, j=0; i<m; ++i) {
    while (j > 0 && p[i] != p[j])
        j = fail[j-1];
    if (p[i] == p[j])
        fail[i] = ++j;
}
vector<int> ans;
for (int i=0, j=0; i<n; ++i) {
    while (j > 0 && t[i] != p[j])
        j = fail[j-1];
    if (t[i] == p[j]) {
        if (j == m-1) {
            ans.push_back(i-j);
            j = fail[j];
        } else
            j++;
    }
}
```

5.2 Rabin-Karp

Usage: The Rabin fingerprint for const-length hashing

Time Complexity: $\mathcal{O}(N)$

```

ull hash, p;
vector<ull> ht;
for (int i=0; i<=l-mid; ++i) {
    if (i == 0) {
        hash = s[0];
        p = 1;
        for (int j=1; j<mid; ++j) {
            hash = hash * pi + s[j];
            p = p * pi; // pi is the prime e.g. 13
        }
    } else
        hash = (hash - p * s[i-1]) * pi + s[i+mid-1];
    ht.push_back(hash);
}

```

5.3 Manacher

Usage: Longest radius of palindrome substring

Time Complexity: $\mathcal{O}(N)$

```

vector<int> man(m);
int r = 0, p = 0;
for (int i=0; i<m; ++i) {
    if (i <= r)
        man[i] = min(man[p*2 - i], r - i);
    while (i-man[i] > 0 && i+man[i] < m-1 && v[i-man[i]-1] ==
v[i+man[i]+1])
        man[i]++;
    if (r < i + man[i]) {
        r = i + man[i];
        p = i;
    }
}

```

5.4 Suffix Array and LCP Array

Time Complexity: $\mathcal{O}(N \log N) - \mathcal{O}(N)$

```

const int m = max(255, n)+1;
vector<int> sa(n), ord(n*2), nord(n*2);
for (int i=0; i<n; ++i) {

```

```

    sa[i] = i;
    ord[i] = s[i];
}
for (int d=1; d<n; d*=2) {
    auto cmp = [&] (int i, int j) {
        if (ord[i] == ord[j])
            return ord[i+d] < ord[j+d];
        return ord[i] < ord[j];
    };
    vector<int> cnt(m), tmp(n);
    for (int i=0; i<n; ++i)
        cnt[ord[i+d]]++;
    for (int i=0; i+1<m; ++i)
        cnt[i+1] += cnt[i];
    for (int i=n-1; i>=0; --i)
        tmp[--cnt[ord[i+d]]] = i;
    fill(cnt.begin(), cnt.end(), 0);
    for (int i=0; i<n; ++i)
        cnt[ord[i]]++;
    for (int i=0; i+1<m; ++i)
        cnt[i+1] += cnt[i];
    for (int i=n-1; i>=0; --i)
        sa[--cnt[ord[tmp[i]]]] = tmp[i];
    nord[sa[0]] = 1;
    for (int i=1; i<n; ++i)
        nord[sa[i]] = nord[sa[i-1]] + cmp(sa[i-1], sa[i]);
    swap(ord, nord);
}
vector<int> inv(n), lcp(n);
for (int i=0; i<n; ++i)
    inv[sa[i]] = i;
for (int i=0, k=0; i<n; ++i) {
    if (inv[i] == 0)
        continue;
    for (int j=sa[inv[i]-1]; s[i+k]==s[j+k]; ++k);
    lcp[inv[i]] = k ? k-- : 0;
}

```

5.5 Aho-Corasick

Time Complexity: $\mathcal{O}(N + \sum M)$

```
struct trie {
    array<trie *, 3> go;
    trie *fail;
    int output, idx;
    trie() {
        fill(go.begin(), go.end(), nullptr);
        fail = nullptr;
        output = idx = 0;
    }
    ~trie() {
        for (auto &x : go)
            delete x;
    }
    void insert(const string &input, int i) {
        if (i == input.size())
            output++;
        else {
            const int x = input[i] - 'A';
            if (!go[x])
                go[x] = new trie();
            go[x]->insert(input, i+1);
        }
    }
};

queue<trie*> q; // make fail links; requires root->insert before
root->fail = root;
q.push(root);
while (!q.empty()) {
    trie *curr = q.front();
    q.pop();
    for (int i=0; i<26; ++i) {
        trie *next = curr->go[i];
        if (!next)
            continue;
        if (curr == root)
            next->fail = root;
        else {
            trie *dest = curr->fail;
            while (dest != root && !dest->go[i])
                dest = dest->fail;
            if (dest->go[i])
                dest = dest->go[i];
            next->fail = dest;
        }
        if (next->fail->output)
            next->output = true;
        q.push(next);
    }
}

trie *curr = root; // start query
bool found = false;
for (char c : s) {
    c -= 'a';
    while (curr != root && !curr->go[c])
        curr = curr->fail;
    if (curr->go[c])
        curr = curr->go[c];
    if (curr->output) {
        found = true;
        break;
    }
}
```

```
trie *dest = curr->fail;
while (dest != root && !dest->go[i])
    dest = dest->fail;
if (dest->go[i])
    dest = dest->go[i];
next->fail = dest;
}
if (next->fail->output)
    next->output = true;
q.push(next);
}
}

trie *curr = root; // start query
bool found = false;
for (char c : s) {
    c -= 'a';
    while (curr != root && !curr->go[c])
        curr = curr->fail;
    if (curr->go[c])
        curr = curr->go[c];
    if (curr->output) {
        found = true;
        break;
    }
}
```

6 Offline Query

6.1 Mo's

Usage: sort by $(L\sqrt{L}, R)$

Time Complexity: $\mathcal{O}(Q \log Q + N\sqrt{N})$

```
sort(q.begin(), q.end(), [&] (const auto &a, const auto &b) {
    if (get<0>(a)/rt != get<0>(b)/rt)
        return get<0>(a)/rt < get<0>(b)/rt;
    return get<1>(a) < get<1>(b);
});

int res = 0, s = get<0>(q[0]), e = get<1>(q[0]);
vector<int> count(1e6), result(m);
```

```

for (int i=s; i<=e; ++i)
    res += count[a[i]]++ == 0;
result[get<2>(q[0])] = res;
for (int i=1; i<m; ++i) {
    while (get<0>(q[i]) < s)
        res += count[a[--s]]++ == 0;
    while (get<1>(q[i]) > e)
        res += count[a[++e]]++ == 0;
    while (get<0>(q[i]) > s)
        res -= --count[a[s++]] == 0;
    while (get<1>(q[i]) < e)
        res -= --count[a[e--]] == 0;
    result[get<2>(q[i])] = res;
}

```

6.2 Parallel Binary Search

Time Complexity: $\mathcal{O}(N \log N)$

```

vector<int> lo(q, -1), hi(q, m), answer(q);
while (true) {
    int fin = 0;
    vector<vector<int>> mids(m);
    for (int i=0; i<q; ++i) {
        if (lo[i] + 1 < hi[i])
            mids[(lo[i] + hi[i])/2].push_back(i);
        else
            fin++;
    }
    if (fin == q)
        break;
    ufind uf;
    uf.init(n+1);
    for (int i=0; i<m; ++i) {
        const auto &[eig, a, b] = edges[i];
        uf.merge(a, b);
        for (int x : mids[i]) {
            if (uf.find(qs[x].first) == uf.find(qs[x].second)) {
                hi[x] = i;
                answer[x] = -uf.par[uf.find(qs[x].first)];
            } else

```

```

        lo[x] = i;
    }
}

```

7 DP Optimization

7.1 Convex Hull Trick w/ Stack

Usage: $dp[i] = \min(dp[j] + b[j] * a[i]), b[j] \geq b[j+1]$

Time Complexity: $\mathcal{O}(N \log N) - \mathcal{O}(N)$ where $a[i] \leq a[i+1]$

```

struct lin {
    ll a, b;
    double s;
    ll f(ll x) { return a*x + b; }
};
inline double cross(const lin &x, const lin &y) {
    return 1.0 * (x.b - y.b) / (y.a - x.a);
}
vector<ll> dp(n);
vector<lin> st;
for (int i=1; i<n; ++i) {
    lin curr = { b[i-1], dp[i-1], 0 };
    while (!st.empty()) {
        curr.s = cross(st.back(), curr);
        if (st.back().s < curr.s)
            break;
        st.pop_back();
    }
    st.push_back(curr);
    int x = -1;
    for (int y = st.size(); y > 0; y /= 2) {
        while (x+y < st.size() && st[x+y].s < a[i])
            x += y;
    }
    dp[i] = s[x].f(a[i]);
}
while (x+1 < st.size() && st[x+1].s < a[i]) ++x; //  $\mathcal{O}(N)$  case

```

7.2 Convex Hull Trick w/ Li-Chao Tree

Usage: update(l, r, 0, { a, b })

Time Complexity: $\mathcal{O}(N \log N)$

```
static constexpr ll INF = 2e18;
struct lin {
    ll a, b;
    ll f(ll x) { return a*x + b; }
};
struct lichao {
    struct node {
        int l, r;
        lin line;
    };
    vector<node> tree;
    void init() { tree.push_back({-1, -1, { 0, -INF }}); }
    void update(ll s, ll e, int n, const lin &line) {
        lin hi = tree[n].line;
        lin lo = line;
        if (hi.f(s) < lo.f(s))
            swap(lo, hi);
        if (hi.f(e) >= lo.f(e)) {
            tree[n].line = hi;
            return;
        }
        const ll m = s + e >> 1;
        if (hi.f(m) > lo.f(m)) {
            tree[n].line = hi;
            if (tree[n].r == -1) {
                tree[n].r = tree.size();
                tree.push_back({-1, -1, { 0, -INF }});
            }
            update(m+1, e, tree[n].r, lo);
        } else {
            tree[n].line = lo;
            if (tree[n].l == -1) {
                tree[n].l = tree.size();
                tree.push_back({-1, -1, { 0, -INF }});
            }
            update(s, m, tree[n].l, hi);
        }
    }
};
```

```
    }
}
ll query(ll s, ll e, int n, ll x) {
    if (n == -1)
        return -INF;
    const ll m = s + e >> 1;
    if (x <= m)
        return max(tree[n].line.f(x), query(s, m, tree[n].l, x));
    else
        return max(tree[n].line.f(x), query(m+1, e, tree[n].r, x));
}
};
```

7.3 Knuth Optimization

Usage: dp[i] = min(dp[i][k] + dp[k][j]) + c[i][j], Monge, Monotonic

Time Complexity: $\mathcal{O}(N^2)$

```
vector<vector<int>> dp(n, vector<int>(n)), opt(n, vector<int>(n));
for (int i=0; i<n; ++i)
    opt[i][i] = i;
for (int j=1; j<n; ++j) {
    for (int s=0; s<n-j; ++s) {
        int e = s+j;
        dp[s][e] = 1e9+7;
        for (int o=opt[s][e-1]; o<min(opt[s+1][e]+1, e); ++o) {
            if (dp[s][e] > dp[s][o] + dp[o+1][e]) {
                dp[s][e] = dp[s][o] + dp[o+1][e];
                opt[s][e] = o;
            }
        }
        dp[s][e] += sum[e+1] - sum[s];
    }
}
```

7.4 Slope Trick

Usage: Use priority queue, convex condition

Time Complexity: $\mathcal{O}(N \log N)$

```
pq.push(A[0]);
for (int i=1; i<N; ++i) {
```

```

    pq.push(A[i] - i);
    pq.push(A[i] - i);
    pq.pop();
    A[i] = pq.top();
}

```

8 Number Theory

8.1 Modular Operator

Usage: For Fermat's little theorem and Pollard rho

Time Complexity: $\mathcal{O}(\log N)$

```

using ull = unsigned long long;
ull modmul(ull a, ull b, ull n) {
    return ((unsigned __int128)a * b) % n;
}
ull modmul(ull a, ull b, ull n) { // if __int128 isn't available
    if (b == 0)
        return 0;
    if (b == 1)
        return a;
    ull t = modmul(a, b/2, n);
    t = (t+t)%n;
    if (b % 2)
        t = (t+a)%n;
    return t;
}
ull modpow(ull a, ull d, ull n) {
    if (d == 0)
        return 1;
    ull r = modpow(a, d/2, n);
    r = modmul(r, r, n);
    if (d % 2)
        r = modmul(r, a, n);
    return r;
}
ull gcd(ull a, ull b) {
    return b ? gcd(b, a%b) : a;
}

```

8.2 Modular Inverse in $\mathcal{O}(N)$

Usage: Get inverse of factorial

Time Complexity: $\mathcal{O}(N) - \mathcal{O}(1)$

```

const int mod = 1e9+7;
vector<int> fact(n+1), inv(n+1), factinv(n+1);
fact[0] = fact[1] = inv[1] = factinv[0] = factinv[1] = 1;
for (int i=2; i<=n; ++i) {
    fact[i] = 1LL * fact[i-1] * i % mod;
    inv[i] = mod - 1LL * mod/i * inv[mod%i] % mod;
    factinv[i] = 1LL * factinv[i-1] * inv[i] % mod;
}

```

8.3 Extended Euclidean

Usage: get a and b as arguments and return the solution (x, y) of equation $ax + by = \gcd(a, b)$.

Time Complexity: $\mathcal{O}(\log a + \log b)$

```

pair<ll, ll> extGCD(ll a, ll b) {
    if (b != 0) {
        auto tmp = extGCD(b, a % b);
        return {tmp.second, tmp.first - (a / b) * tmp.second};
    } else return {1ll, 0ll};
}

```

8.4 Miller-Rabin

Usage: Fast prime test for big integers

Time Complexity: $\mathcal{O}(k \log N)$

```

bool is_prime(ull n) {
    const ull as[7] = {2, 325, 9375, 28178, 450775, 9780504,
1795265022};
    // const ull as[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
37}; // easier to remember
    auto miller_rabin = [] (ull n, ull a) {
        ull d = n-1, temp;
        while (d % 2 == 0) {
            d /= 2;
            temp = modpow(a, d, n);

```

```

        if (temp == n-1)
            return true;
    }
    return temp == 1;
};
for (ull a : as) {
    if (a >= n)
        break;
    if (!miller_rabin(n, a))
        return false;
}
return true;
}

```

8.5 Chinese Remainder Theorem

Usage: Solution for the system of linear congruence

Time Complexity: $\mathcal{O}(\log N)$

```

w1 = modpow(mod2, mod1-2, mod1);
w2 = modpow(mod1, mod2-2, mod2);
ll ans = ((__int128)mod2 * w1 * f1[i] + (__int128)mod1 * w2 * f2[i])
% (mod1*mod2);

```

8.6 Pollard Rho

Usage: Factoring large numbers fast

Time Complexity: $\mathcal{O}(N^{1/4})$

```

void pollard_rho(ull n, vector<ull> &factors) {
    if (n == 1)
        return;
    if (n % 2 == 0) {
        factors.push_back(2);
        pollard_rho(n/2, factors);
        return;
    }
    if (is_prime(n)) {
        factors.push_back(n);
        return;
    }
}

```

```

ull x, y, c = 1, g = 1;
auto f = [&] (ull x) { return (modmul(x, x, n) + c) % n; };
y = x = 2;
while (g == 1 || g == n) {
    if (g == n) {
        c = rand() % 123;
        y = x = rand() % (n-2) + 2;
    }
    x = f(x);
    y = f(f(y));
    g = gcd(n, y>x ? y-x : x-y);
}
pollard_rho(g, factors);
pollard_rho(n / g, factors);
}

```

9 ETC

9.1 Ternary Search

Time Complexity: $\mathcal{O}(\log N)$

```

int l = 0, r = T;
while (l+2 < r) {
    int p = (2*l+r)/3, q = (l+2*r)/3;
    ll pd = calc(p, N, stars), qd = calc(q, N, stars);
    if (pd <= qd)
        r = q-1;
    else
        l = p+1;
} // check l..r

```

9.2 Randomized Meldable Heap

Usage: Min-heap H is declared as `Heap<T> H`. You can use `push`, `size`, `empty`, `top`, `pop` as `std::priority_queue`. Use `H.meld(G)` to meld contents from G to H.

Time Complexity: $\mathcal{O}(\log n)$

```

namespace Meldable {
mt19937 gen(0x94949);

```

```

template<typename T>
struct Node {
    Node *l, *r;
    T v;
    Node(T x): l(0), r(0), v(x){}
};

template<typename T>
Node<T>* Meld(Node<T>* A, Node<T>* B) {
    if(!A) return B; if(!B) return A;
    if(B->v < A->v) swap(A, B);
    if(gen() & 1) A->l = Meld(A->l, B);
    else A->r = Meld(A->r, B);
    return A;
}

template<typename T>
struct Heap {
    Node<T> *r; int s;
    Heap(): r(0), s(0){}
    void push(T x) {
        r = Meld(new Node<T>(x), r);
        ++s;
    }
    int size(){ return s; }
    bool empty(){ return s == 0; }
    T top(){ return r->v; }
    void pop() {
        Node<T>* p = r;
        r = Meld(r->l, r->r);
        delete p;
        --s;
    }
    void Meld(Heap x) {
        s += x->s;
        r = Meld(r, x->r);
    }
};

```

9.3 Berlekamp-Massey

Usage: get_nth({1, 1, 2, 3, 5}, n)

```

const int mod = 998244353;
using lint = long long;
lint ipow(lint x, lint p){
    lint ret = 1, piv = x;
    while(p){
        if(p & 1) ret = ret * piv % mod;
        piv = piv * piv % mod;
        p >>= 1;
    }
    return ret;
}

vector<int> berlekamp_massey(vector<int> x){
    vector<int> ls, cur;
    int lf, ld;
    for(int i=0; i<x.size(); i++){
        lint t = 0;
        for(int j=0; j<cur.size(); j++){
            t = (t + 1ll * x[i-j-1] * cur[j]) % mod;
        }
        if((t - x[i]) % mod == 0) continue;
        if(cur.empty()){
            cur.resize(i+1);
            lf = i;
            ld = (t - x[i]) % mod;
            continue;
        }
        lint k = -(x[i] - t) * ipow(ld, mod - 2) % mod;
        vector<int> c(i-lf-1);
        c.push_back(k);
        for(auto &j : ls) c.push_back(-j * k % mod);
        if(c.size() < cur.size()) c.resize(cur.size());
        for(int j=0; j<cur.size(); j++){
            c[j] = (c[j] + cur[j]) % mod;
        }
        if(i-lf+(int)ls.size()>=(int)cur.size()){
            tie(ls, lf, ld) = make_tuple(cur, i, (t - x[i]) % mod);
        }
    }
}

```



```

    cur = c;
}
for(auto &i : cur) i = (i % mod + mod) % mod;
return cur;
}

int get_nth(vector<int> rec, vector<int> dp, lint n){
    int m = rec.size();
    vector<int> s(m), t(m);
    s[0] = 1;
    if(m != 1) t[1] = 1;
    else t[0] = rec[0];
    auto mul = [&rec](vector<int> v, vector<int> w){
        int m = v.size();
        vector<int> t(2 * m);
        for(int j=0; j<m; j++){
            for(int k=0; k<m; k++){
                t[j+k] += 11l * v[j] * w[k] % mod;
                if(t[j+k] >= mod) t[j+k] -= mod;
            }
        }
        for(int j=2*m-1; j>=m; j--){
            for(int k=1; k<=m; k++){
                t[j-k] += 11l * t[j] * rec[k-1] % mod;
                if(t[j-k] >= mod) t[j-k] -= mod;
            }
        }
        t.resize(m);
        return t;
    };
    while(n){
        if(n & 1) s = mul(s, t);
        t = mul(t, t);
        n >>= 1;
    }
    lint ret = 0;
    for(int i=0; i<m; i++) ret += 11l * s[i] * dp[i] % mod;
    return ret % mod;
}

int guess_nth_term(vector<int> x, lint n){
    if(n < x.size()) return x[n];

```

```

    vector<int> v = berlekamp_massey(x);
    if(v.empty()) return 0;
    return get_nth(v, x, n);
}

struct elem{int x, y, v;}; // A_(x, y) <- v, 0-based. no duplicate
please..
vector<int> get_min_poly(int n, vector<elem> M){
    // smallest poly P such that A^i = sum_{j < i} {A^j \times P_j}
    vector<int> rnd1, rnd2;
    mt19937 rng(0x14004);
    auto randint = [&rng](int lb, int ub){
        return uniform_int_distribution<int>(lb, ub)(rng);
    };
    for(int i=0; i<n; i++){
        rnd1.push_back(randint(1, mod - 1));
        rnd2.push_back(randint(1, mod - 1));
    }
    vector<int> gobs;
    for(int i=0; i<2*n+2; i++){
        int tmp = 0;
        for(int j=0; j<n; j++){
            tmp += 11l * rnd2[j] * rnd1[j] % mod;
            if(tmp >= mod) tmp -= mod;
        }
        gobs.push_back(tmp);
    }
    vector<int> nxt(n);
    for(auto &i : M){
        nxt[i.x] += 11l * i.v * rnd1[i.y] % mod;
        if(nxt[i.x] >= mod) nxt[i.x] -= mod;
    }
    rnd1 = nxt;
}

auto sol = berlekamp_massey(gobs);
reverse(sol.begin(), sol.end());
return sol;
}

lint det(int n, vector<elem> M){
    vector<int> rnd;
    mt19937 rng(0x14004);
    auto randint = [&rng](int lb, int ub){

```

```

    return uniform_int_distribution<int>(lb, ub)(rng);
};
for(int i=0; i<n; i++) rnd.push_back(randint(1, mod - 1));
for(auto &i : M){
    i.v = 111 * i.v * rnd[i.y] % mod;
}
auto sol = get_min_poly(n, M)[0];
if(n % 2 == 0) sol = mod - sol;
for(auto &i : rnd) sol = 111 * sol * ipow(i, mod - 2) % mod;
return sol;
}

```

9.4 Splay Tree w/ Rotate

9.5 Useful Stuff

- Catalan Number
1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900
 $C_n = \text{binomial}(n * 2, n) / (n + 1);$
- 길이가 $2n$ 인 올바른 괄호 수식의 수
- $n + 1$ 개의 리프를 가진 풀 바이너리 트리의 수
- $n + 2$ 각형을 n 개의 삼각형으로 나누는 방법의 수
- Burnside's Lemma
경우의 수를 세는데, 특정 transform operation(회전, 반사, ...) 해서 같은 경우들은 하나로 친다. 전체 경우의 수는? 각 operation마다 이 operation을 했을 때 변하지 않는 경우의 수를 센다 (단, "아무것도 하지 않는다" 라는 operation도 있어야 함!)
전체 경우의 수를 더한 후, operation의 수로 나눈다. (답이 맞다면 항상 나누어 떨어져야 한다)
- 알고리즘 게임
- Nim Game의 해법 : 각 더미의 돌의 개수를 모두 XOR했을 때 0 이 아니면 첫번째, 0 이면 두번째 플레이어가 승리.
- Grundy Number : 어떤 상황의 Grundy Number는, 가능한 다음 상황들의 Grundy Number를 모두 모은 다음, 그 집합에 포함 되지 않는 가장 작은 수가 현재 state의 Grundy Number가 된다. 만약 다음 state가 독립된 여러개의 state 들로 나뉠 경우, 각각의 state의 Grundy Number의 XOR 합을 생각한다.
- Subtraction Game : 한 번에 k 개까지의 돌만 가져갈 수 있는 경우, 각 더미의 돌의 개수를 $k + 1$ 로 나눈 나머지를 XOR 합하여 판단한다.
- Index-k Nim : 한 번에 최대 k 개의 더미를 골라 각각의 더미에서 아무렇게나 돌을 제거할 수 있을 때, 각 binary digit에 대하여 합을 $k + 1$ 로 나눈 나머지를

계산한다. 만약 이 나머지가 모든 digit에 대하여 0이라면 두번째, 하나라도 0이 아니라면 첫번째 플레이어가 승리.

- Pick's Theorem
격자점으로 구성된 simple polygon이 주어짐. I 는 polygon 내부의 격자점 수, B 는 polygon 선분 위 격자점 수, A 는 polygon의 넓이라고 할 때, 다음과 같은 식이 성립한다. $A = I + B/2 - 1$
- 가장 가까운 두 점 : 분할정복으로 가까운 6개의 점만 확인
- 홀의 결혼 정리 : 이분그래프(L-R)에서, 모든 L을 매칭하는 필요충분 조건 = L에서 임의의 부분집합 S를 골랐을 때, 반드시 $(S \text{의 크기}) \leq (S \text{와 연결되어있는 모든 R의 크기})$ 이다.
- 오일러 정리 : $V - E + f(\text{면})$ 가 일정
- 소수 : 10 007 , 10 009 , 10 111 , 31 567 , 70 001 , 1 000 003 , 1 000 033 , 4 000 037 , 99 999 989 , 999 999 937 , 1 000 000 007 , 1 000 000 009 , 9 999 999 967 , 99 999 999 977
- 소수 개수 : (1e5 이하 : 9592), (1e7 이하 : 664 579) , (1e9 이하 : 50 847 534)
- 10^{15} 이하의 정수 범위의 나눗셈 한번은 오차가 없다.
- N 의 약수의 개수 = $O(N^{1/3})$, N 의 약수의 합 = $O(N \log \log N)$
- $\phi(mn) = \phi(m)\phi(n), \phi(pr^n) = pr^n - pr^{n-1}, a^{\phi(n)} \equiv 1 \pmod{n}$ if coprime
- Euler's phi $\phi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$
- Lucas' Theorem $\binom{m}{n} \equiv \prod \binom{m_i}{n_i} \pmod{p}$ m_i, n_i 는 p^i 의 계수

9.6 Template

```

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

using namespace std;
using ll = long long;
using pii = pair<int, int>;

int main() {
    ios::sync_with_stdio(false);

```

```

cin.tie(nullptr);
int t; cin >> t;
while (t--)
    solve();
return 0;
}

// precision
cout.precision(16);
cout << fixed;

// gcc bit operator
__builtin_popcount(bits) // popcountll for ll
__builtin_clz(bits) // left
__builtin_ctz(bits) // right

// random number generator
random_device rd;
mt19937 mt;
uniform_int_distribution<> half(0, 1);
cout << half(mt);

// 128MB = int * 33,554,432

```

9.7 제출하기 전 생각해볼 것

- min, max 입력 테스트
- 나눗셈이 들어가면 0과 음수 확인
- 곱셈이 들어가면 오버플로우 확인
- mod 필요하면 모든 중간과정과 끝 확인
- 출력 정밀도 확인
- FastIO

9.8 자주 쓰이는 문제 접근법

- 비슷한 문제를 풀어본 적이 있던가?
- 단순한 방법에서 시작할 수 있을까? (brute force)

- 내가 문제를 푸는 과정을 수식화할 수 있을까? (예제를 직접 해결해보면서)
- 문제를 단순화할 수 없을까?
- 그림으로 그려볼 수 있을까?
- 수식으로 표현할 수 있을까?
- 문제를 분해할 수 있을까?
- 뒤에서부터 생각해서 문제를 풀 수 있을까?
- 순서를 강제할 수 있을까?
- 특정 형태의 답만을 고려할 수 있을까? (정규화)
- 특수 조건을 꼭 활용
- 여사건으로 생각하기
- Convexity 파악하고 최적화
- 게임이론
- 경우 나누어 생각
- 해법에서 역순으로
- 딱 맞는 시간복잡도에 집착하지 말자
- $N \leq 8 \rightarrow N!$
- $10 \leq N \leq 20 \rightarrow 2^N$
- $30 \leq N \leq 40 \rightarrow 2^{N/2}$
- $N \leq 50 \rightarrow N^4$
- $400 \leq N \leq 500 \rightarrow N^3$
- $N \leq 1000 \rightarrow N^2, N^4(\text{mitm})$
- $N \leq 3000 \rightarrow N^2$
- $N \leq 1e5 \rightarrow N \log N, N \log^2 N$
- $N \geq 1e6 \rightarrow \text{sqrt}$ 금지
- 문제에 의미있는 작은 상수 이용
- 최후의 수단 버킷질, 상수 커팅