

Team Note of 세상에 나쁜 알고리즘은 없다

hamuim, overnap, pani

Compiled on August 2, 2024

Contents

1 Data Structures

1.1 Sparse Table	2
1.2 Merge Sort Tree	2
1.3 Persistence Segment Tree	2
1.4 Segment Tree Beats	3
1.5 Fenwick RMQ	4
1.6 Randomized Meldable Heap	4
1.7 Link-Cut Tree	5

2 Graph & Flow

2.1 Hopcroft-Karp & König's	8
2.2 Min Cost Max Flow	9
2.3 Dinic's	9
2.4 Strongly Connected Component	10
2.5 Biconnected Component	10
2.6 Lowest Common Ancestor	11
2.7 Heavy-Light Decomposition	11
2.8 Centroid Decomposition	11

3 Geometry

3.1 Counter Clockwise	12
3.2 Line intersection	12
3.3 Graham Scan	12
3.4 Monotone Chain	12
3.5 Rotating Calipers	12
3.6 Bulldozer Trick	13

4 Fast Fourier Transform

4.1 Fast Fourier Transform	13
4.2 Number Theoretic Transform	14
4.3 Fast Walsh Hadamard Transform	14

5 String

5.1 Knuth-Morris-Pratt	14
5.2 Rabin-Karp	15
5.3 Manacher	15
5.4 Suffix Array and LCP Array	15
5.5 Aho-Corasick	16

6 Offline Query

6.1 Mo's	17
6.2 Parallel Binary Search	17

7 DP Optimization

7.1 Convex Hull Trick w/ Stack	17
7.2 Convex Hull Trick w/ Li-Chao Tree	18
7.3 Divide and Conquer Optimization	18
7.4 Monotone Queue Optimization	19
7.5 Aliens Trick	19
7.6 Knuth Optimization	19
7.7 Slope Trick	19
7.8 Sum Over Subsets	20

8 Number Theory

8.1 Modular Operator	20
8.2 Modular Inverse in $\mathcal{O}(N)$	20
8.3 Extended Euclidean	20

8.4	Miller-Rabin	20
8.5	Chinese Remainder Theorem	21
8.6	Pollard Rho	21
9	ETC	21
9.1	Gaussian Elimination	21
9.2	Useful Stuff	22
9.3	Template	22
9.4	자주 쓰이는 문제 접근법	23
9.5	DP 최적화 접근	23
9.6	Fast I/O	23

1 Data Structures

1.1 Sparse Table

Usage: $\text{RMQ } l:r: \min(\text{lift}[l][\text{len}], \text{lift}[r-(1 \ll \text{len})+1][\text{len}])$

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]--;
}
```

1.2 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;
}
};
```

1.3 Persistence Segment Tree

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

```
struct pst {
    struct node {
        int cnt = 0;
        array<int, 2> go{};
    };
    vector<node> tree;
    vector<int> roots;
    pst() {
        roots.push_back(1);
        tree.resize(1 << 18);
        for (int i = 1; i < (1 << 17); ++i) {
```

```

    tree[i].go[0] = i * 2;
    tree[i].go[1] = i * 2 + 1;
}
}
int insert(int x, int prev) {
    int curr = tree.size();
    roots.push_back(curr);
    tree.emplace_back();
    for (int i = 16; i >= 0; --i) {
        const int next = (x >> i) & 1;
        tree[curr].go[next] = tree.size();
        tree.emplace_back();
        tree[curr].go[!next] = tree[prev].go[!next];
        tree[curr].cnt = tree[prev].cnt + 1;
        curr = tree[curr].go[next];
        prev = tree[prev].go[next];
    }
    tree[curr].cnt = tree[prev].cnt + 1;
    return roots.back();
}
int query(int u, int v, int lca, int lca_par, int k) {
    int ret = 0;
    for (int i = 16; i >= 0; --i) {
        const int cnt = tree[tree[u].go[0]].cnt +
            tree[tree[v].go[0]].cnt -
            tree[tree[lca].go[0]].cnt -
            tree[tree[lca_par].go[0]].cnt;

        if (cnt >= k) {
            u = tree[u].go[0];
            v = tree[v].go[0];
            lca = tree[lca].go[0];
            lca_par = tree[lca_par].go[0];
        } else {
            k -= cnt;
            u = tree[u].go[1];
            v = tree[v].go[1];
            lca = tree[lca].go[1];
            lca_par = tree[lca_par].go[1];
            ret += 1 << i;
        }
    }
}

```

```

    }
    return ret;
}
};

```

1.4 Segment Tree Beats

Usage: Note the potential function

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

```

struct seg {
    vector<node> tree;
    void push(int x, int s, int e) {
        tree[x].x += tree[x].l;
        tree[x].o += tree[x].l;
        tree[x].a += tree[x].l;
        if (s != e) {
            tree[x*2].l += tree[x].l;
            tree[x*2+1].l += tree[x].l;
        }
        tree[x].l = 0;
    }
    void init(int x, int s, int e, const vector<int> &a) {
        if (s == e)
            tree[x].x = tree[x].o = tree[x].a = a[s];
        else {
            const int m = (s+e) / 2;
            init(x*2, s, m, a);
            init(x*2+1, m+1, e, a);
            tree[x] = tree[x*2] + tree[x*2+1];
        }
    }
    void off(int x, int s, int e, int l, int r, int v) {
        push(x, s, e);
        if (e < l || r < s || (tree[x].o & v) == 0)
            return;
        if (l <= s && e <= r && !(v & (tree[x].a ^ tree[x].o))) {
            tree[x].l -= v & tree[x].o;
            push(x, s, e);
        } else {
            const int m = (s+e) / 2;

```

```

    off(x*2, s, m, l, r, v);
    off(x*2+1, m+1, e, l, r, v);
    tree[x] = tree[x*2] + tree[x*2+1];
}
}
void on(int x, int s, int e, int l, int r, int v) {
    push(x, s, e);
    if (e < l || r < s || (tree[x].a & v) == v)
        return;
    if (l <= s && e <= r && !(v & (tree[x].a ^ tree[x].o))) {
        tree[x].l += v & ~tree[x].o;
        push(x, s, e);
    } else {
        const int m = (s+e) / 2;
        on(x*2, s, m, l, r, v);
        on(x*2+1, m+1, e, l, r, v);
        tree[x] = tree[x*2] + tree[x*2+1];
    }
}
int sum(int x, int s, int e, int l, int r) {
    push(x, s, e);
    if (e < l || r < s)
        return 0;
    if (l <= s && e <= r)
        return tree[x].x;
    const int m = (s+e) / 2;
    return max(sum(x*2, s, m, l, r), sum(x*2+1, m+1, e, l, r));
}
};

```

1.5 Fenwick RMQ

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

```

struct fenwick {
    static constexpr pii INF = {1e9 + 7, -(1e9 + 7)};
    vector<pii> tree1, tree2;
    const vector<int> &arr;
    static pii op(pii l, pii r) {
        return {min(l.first, r.first), max(l.second, r.second)};
    }
}

```

```

fenwick(const vector<int> &a) : arr(a) {
    const int n = a.size();
    tree1.resize(n + 1, INF);
    tree2.resize(n + 1, INF);
    for (int i = 0; i < n; ++i)
        update(i, a[i]);
}
void update(int x, int v) {
    for (int i = x + 1; i < tree1.size(); i += i & -i)
        tree1[i] = op(tree1[i], {v, v});
    for (int i = x + 1; i > 0; i -= i & -i)
        tree2[i] = op(tree2[i], {v, v});
}
pii query(int l, int r) {
    pii ret = INF;
    l++, r++;
    int i;
    for (i = r; i - (i & -i) >= l; i -= i & -i)
        ret = op(tree1[i], ret);
    for (i = l; i + (i & -i) <= r; i += i & -i)
        ret = op(tree2[i], ret);
    ret = op({arr[i - 1], arr[i - 1]}, ret);
    return ret;
}
};

```

1.6 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);
    }
};

```

1.7 Link-Cut Tree

```

struct Node {
    Node *l, *r, *p;
    bool flip;
    int sz;
    T now, sum, lz;
    Node() {
        l = r = p = nullptr;

```

```

        sz = 1;
        flip = false;
        now = sum = lz = 0;
    }
    bool IsLeft() const { return p && this == p->l; }
    bool IsRoot() const { return !p || (this != p->l && this != p->r); }
    friend int GetSize(const Node *x) { return x ? x->sz : 0; }
    friend T GetSum(const Node *x) { return x ? x->sum : 0; }
    void Rotate() {
        p->Push();
        Push();
        if (IsLeft())
            r && (r->p = p), p->l = r, r = p;
        else
            l && (l->p = p), p->r = l, l = p;
        if (!p->IsRoot())
            (p->IsLeft() ? p->p->l : p->p->r) = this;
        auto t = p;
        p = t->p;
        t->p = this;
        t->Update();
        Update();
    }
    void Update() {
        sz = 1 + GetSize(l) + GetSize(r);
        sum = now + GetSum(l) + GetSum(r);
    }
    void Update(const T &val) {
        now = val;
        Update();
    }
    void Push() {
        Update(now + lz);
        if (flip)
            swap(l, r);
        for (auto c : {l, r})
            if (c)
                c->flip ^= flip, c->lz += lz;
        lz = 0;

```

```

    flip = false;
}
};
Node *rt;
Node *Splay(Node *x, Node *g = nullptr) {
    for (g || (rt = x); x->p != g; x->Rotate()) {
        if (!x->p->IsRoot())
            x->p->p->Push();
        x->p->Push();
        x->Push();
        if (x->p->p != g)
            (x->IsLeft() ^ x->p->IsLeft() ? x : x->p)->Rotate();
    }
    x->Push();
    return x;
}
Node *Kth(int k) {
    for (auto x = rt;; x = x->r) {
        for (; x->Push(), x->l && x->l->sz > k; x = x->l)
            ;
        if (x->l)
            k -= x->l->sz;
        if (!k--)
            return Splay(x);
    }
}
Node *Gather(int s, int e) {
    auto t = Kth(e + 1);
    return Splay(t, Kth(s - 1))->l;
}
Node *Flip(int s, int e) {
    auto x = Gather(s, e);
    x->flip ^= 1;
    return x;
}
Node *Shift(int s, int e, int k) {
    if (k >= 0) { // shift to right
        k %= e - s + 1;
        if (k)
            Flip(s, e), Flip(s, s + k - 1), Flip(s + k, e);
    }
}

```

```

    } else { // shift to left
        k = -k;
        k %= e - s + 1;
        if (k)
            Flip(s, e), Flip(s, e - k), Flip(e - k + 1, e);
    }
    return Gather(s, e);
}
int Idx(Node *x) { return x->l->sz; }
////////// Link Cut Tree Start //////////
Node *Splay(Node *x) {
    for (; !x->IsRoot(); x->Rotate()) {
        if (!x->p->IsRoot())
            x->p->p->Push();
        x->p->Push();
        x->Push();
        if (!x->p->IsRoot())
            (x->IsLeft() ^ x->p->IsLeft() ? x : x->p)->Rotate();
    }
    x->Push();
    return x;
}
void Access(Node *x) {
    Splay(x);
    x->r = nullptr;
    x->Update();
    for (auto y = x; x->p; Splay(x))
        y = x->p, Splay(y), y->r = x, y->Update();
}
int GetDepth(Node *x) {
    Access(x);
    x->Push();
    return GetSize(x->l);
}
Node *GetRoot(Node *x) {
    Access(x);
    for (x->Push(); x->l; x->Push())
        x = x->l;
    return Splay(x);
}
}

```

```

Node *GetPar(Node *x) {
    Access(x);
    x->Push();
    if (!x->l)
        return nullptr;
    x = x->l;
    for (x->Push(); x->r; x->Push())
        x = x->r;
    return Splay(x);
}

void Link(Node *p, Node *c) {
    Access(c);
    Access(p);
    c->l = p;
    p->p = c;
    c->Update();
}

void Cut(Node *c) {
    Access(c);
    c->l->p = nullptr;
    c->l = nullptr;
    c->Update();
}

Node *GetLCA(Node *x, Node *y) {
    Access(x);
    Access(y);
    Splay(x);
    return x->p ? x->p : x;
}

Node *Ancestor(Node *x, int k) {
    k = GetDepth(x) - k;
    assert(k >= 0);
    for (;;) x->Push() {
        int s = GetSize(x->l);
        if (s == k)
            return Access(x), x;
        if (s < k)
            k -= s + 1, x = x->r;
        else
            x = x->l;
    }
}

```

```

    }
}

void MakeRoot(Node *x) {
    Access(x);
    Splay(x);
    x->flip ^= 1;
    x->Push();
}

bool IsConnect(Node *x, Node *y) { return GetRoot(x) == GetRoot(y); }

void PathUpdate(Node *x, Node *y, T val) {
    Node *root = GetRoot(x); // original root
    MakeRoot(x);
    Access(y); // make x to root, tie with y
    Splay(x);
    x->lz += val;
    x->Push();
    MakeRoot(root); // Revert
    // edge update without edge vertex...
    Node *lca = GetLCA(x, y);
    Access(lca);
    Splay(lca);
    lca->Push();
    lca->Update(lca->now - val);
}

T VertexQuery(Node *x, Node *y) {
    Node *l = GetLCA(x, y);
    T ret = l->now;
    Access(x);
    Splay(l);
    if (l->r)
        ret = ret + l->r->sum;
    Access(y);
    Splay(l);
    if (l->r)
        ret = ret + l->r->sum;
    return ret;
}

Node *GetQueryResultNode(Node *u, Node *v) {
    if (!IsConnect(u, v))

```

```

    return 0;
    MakeRoot(u);
    Access(v);
    auto ret = v->l;
    while (ret->mx != ret->now) {
        if (ret->l && ret->mx == ret->l->mx)
            ret = ret->l;
        else
            ret = ret->r;
    }
    Access(ret);
    return ret;
} // code from justicehui

```

2 Graph & Flow

2.1 Hopcroft-Karp & König's

Usage: Dinic's variant. Maximum Matching = Minimum Vertex Cover = S - Maximum Independence Set

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

```

while (true) {
    vector<int> level(sz, -1);
    queue<int> q;
    for (int x : 1) {
        if (match[x] == -1) {
            level[x] = 0;
            q.push(x);
        }
    }
    while (!q.empty()) {
        const int x = q.front();
        q.pop();
        for (int next : e[x]) {
            if (match[next] != -1 && level[match[next]] == -1) {
                level[match[next]] = level[x] + 1;
                q.push(match[next]);
            }
        }
    }
}

```

```

}
if (level.empty() || *max_element(level.begin(), level.end()) == -1)
    break;
function<bool(int)> dfs = [&](int x) {
    for (int next : e[x]) {
        if (match[next] == -1 ||
            (level[match[next]] == level[x] + 1 && dfs(match[next])))
        {
            match[next] = x;
            match[x] = next;
            return true;
        }
    }
    return false;
};
int total = 0;
for (int x : 1) if (level[x] == 0) total += dfs(x);
if (total == 0) break;
flow += total;
}
set<int> alt; // Konig
function<void(int, bool)> dfs = [&](int x, bool left) {
    if (alt.contains(x)) return;
    alt.insert(x);
    for (int next : e[x]) {
        if ((next != match[x]) && left) dfs(next, false);
        if ((next == match[x]) && !left) dfs(next, true);
    }
};
for (int x : 1) if (match[x] == -1) dfs(x, true);
int test = 0;
for (int i : 1) {
    if (alt.contains(i)) {
        auto &[y, x] = pos[i];
        s[y][x] = 'C';
    }
}
for (int i : r) {
    if (!alt.contains(i)) {

```



```

    auto &[y, x] = pos[i];
    s[y][x] = 'C';
}
}

```

2.2 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++;
}
}

```

2.3 Dinic's

Time Complexity: $\mathcal{O}(V^2E)$, $\mathcal{O}(\min(V^{2/3}E, E^{3/2}))$ on unit capacity

```

while (true) {
    vector<int> level(n * 2 + 2, -1);
    queue<int> q;
    level[st] = 0;
    q.push(st);
    while (!q.empty()) {
        const int x = q.front();
        q.pop();
        for (int next : e[x]) {
            if (level[next] == -1 && cap[x][next] - flow[x][next] > 0) {
                level[next] = level[x] + 1;
                q.push(next);
            }
        }
    }
    if (level[dt] == -1)
        break;
    vector<int> vis(n * 2 + 1);
    function<int(int, int)> dfs = [&](int x, int total) {
        if (x == dt)
            return total;
        for (int &i = vis[x]; i < e[x].size(); ++i) {
            const int next = e[x][i];
            if (level[next] == level[x] + 1 && cap[x][next] -
                flow[x][next] > 0) {
                const int res = dfs(next, min(total, cap[x][next] -
                    flow[x][next]));
                if (res > 0) {
                    flow[x][next] += res;
                    flow[next][x] -= res;
                    return res;
                }
            }

```

```

    }
  }
}
return 0;
};
while (true) {
  const int res = dfs(st, 1e9 + 7);
  if (res == 0)
    break;
  ans += res;
}
}
}

```

2.4 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;
};

```

2.5 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;
};

```

2.6 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];
}
```

2.7 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), rt(n), d(n), sz(n);
function<void (int)> dfs1 = [&] (int x) {
    sz[x] = 1;
    for (int &next : e[x]) {
        if (next == par[x]) continue;
        d[next] = d[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;
        rt[next] = next == e[x][0] ? rt[x] : next;
        dfs2(next);
    }
};
```

2.8 Centroid Decomposition

Usage: cent[x] is the parent in centroid tree

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

```
vector<int> sz(n);
vector<bool> fin(n);
function<int (int, int)> get_size = [&] (int x, int p) {
    sz[x] = 1;
    for (int next : e[x])
        if (!fin[next] && next != p) sz[x] += get_size(next, x);
    return sz[x];
};
function<int (int, int, int)> get_cent = [&] (int x, int p, int all)
{
    for (int next : e[x])
        if (!fin[next] && next != p && sz[next]*2 > all) return
            get_cent(next, x, all);
    return x;
};
vector<int> cent(n, -1);
function<void (int, int)> get_cent_tree = [&] (int x, int p) {
    get_size(x, p);
    x = get_cent(x, p, sz[x]);
    fin[x] = true;
    cent[x] = p;
    function<void (int, int, int, bool)> dfs = [&] (int x, int p,
        int d, bool test) {
        if (test) // update answer
        else // update state
            for (int next : e[x])
                if (!fin[next] && next != p) dfs(next, x, d, test);
    };
    for (int next : e[x]) {
        if (!fin[next]) {
            dfs(next, x, init, true);
            dfs(next, x, init+curr, false);
        }
    }
    for (int next : e[x])
        if (!fin[next] && next != p) get_cent_tree(next, x);
};
get_cent_tree(0, -1);
```

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
        i++;
}

```

3.6 Bulldozer Trick

Usage: Traverse the entire sorting state of 2D points

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

```

struct Line{
    ll i, j, dx, dy; // dx >= 0
    Line(int i, int j, const Point &pi, const Point &pj)
        : i(i), j(j), dx(pj.x-pi.x), dy(pj.y-pi.y) {}
    bool operator < (const Line &l) const {
        return make_tuple(dy*l.dx, i, j) < make_tuple(l.dy*dx, l.i,
            l.j);
    }
    bool operator == (const Line &l) const {
        return dy * l.dx == l.dy * dx;
    }
};
void Solve(){
    sort(A+1, A+N+1); iota(P+1, P+N+1, 1);
    vector<Line> V; V.reserve(N*(N-1)/2);
    for(int i=1; i<=N; i++) for(int j=i+1; j<=N; j++)
        V.emplace_back(i, j, A[i], A[j]);
    sort(V.begin(), V.end());
    for(int i=0, j=0; i<V.size(); i=j){
        while(j < V.size() && V[i] == V[j]) j++;
        for(int k=i; k<j; k++){

```

```

            int u = V[k].i, v = V[k].j; // point id, index -> Pos[id]
            swap(Pos[u], Pos[v]); swap(A[Pos[u]], A[Pos[v]]);
            if(Pos[u] > Pos[v]) swap(u, v);
            // @TODO
        }
    }
} // code from justicehui

```

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;
}
vector<cd> div(vector<cd> &f, int sz) {
    vector<cd> ret = {1 / f[0]};
    for (int i=1; i<sz; i*=2) {
        vector<cd> tmp(f.begin(), f.begin()+min((int)f.size(), i*2));
        tmp = mult(ret, tmp);
        tmp.resize(i * 2);
        for (cd &x : tmp) x = -x;
        tmp[0] += 2;
        ret = mult(ret, tmp);
        ret.resize(i * 2);
    }
    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;
    }
}
}

```


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; // 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 Divide and Conquer Optimization

Usage: `dp[t][i] = min(dp[t-1][j] + c[j][i])`, `c` is Monge

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

```

vector<vector<ll>> dp(n, vector<ll>(t));
function<void (int, int, int, int, int)> dnc = [&] (int l, int r,
int s, int e, int u) {
    if (l > r)
        return;
    const int mid = (l + r) / 2;
    int opt;
    for (int i=s; i<=min(e, mid); ++i) {
        ll x = sum[i][mid] + C;
        if (i && u)
            x += dp[i-1][u-1];
        if (x >= dp[mid][u]) {
            dp[mid][u] = x;
        }
    }
};

```

```

        opt = i;
    }
}
dnc(l, mid-1, s, opt, u);
dnc(mid+1, r, opt, e, u);
};
for (int i=0; i<t; ++i)
    dnc(0, n-1, 0, n-1, i);

```

7.4 Monotone Queue Optimization

Usage: $dp[i] = \min(dp[j] + c[j][i])$, c is Monge, find cross
Time Complexity: $\mathcal{O}(N \log N)$

```

auto cross = [&](ll p, ll q) {
    ll lo = min(p, q) - 1, hi = n + 1;
    while (lo + 1 < hi) {
        const ll mid = (lo + hi) / 2;
        if (f(p, mid) < f(q, mid)) lo = mid;
        else hi = mid;
    }
    return hi;
};
deque<pll> st;
for (int i = 1; i <= n; ++i) {
    pll curr{i - 1, 0};
    while (!st.empty() &&
        (curr.second = cross(st.back().first, i - 1)) <=
        st.back().second)
        st.pop_back();
    st.push_back(curr);
    while (st.size() > 1 && st[1].second <= i) st.pop_front();
    dp[i] = f(st[0].first, i);
}

```

7.5 Aliens Trick

Usage: $dp[t][i] = \min(dp[t-1][j] + c[j+1][i])$, c is Monge, find
 lambda w/ half bs
Time Complexity: $\mathcal{O}(N \log N)$

```

ll lo = 0, hi = 1e15;
while (lo + 1 < hi) {
    const ll mid = (lo + hi) / 2;
    auto [dp, cnt] = dec(mid); // the best DP[N][K] and its K value
    if (cnt < k) hi = mid;
    else lo = mid;
}
cout << (dec(lo).first - lo * k) / 2;

```

7.6 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];
    }
}

```

7.7 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();
}
```

7.8 Sum Over Subsets

Usage: `dp[mask] = sum(A[i]), i is in mask`

Time Complexity: $\mathcal{O}(N2^N)$

```
for (int i=0; i<(1<<n); i++)
    f[i] = a[i];
for (int j=0; j<n; j++)
    for(int i=0; i<(1<<n); i++)
        if (i & (1<<j)) f[i] += f[i ^ (1<<j)];
```

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 Gaussian Elimination

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

```

struct basis {
    const static int n = 30; // log2(1e9)
    array<int, n> data{};
    void insert(int x) {
        for (int i=0; i<n; ++i)
            if (data[i] && (x >> (n-1-i) & 1)) x ^= data[i];
        int y;
        for (y=0; y<n; ++y)
            if (!data[y] && (x >> (n-1-y) & 1)) break;
        if (y < n) {
            for (int i=0; i<n; ++i)
                if (data[i] >> (n-1-y) & 1) data[i] ^= x;
            data[y] = x;
        }
    }
    basis operator+(const basis &other) {
        basis ret{};
        for (int x : data) ret.insert(x);
    }
}

```

```

    for (int x : other.data) ret.insert(x);
    return ret;
}
};

```

9.2 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.
- 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 characteristic : $v - e + f$ (면, 외부 포함) = $1 + c$ (킴포넌트)
- 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
- 스케줄링에서 데드라인이 빠른 걸 쓰는게 이득. 늦은 스케줄이 안들어갈 때 가장 시간 소모가 큰 스케줄 1개를 제거하면 이득.

9.3 Template

```

// 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(rd());
uniform_int_distribution<> half(0, 1);
cout << half(mt);
// 128MB = int * 33,554,432
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c

```

```

    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
}
size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM =
        chrono::steady_clock::now().time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
}
};
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename K, typename V, typename Comp = less<K>>
using ordered_map =
    tree<K, V, Comp, rb_tree_tag,
        tree_order_statistics_node_update>;
template <typename K, typename Comp = less<K>> // less_equal for
multiset
using ordered_set = ordered_map<K, null_type, Comp>;

```

9.4 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 있을까?
- 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
- 게임이론 - 거울 전략 혹은 DP 연계
- 겁먹지 말고 경우 나누어 생각
- 해법에서 역순으로 가능한가?
- 딱 맞는 시간복잡도에 집착하지 말자
- 문제에 의미있는 작은 상수 이용
- 스몰투라지나 트라이 같은 트릭 생각
- 잘못된 방법으로 파고들지 말고 버리자

9.5 DP 최적화 접근

- $C[i, j] = A[i] * B[j]$ 이고 A, B가 단조증가, 단조감소이면 Monge
- l.r의 값들의 sum이나 min은 Monge
- 식 정리해서 일차(CHT) 혹은 비슷한(MQ) 함수를 발견, 구현 힘들면 Li-Chao
- $a \leq b \leq c \leq d$ 에서 $A[a, c] + A[b, d] \leq A[a, d] + A[b, c]$
- Monge 성질을 보이기 어려우면 N^2 나이브 짜서 opt의 단조성을 확인하고 찍맞
- 식이 간단하거나 변수가 독립적이면 DP 테이블을 세그 위에 올려서 해결
- 침착하게 점화식부터 세우고 Monge인지 판별
- Monge에 집착하지 말고 단조성이나 볼록성만 보여도 됨

9.6 Fast I/O

```

#pragma GCC optimize("O3")
#pragma GCC optimize("Ofast")
#pragma GCC optimize("unroll-loops")

inline int readChar();
template<class T = int> inline T readInt();
template<class T> inline void writeInt(T x, char end = 0);
inline void writeChar(int x);

```

```

inline void writeWord(const char *s);
static const int buf_size = 1 << 18;
inline int getChar(){
    #ifndef LOCAL
        static char buf[buf_size];
        static int len = 0, pos = 0;
        if(pos == len) pos = 0, len = fread(buf, 1, buf_size, stdin);
        if(pos == len) return -1;
        return buf[pos++];
    #endif
}
inline int readChar(){
    #ifndef LOCAL
        int c = getChar();
        while(c <= 32) c = getChar();
        return c;
    #else
        char c; cin >> c; return c;
    #endif
}
template <class T>
inline T readInt(){
    #ifndef LOCAL
        int s = 1, c = readChar();
        T x = 0;
        if(c == '-') s = -1, c = getChar();
        while('0' <= c && c <= '9') x = x * 10 + c - '0', c = getChar();
        return s == 1 ? x : -x;
    #else
        T x; cin >> x; return x;
    #endif
}
static int write_pos = 0;
static char write_buf[buf_size];
inline void writeChar(int x){
    if(write_pos == buf_size) fwrite(write_buf, 1, buf_size,
        stdout), write_pos = 0;
    write_buf[write_pos++] = x;
}
template <class T>

```

```

inline void writeInt(T x, char end){
    if(x < 0) writeChar('-'), x = -x;
    char s[24]; int n = 0;
    while(x || !n) s[n++] = '0' + x % 10, x /= 10;
    while(n--) writeChar(s[n]);
    if(end) writeChar(end);
}
inline void writeWord(const char *s){
    while(*s) writeChar(*s++);
}
struct Flusher{
    ~Flusher(){ if(write_pos) fwrite(write_buf, 1, write_pos,
        stdout), write_pos = 0; }
}flusher;

```