

fresh water (Hongik University)

Contents

0.1 템플릿	2	5 수학	12
1 자료구조	2	5.1 빠른 조합(nCr) + 나눗셈 역원	12
1.1 팬윅 트리 (BIT)	2	5.2 채(sieve) : 소수, 소인수분해, 오일러 피	13
1.2 세그먼트 트리	2	5.3 소수 찾기 - Fast (밀러-라빈)	13
1.3 레이지 세그먼트 트리	2	5.4 소인수분해 - Fast (풀라드-로)	14
1.4 금광 세그먼트 트리	3	5.5 FFT (고속 푸리에 변환)	14
1.5 유니온 파인드	3	5.6 중국인의 나머지 정리 + 확장 유clidean	15
1.6 BBST with k-th (Ordered Tree)	3	5.7 mint - 빠른 모듈러 연산	15
1.7 퍼시스턴트 세그먼트 트리	3		
2 그래프	4	6 문자열	16
2.1 SCC (타잔 알고리즘)	4	6.1 해싱	16
2.2 2-SAT	4	6.2 트라이 (문자열 전용 set)	16
2.3 BCC - 단절점, 단절선	5	6.3 KMP	16
2.4 SPFA (벨만 포드)	5	6.4 접미사 배열 (Suffix Array + LCP)	16
2.5 방향 그래프 사이클 판정	6	6.5 모든 팰린드롬 찾기 - Manacher	17
2.6 다익스트라 알고리즘	6	6.6 Z Algorithm	17
2.7 오프라인 동적 연결성 판정	6	6.7 아호코라식 (한 문자열 : 여러 패턴)	18
2.8 위상 정렬	7		
3 트리	7	7 DP	18
3.1 LCA (최소 공통 조상)	7	7.1 컨벡스 헬 트리 (Convex Hull Trick; CHT)	18
3.2 Heavy-light Decomposition	8	7.2 리차오 트리(Li Chao Tree)	19
3.3 Centroid Decomposition	8	7.3 벌레캠프(Berlekamp-Massey)	19
4 플로우	9	7.4 Monotone Queue (덱으로 최소값)	20
4.1 이분 매칭 - Easy	9		
4.2 이분 매칭 - Fast (호프크로프트 카프)	9	8 기타	20
4.3 최대 플로우 - Easy (에드몬드 카프)	10	8.1 기본 기타 라이브러리 (CCW)	20
4.4 최대 플로우 - Fast (디닉)	11	8.2 볼록 꺽질(컨벡스 헬)	23
4.5 MCMF (Minimum Cost Maximum Flow)	12	8.3 회전하는 캘리퍼스 - 볼록 꺽질에서 가장 먼 점	23
9 기타			
9.1 Mo's Algorithm + Sqrt Decomposition		9.1 Mo's Algorithm + Sqrt Decomposition	23
9.2 삼분 탐색		9.2 삼분 탐색	24
9.3 유용한 이론들		9.3 유용한 이론들	24
9.4 자주 쓰이는 테크닉들		9.4 자주 쓰이는 테크닉들	24

0.1 템플릿

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
#define xx first
#define yy second
#define all(v) (v).begin(), (v).end()
#define loop(e, v) for (auto& (e) : (v))
#define mem(v, e) memset(v, e, sizeof(v))
```

1 자료구조

1.1 팬윅 트리 (BIT)

Usage: 1-index인거 항상 조심!! $k\text{th}(k) = 1\text{-indexed } k\text{번째 원소}$

```
const int MAX;
int seg[MAX+1] = {};
int query(int i){
    int ret = 0;
    while(i > 0){ ret += seg[i]; i -= i & -i;}
    return ret;
}
void update(int i, int a){
    while(i <= MAX){ seg[i] += a; i += i & -i;}
}
int kth(int k){
    int ret = 0;
    for(int i=30; i>=0; --i){
        int pivot = ret + (1<<i);
        if(pivot <= MAX && seg[pivot] < k){
            k -= seg[pivot]; ret = pivot;
        }
    }
    return ret+1;
}
```

1.2 세그먼트 트리

Usage: 구간합 세그먼트 트리

```
ll seg[MAX*4];
ll query(int l, int r, int no, int nl, int nr){
    if(r < nl || nr < l) return 0;
    if(l <= nl && nr <= r) return seg[no];
    int mid = (nl + nr)/2;
    return query(l, r, no*2, nl, mid) + query(l, r, no*2+1, mid+1, nr);
}
ll update(int i, ll v, int no, int nl, int nr){
    if(i<nl || nr<i) return seg[no];
    if(nl == i && nr == i) return seg[no] = v; // += v;
    int mid = (nl + nr)/2;
    return seg[no] = update(i,v,no*2,nl,mid) +
        update(i,v,no*2+1,mid+1,nr);
}

1.3 레이지 세그먼트 트리

Usage: 구간합 세그먼트 트리

const int MAX;
ll seg[MAX*4], lazy[MAX*4];
void prop(int no, int nl, int nr){
    if(lazy[no] != 0){
        seg[no] += (nr - nl +1)*lazy[no];
        if(nl != nr){
            lazy[no*2] += lazy[no];
            lazy[no*2+1] += lazy[no];
        }
        lazy[no] = 0;
    }
}
ll update(int l, int r, ll v, int no, int nl, int nr){
    prop(no, nl, nr);
    if(r<nl || nr<l) return seg[no];
    if(l <= nl && nr <= r){
        lazy[no] = v; prop(no, nl, nr); return seg[no];
    }
    int m = (nl+nr)/2;
    return seg[no] = update(l,r,v,no*2,nl,m) +
        update(l,r,v,no*2+1,m+1,nr);
}
```

```

11 query(int l, int r, int no, int nl, int nr){
    prop(no, nl, nr);
    if(r<nl || nr<l) return 0;
    if(l <= nl && nr <= r) return seg[no];
    int m = (nl + nr)/2;
    return query(l,r,no*2,nl,m) + query(l,r,no*2+1,m+1,nr);
}

```

1.4 금광 세그먼트 트리

Usage: 구간합의 최대를 구하는 세그먼트 트리

```

inline ll safe_add (const ll& a, const ll& b) {
    if (a > -LINF && b > -LINF) return a+b;
    if (a > -LINF) return a;
    if (b > -LINF) return b;
    return -LINF;
}

struct Node {
    ll lval = -LINF, rval = -LINF, val = -LINF, all = 0;
    Node operator+ (const Node& n) const { return {safe_add(lval,n.lval),
        safe_add(rval,n.rval), safe_add(val,n.val), safe_add(all,n.all)}; }
} seg[MAX*4];
Node func(Node a, Node b) {
    return {
        max({a.lval, safe_add(a.all,b.lval)}),
        max({safe_add(a.rval,b.all), b.rval}),
        max({a.val, b.val, safe_add(a.rval,b.lval)}),
        max({safe_add(a.all,b.all)})
    };
}

```

1.5 유니온 파인드

Usage: parent[i] < 0 이면 해당 집합의 크기

```

const int MAX;
int parent[MAX];
void clear(){ memset(parent, -1, sizeof(parent));}
int find(int a){
    if(parent[a]<0) return a; return parent[a] = find(parent[a]);
}
void merge(int a, int b){
    a = find(a), b = find(b); if(a==b) return;
}

```

```

    if(parent[a] < parent[b]) swap(a,b);
    parent[b] += parent[a]; parent[a] = b;
}

```

1.6 BBST with k-th (Ordered Tree)

```

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/trie_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
// multiset 사용시 pair<원소값, cnt++>
int main(){
    ordered_set X;
    //insert(a) : a를 삽입한다
    X.insert(1); // 0번째 원소 = 1
    X.insert(2); // 1번째 원소 = 2
    X.insert(4); // 2번째 원소 = 4
    X.insert(8); // 3번째 원소 = 8
    X.insert(16); // 4번째 원소 = 16
    //iterator
    for(auto it = X.begin(); it != X.end(); ++it)
        cout << *it << ' ';
    cout << '\n'; // 1 2 4 8 16
    //find_by_order(a) : a번째 원소의 iterator를 반환한다.
    //단, a > X.size()면 X.end()를 반환한다.
    cout<<*X.find_by_order(1)<<endl; // 2
    cout<<*X.find_by_order(2)<<endl; // 4
    cout<<*X.find_by_order(4)<<endl; // 16
    cout<<(X.end()==X.find_by_order(6))<<endl; // true
    //order_of_key(a) : a보다 작은 원소의 개수를 반환한다.
    cout<<X.order_of_key(-5)<<endl; // 0
    cout<<X.order_of_key(1)<<endl; // 0
    cout<<X.order_of_key(3)<<endl; // 2
    cout<<X.order_of_key(4)<<endl; // 2
    cout<<X.order_of_key(400)<<endl; // 5
}

```

1.7 퍼시스턴트 세그먼트 트리

Usage: 공간 복잡도가 NlogN인 2차원 세그먼트 트리

```

const int MAX = 100100;
struct Node{

```

```

int v = 0; //반드시 항등원
Node *left = 0, *right = 0;
};

vector<Node*> Tree;
void init(){
    Tree.clear();
    Tree.push_back(new Node());
}

void make_tree(int i, int v){
    Tree.push_back(make_tree(Tree.back(), i, v, 0, MAX-1));
}

Node* make_tree(Node *prev, int i, int v, int nl, int nr){
    Node *now = new Node();
    if(i < nl || nr < i) return prev;
    if (nl == nr) {
        now->v = prev->v + v;
        return now;
    }
    int mid = (nl + nr) / 2;
    if (!prev->left)
        prev->left = new Node(), prev->right = new Node();
    now->left = make_tree(prev->left, i, v, nl, mid);
    now->right = make_tree(prev->right, i, v, mid+1, nr);
    now->v = now->left->v + now->right->v;
    return now;
}
int query(int root_i, int l, int r){
    return query(Tree[root_i], l, r, 0, MAX-1);
}
int query(Node *here, int l, int r, int nl, int nr){
    if(here == 0) return 0;
    if(nr < l || r < nl) return 0;
    if(l <= nl && nr <= r) return here->v;
    int mid = (nl + nr)/2;
    return query(here->left, l, r, nl, mid) +
           query(here->right, l, r, mid+1, nr);
}
for(int i=0; i<size_of_x; ++i) PST.make_tree(p.yy, 1);

```

2 그래프

2.1 SCC (타잔 알고리즘)

Usage: 같은 그룹끼리는 사이클이 항상 있고, 다른 그룹끼리는 사이클이 절대 없다. *scc*에 그룹이 위상정렬의 역순으로 정렬됨.

complexity: $\mathcal{O}(V + E)$

```

int vst[MAX], sccn[MAX], cnt1, cnt2;
vector<int> adj[MAX]; vector<vector<int>> scc;
stack<int> s;
int DFS(int here) {
    vst[here] = cnt1++;
    int ret = vst[here];
    s.push(here);
    for (int &there : adj[here]) {
        if (vst[there] == -1) ret = min(ret, DFS(there));
        else if (sccn[there] == -1) ret = min(ret, vst[there]);
    }
    if (ret == vst[here]) {
        scc.emplace_back(vector<int>());
        auto &V = scc.back();
        while (true) {
            int tmp = s.top(); s.pop();
            sccn[tmp] = cnt2; V.push_back(tmp);
            if (tmp == here) break;
        }
        ++cnt2;
    }
    return ret;
}
void make_SCC(int V) {
    memset(vst, -1, sizeof(vst));
    memset(sccn, -1, sizeof(sccn));
    cnt1 = 0, cnt2 = 0;
    for (int i = 0; i < V; ++i) if (vst[i] == -1) DFS(i);
}

```

2.2 2-SAT

Usage: $(X_1 \mid\mid Y_1) \&\& (X_2 \mid\mid Y_2) \&\& \dots$

complexity: $\mathcal{O}(V + E)$

```

const int MAX=10010*2; // (변수 개수)*2 [반드시 짹수]
//여기애 "SCC"를 넣으세요!! (struct 없이)
vector<int> solve() {
    make_SCC(MAX);
    for (int i = 0; i < MAX; i += 2)
        if (sccn[i] == sccn[i + 1])
            return vector<int>();
    vector<int> ans(MAX / 2, -1);
    int arr[MAX];
    for (int i = 0; i < MAX; ++i) arr[i] = i;
    sort(arr, arr + MAX, [&](int &a, int &b) {
        if (sccn[a] != sccn[b]) return sccn[a] > sccn[b];
        return a < b;
    });
    for (int i = 0; i < MAX; ++i)
        if (ans[arr[i] / 2] == -1)
            ans[arr[i] / 2] = !(arr[i] & 1);
    return ans;
}
void makeEdge(int X, bool XisTrue, int Y, bool YisTrue) {
    adj[2 * X + !XisTrue].push_back(2 * Y + YisTrue);
    adj[2 * Y + !YisTrue].push_back(2 * X + XisTrue);
}

```

2.3 BCC - 단절점, 단절선

Usage: 단방향 그래프의 단절점과 단절선.

complexity: $\mathcal{O}(V + E)$

```

const int MAX = 100;
vector<pii> graph[MAX]; // { next vertex id, edge id }
int up[MAX], vst[MAX], vtime;
vector<int> stk;
int is_cut[MAX]; // v is cut vertex if is_cut[v] > 0
vector<int> bridge; // list of edge ids
vector<int> bcc_edges[MAX]; // list of edge ids in a bcc
int bcc_cnt;
void dfs(int no, int pe) { // node, parent edge
    up[no] = vst[no] = ++vtime;
    int child = 0;
    for (const auto& e : graph[no]) {
        int next = e.xx, eid = e.yy;
        if (eid == pe) continue;

```

```

        if (vst[next] == 0) {
            stk.push_back(eid);
            ++child;
            dfs(next, eid);
            if (up[next] == vst[next]) bridge.push_back(eid);
            if (up[next] >= vst[no]) {
                ++bcc_cnt;
                do {
                    auto lasteid = stk.back();
                    stk.pop_back();
                    bcc_edges[bcc_cnt].push_back(lasteid);
                    if (lasteid == eid) break;
                } while (!stk.empty());
                is_cut[no]++;
            }
            up[no] = min(up[no], up[next]);
        }
        else if (vst[next] < vst[no]) {
            stk.push_back(eid);
            up[no] = min(up[no], vst[next]);
        }
    }
    if (pe == -1 && is_cut[no] == 1)
        is_cut[no] = 0;
}
void get_bcc() {
    vtime = 0;
    memset(vst, 0, sizeof(vst));
    memset(is_cut, 0, sizeof(is_cut));
    bridge.clear();
    for (int i = 0; i < MAX; ++i) bcc_edges[i].clear();
    bcc_cnt = 0;
    for (int i = 0; i < MAX; ++i) {
        if (vst[i] == 0)
            dfs(i, -1);
    }
}

```

2.4 SPFA (벨만 포드)

Usage: $\text{vector}<\text{int}> \text{adj}[\text{MAX}] = (\text{가중치}, \text{번호})$

complexity: $\mathcal{O}(VE), \text{Average}O(E)$

```

const int MAX; const ll INF;
vector<pll> adj[555]; //{정점, 가중치}
bool SPFA(int start, vector<ll> &dist){
    dist = vector<ll>(MAX, INF); dist[start] = 0;
    queue<int> Q; bool inQ[MAX] = {}; int visit[MAX] = {};
    Q.push(start); inQ[start] = true; ++visit[start];
    while(!Q.empty()){
        int here = Q.front(); Q.pop(); inQ[here] = false;
        for(auto &P : adj[here]){
            ll there = P.first, newDist = dist[here] + P.second;
            if(newDist < dist[there]){
                dist[there] = newDist;
                if(!inQ[there]){
                    Q.push(there); inQ[there] = true;
                    if(++visit[there] >= n+1) return false;
                }
            }
        }
    }
} return true;
}

```

2.5 방향 그래프 사이클 판정

```

int visit[MAX];
bool haveCycle(int here){
    if(visit[here]) return visit[here] == -1;
    visit[here] = -1;
    for(int &there : adj[here]) if(haveCycle(there)) return true;
    visit[here] = 1; return false;
}
memset(visit, -1, sizeof(visit));

```

2.6 다익스트라 알고리즘

complexity: $\mathcal{O}((V + E)\log(V + E))$

```

const int MAX; const ll INF;
vector<pll> adj[MAX]; //{weight, vertex}
void dijkstra(int start, vector<ll> &dist){
    dist = vector<ll>(MAX, INF); dist[start] = 0;
    priority_queue<pll, vector<pll>, greater<pll>> q;
    q.emplace(0, start);
    while(!q.empty()){

```

```

        auto [cost, u] = q.top(); q.pop();
        if(cost > dist[u]) continue;
        for(auto &[w, v] : adj[u]){
            ll ncost = cost + w;
            if(dist[v] > ncost){
                q.emplace(ncost, v); dist[v] = ncost;
            }
        }
    }
}

```

2.7 오프라인 동적 연결성 판정

complexity: $\mathcal{O}(Qlg^2Q)$

```

struct UnionFindWithRollback{
    int parent[100100];
    stack<tuple<int,int,int>> S;
    UnionFindWithRollback(){
        memset(parent, -1, sizeof(parent));
    }
    int find(int x){
        if(parent[x]<0) return x; return find(parent[x]);
    }
    bool merge(int x, int y){
        x = find(x), y = find(y); if(x == y) return 0;
        if(parent[x] < parent[y]) swap(x,y);
        S.push({x, y, parent[x]});
        parent[y] += parent[x]; parent[x] = y; return 1;
    }
    void rollback(){
        auto [x,y,sz] = S.top();
        parent[y] -= sz; parent[x] = sz; S.pop();
    }
}UF;
int query[100100][3];
vector<pii> tree[400400];
//라이프타임이 s~e인 간선 u-v를 분할-정복 트리의 노드에 추가.
//조건 : 1~r에 살아있는, 즉, s <= l && r <= e
void init(int u, int v, int s, int e, int no, int l, int r){
    if(e < l || r < s) return;
    if(s <= l && r <= e) {
        tree[no].push_back({u, v}); return;
    }
    int mid = (l+r)/2;
    init(u, v, s, mid, no*2, l, mid);
    init(u, v, mid+1, e, no*2+1, mid+1, r);
}

```

```

}

int mid = (l+r)/2;
init(u, v, s, e, no*2, l, mid); init(u, v, s, e, no*2+1, mid+1, r);
}

void solve(int no, int l, int r){
    //query[no]를 UF에 추가
    int change = 0;
    for(auto &p : tree[no]) change += UF.merge(p.xx, p.yy);
    //리프에 도달했으면, 현재 구간이 쿼리라면 해결!
    if(l == r && query[l][0] == 3)
        cout << (UF.find(query[l][1]) == UF.find(query[l][2])) << '\n';
    if(l != r){
        int mid = (l+r)/2; solve(no*2, l, mid); solve(no*2+1, mid+1, r);
    }
    //롤백
    while(change--) UF.rollback();
}

void main() {
    //1,2,3 u v : 추가, 제거, 쿼리 (u < v)
    map<pii, int> lifetime_map;
    for(int i=0; i<m; ++i){
        int &u = query[i][1], &v = query[i][2];
        if(query[i][0] == 1){ //간선 추가
            lifetime_map[{u, v}] = i;
        }
        else if(query[i][0] == 2){ //간선 삭제
            int s = lifetime_map[{u, v}];
            init(u, v, s, i, 1, 0, m-1);
            lifetime_map.erase({u, v});
        }
    }
    for(auto &p : lifetime_map){
        int u = p.xx.xx, v = p.xx.yy, s = p.yy;
        init(u, v, s, m-1, 1, 0, m-1);
    }
    solve(1, 0, m-1);
}

```

2.8 위상 정렬

```

const int MAX;
int indegree[MAX], result[MAX];
vector<int> adj[MAX];

```

```

bool topological_sort(int n){
    queue<int> Q;
    for(int i=0; i<n; ++i)
        if(!indegree[i]) Q.push(i);
    for(int i=0; i<n; ++i){
        if(Q.empty()) return false;
        int here = Q.front();
        Q.pop();
        result[i] = here;
        for(int &there : adj[here])
            if(!(--indegree[there]))
                Q.push(there);
    }
    return true;
}

```

3 트리

3.1 LCA (최소 공통 조상)

complexity: init - $\mathcal{O}(N \log N)$, query - $\mathcal{O}(\log N)$

```

const int MAX, pMAX;
int depth[MAX], parent[MAX][pMAX];
void init(){
    //TODO : make_tree : fill depth[i] & parent[i][0]
    //parent[i][j] = i의 1<<j번째 조상은?
    for(int j=1; j<pMAX; ++j)
        for(int i=1; i<=n; ++i)
            parent[i][j] = parent[parent[i][j-1]][j-1];
}

int LCA(int u, int v){
    if(depth[u] > depth[v]) swap(u,v);
    //v를 u의 높이까지 올린다
    int diff = depth[v] - depth[u];
    for(int i=0; i<pMAX; ++i){
        if(diff & (1<<i))
            v = parent[v][i];
    }
    if(u == v) return u;
    //부모가 다르면 올린다
    for(int i=pMAX-1; i>=0; --i)
        if(parent[u][i] != parent[v][i])

```

```

        u = parent[u][i], v = parent[v][i];
    return parent[u][0];
}

```

3.2 Heavy-light Decomposition

Usage: 모든 경로가 항상 $\log N$ 개의 체인만 지난다. set, Segment Tree 등을 끼얹기. 1. adj 채우고 2. make_tree(root,root) 3.HLD(root,root,root). 가중치가 간선에 붙은 구현이며, 정점에 붙으면 쿼리를 $DFS_cnt[x]+1 - DFS_cnt[x]$.

```

const int MAX;
struct SegmentTree{
    //query, update시 l > r이 들어올 수 있음.
}S;
vector<int> adj[MAX]; //입력 필요
int level[MAX]={}, sub_size[MAX]={}, parent[MAX]={};
int DFS_cnt[MAX]={}, dn=0, chain[MAX]={}, tail[MAX]={};
void make_tree(int here, int p){
    parent[here] = p;
    sub_size[here] = 1;
    for(int &i : adj[here]){
        if(i == parent[here]) continue;
        level[i] = level[here]+1;
        make_tree(i, here);
        sub_size[here] += sub_size[i];
    }
}
// DFS_cnt[i] = i번 정점의 DFS 넘버
// chain[i] = DFS 넘버가 i인 정점이 속하는 체인 넘버
// 체인의 머리의 노드 넘버는, 체인 넘버와 동일하다
// tail[i] = 체인 넘버가 i인 체인의 꼬리의 DFS 넘버
void HLD(int here, int p, int chain_cnt){
    DFS_cnt[here] = ++dn;
    chain[dn] = chain_cnt;
    tail[chain_cnt] = dn;
    int heavy_idx = -1;
    for(int &i : adj[here])
        if(i != p && (heavy_idx == -1 || sub_size[i] > sub_size[heavy_idx]))
            heavy_idx = i;
    if(heavy_idx != -1) HLD(heavy_idx, here, chain_cnt);
    for(int &i : adj[here])
        if(i != p && i != heavy_idx)

```

```

            HLD(i, here, i);
        }
void update(int node, int value){
    S.update(DFS_cnt[node], value);
}
int query(int x, int y){
    if(chain[DFS_cnt[x]] == chain[DFS_cnt[y]]){ //같은 체인
        if(level[x] > level[y]) swap(x,y);
        //노드에 가중치가 달린 경우 DFS_cnt[x]+1 -> DFS_cnt[x]
        return S.query(DFS_cnt[x] + 1, DFS_cnt[y]);
    }
    int x_head = chain[DFS_cnt[x]], y_head = chain[DFS_cnt[y]];
    if(level[x_head] > level[y_head])
        return max(S.query(DFS_cnt[x_head], DFS_cnt[x]), query(parent[x_head], y));
    else
        return max(S.query(DFS_cnt[y_head], DFS_cnt[y]), query(x, parent[y_head]));
}

```

3.3 Centroid Decomposition

Usage: 트리에서 분할정복

```

bool vst[MAX]; //i를 센트로이드로 문제를 푼 적이 있는가?
vector<int> adj[MAX]; int sz[MAX];
int get_size(int u, int p) {
    sz[u] = 1;
    for (auto &v : adj[u]) {
        if (v == p || vst[v]) continue;
        sz[u] += get_size(v, u);
    }
    return sz[u];
}
int get_cen(int u, int p, int cap) { //크기가 cap인 서브트리의 cen
    for (auto &v : adj[u]) {
        if (v == p || vst[v] || sz[v] <= cap / 2) continue;
        return get_cen(v, u, cap);
    }
    return u;
}
int solve(int u){
    int cen = get_cen(u, u, get_size(u, u)), ans = INF;
    for(auto &v : adj[cen]){/*TODO*/} //cen을 지나는 경로들을 처리
    for(auto &v : adj[cen]){ //분할&정복 : 자식들을 호출

```

```

    if(vst[v]) continue; ans = min(ans, solve(v));
} return ans;
}

```

4 플로우

4.1 이분 매칭 - Easy

Usage: 코너의 정리 - 최소 버텍스 커버 = 최대 이분 매칭
(A 중 소스에서 도달 불가능 + B 중 소스에서 도달 가능)

complexity: $\mathcal{O}(VE)$

```

const int MAXA, MAXB;
int A[MAXA], B[MAXB];
bool visit[MAXA];
vector<int> adj[MAXA];
bool DFS(int here){
    if(visit[here]) return false;
    visit[here] = true;
    for(int &there : adj[here])
        if(B[there] == -1 || DFS(B[there])){
            A[here] = there, B[there] = here;
            return true;
        }
    return false;
}
int biMatch(){
    memset(A, -1, sizeof(A));
    memset(B, -1, sizeof(B));
    int match = 0;
    for(int i=0; i<MAXA; ++i)
        if(A[i] == -1){
            memset(visit, 0, sizeof(visit));
            match += DFS(i);
        }
    return match;
}

```

4.2 이분 매칭 - Fast (호프크로프트 카프)

Usage: $n = A$ 의 크기, $MAX = \max(A\text{size}, B\text{size})$

1. bfs-bmatching으로 레벨[=매칭 중이지 않은 정점과의 최단거리]을 매긴다.
2. dfs-bmatching으로 level을 이용해 효율적으로 이분매칭을 할 수 있다

complexity: $\mathcal{O}(E\sqrt{V})$

```

const int MAX;//MAX = max(Asize,Bsize)
int n, A[MAX], B[MAX], dist[MAX];
bool used[MAX];
vector<int> adj[MAX];
void bfs_bmatching(){
    queue<int> Q;
    for (int i = 0; i < n; ++i) {
        if (!used[i]) dist[i] = 0; Q.push(i);
        else dist[i] = MAX * 2 + 1;
    }
    while (!Q.empty()) {
        int a = Q.front(); Q.pop();
        for (int &b : adj[a]) {
            if (B[b] != -1 && dist[B[b]] == MAX * 2 + 1) {
                dist[B[b]] = dist[a] + 1;
                Q.push(B[b]);
            }
        }
    }
}
bool dfs_bmatching(int a) {
    for (int &b : adj[a]) {
        if (B[b] == -1 || (dist[B[b]] == dist[a] + 1 &&
            dfs_bmatching(B[b]))) {
            used[a] = true;
            A[a] = b; B[b] = a;
            return true;
        }
    }
    return false;
}
int Hopcroft_Karp() {
    int match = 0, flow;
    memset(A, -1, sizeof(A));
    memset(B, -1, sizeof(B));
    memset(dist, 0, sizeof(dist));
    memset(used, 0, sizeof(used));
    while (1) {
        bfs_bmatching();
        flow = 0;
        for (int i = 0; i < n; ++i)

```

```

        if (!used[i] && dfs_bmatching(i)) flow++;
        if (!flow) break;
        match += flow;
    }
    return match;
}

//tourist's O(VE) faster than hopcroft
const int MAX = 100;
vector<int> adj[MAX];
int iter, pa[MAX], pb[MAX], was[MAX];
bool dfs(int v) {
    was[v] = iter;
    for (int u : adj[v])
        if (pb[u] == -1) {
            pa[v] = u; pb[u] = v; return 1;
        }
    for (int u : adj[v])
        if (was[pb[u]] != iter && dfs(pb[u])) {
            pa[v] = u; pb[u] = v; return 1;
        }
    return 0;
}
int biMatch() {
    memset(pa, -1, sizeof(pa));
    memset(pb, -1, sizeof(pb));
    memset(was, 0, sizeof(was));
    iter = 0;
    int res = 0;
    while (true) {
        iter++;
        int add = 0;
        for (int i = 0; i < MAX; i++)
            if (pa[i] == -1 && dfs(i)) add++;
        if (!add) break;
        res += add;
    }
    return res;
}

```

4.3 최대 플로우 - Easy (에드몬드 카프)

Usage: Max-Flow Min-Cut : S와 E로 정점 그룹을 나눌 때, 최대 매칭이 곧 최소 컷의 용량의 합이다. 즉, 잔여 용량이 남은 간선을 타고 갈 수 있는 정점은 S에 속한다.

complexity: $\min(O(VE^2), O(Ef))$

```

const int MAX = 1000, INF = INT_MAX;
struct edge {
    int next, cap, flow = 0, rev_idx;
    edge() {}
    edge(int n, int c) : next(n), cap(c) {}
    int remain() {
        return cap - flow;
    }
};
vector<edge> adj[MAX];
void makeEdge(int u, int v, int c) {
    adj[u].emplace_back(v, c);
    adj[v].emplace_back(u, 0);
    adj[u].back().rev_idx = adj[v].size() - 1;
    adj[v].back().rev_idx = adj[u].size() - 1;
}
int parent[MAX];
pii path[MAX];
int solve(int S, int E) {
    int ans = 0;
    queue<int> q;
    while (1) {
        memset(parent, -1, sizeof(parent));
        q.push(S);
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (int i = 0; i < adj[u].size(); ++i) {
                auto &e = adj[u][i];
                if (e.remain() > 0 && parent[e.next] == -1) {
                    parent[e.next] = u;
                    path[e.next] = {u, i};
                    q.emplace(e.next);
                }
            }
        }
        if (parent[E] == -1) break;
        int ret = INF;
        for (int i = E; i != S; i = parent[i])

```

```

        ret = min(ret, adj[path[i].xx][path[i].yy].remain());
    for (int i = E; i != S; i = parent[i]) {
        auto &e = adj[path[i].xx][path[i].yy];
        e.flow += ret;
        adj[e.next][e.rev_idx].flow -= ret;
    }
    ans += ret;
}
return ans;
}

```

4.4 최대 플로우 - Fast (디닉)

Usage: INF = 최대 유량 값, 0-1 cap에서 $\min(O(EV^{2/3}), O(E^{3/2}))$

레벨은 ”소스까지의 최단거리의 간선의 개수”를 의미한다.

BFS로 ’레벨 그래프’를 만든다. 레벨 그래프에서 포함하는 간선은

1. 유량이 남아있는 간선 2. 레벨의 차이가 1인 정점을 연결하는 간선

레벨 그래프를 생성하고, 에드몬드 카프처럼 DFS를 돌려 유량을 흘려보내 준다.

complexity: $\mathcal{O}(EV^2)$

```
const int MAX, INF; //MAX = 정점의 개수, INF = 최대 용량
```

```
int level[MAX], work[MAX];
```

```
struct Edge{
```

```
    int next, cap, flow=0; Edge* rev;
```

```
    Edge(){}
```

```
    Edge(int _next, int _cap) : next(_next), cap(_cap) {};
```

```
};
```

```
vector<Edge*> adj[MAX];
```

```
inline void make_edge(int u, int v, int cap){
```

```
    Edge *uv = new Edge(v, cap), *vu = new Edge(u, 0);
    uv->rev = vu, vu->rev = uv;
    adj[u].push_back(uv);
    adj[v].push_back(vu);
```

```
}  
//싱크까지 도달 가능한 레벨 그래프를 생성 할 수 있는가?
```

```
bool bfs(int S, int E){
```

```
    memset(level, -1, sizeof(level));
```

```
    level[S] = 0;
```

```
    queue<int> Q;
```

```
    Q.push(S);
```

```
    while(!Q.empty()){

    }
```

```
        int here = Q.front(); Q.pop();
        for(auto &i : adj[here]){

    }
```

```

        int there = i->next;
        //레벨이 정해지지 않고, 남은 용량이 있는 간선인가?
        if(level[there] == -1 && i->cap >0){
            level[there] = level[here]+1;
            Q.push(there);
        }
    }
}
return (level[E] != -1);
}

//here에서 E에 도달할 때 까지 flow 만큼 흘려보내본다.
//싱크까지 얼마나 유량을 흘릴 수 있었는지 리턴한다.
int dfs(int here, int flow, int E){
    if(here == E) return flow;
    int size = adj[here].size();
    for(int &i = work[here]; i<size; ++i){
        int there = adj[here][i]->next, c = adj[here][i]->cap;
        if(level[here]+1 == level[there] && c >0){
            int nc = dfs(there, min(flow, c), E);
            if(nc){
                adj[here][i]->cap -= nc;
                adj[here][i]->rev->cap += nc;
                return nc;
            }
        }
    }
    return 0;
}

int maxflow(int S, int E){
    int total = 0;
    while(bfs(S,E)){
        memset(work, false, sizeof(work));
        while(true){
            int flow = dfs(S, INF, E);
            if(flow == 0) break;
            total += flow;
        }
    }
    return total;
}

```

4.5 MCMF (Minimum Cost Maximum Flow)

Usage: INF = 가중치의 최장거리. (가중치의 최소값, 유량의 최대값)을 반환한다.
complexity: $\mathcal{O}(VEf)$, Average = $O(Ef)$

```
const int MAX, INF; //MAX = 최대 정점 개수, INF = 가중치의 최장거리
struct MinCostMaxFlow{
    struct Edge{
        int next, cap, cost, flow=0;
        Edge* rev;
        Edge(){};
        Edge(int _next, int _cap, int _cost) : next(_next), cap(_cap),
        cost(_cost) {};
        int remain(){
            return cap - flow;
        }
        void push(int x){
            flow += x;
            rev->flow -= x;
        }
    };
    vector<Edge*> adj[MAX];
    inline void makeEdge(int u, int v, int cap, int cost){
        Edge *uv = new Edge(v, cap, cost), *vu = new Edge(u, 0, -cost);
        uv->rev = vu, vu->rev = uv;
        adj[u].push_back(uv);
        adj[v].push_back(vu);
    }
    pair<int,int> MCMF(int S, int E){
        int minCost = 0, maxFlow = 0;
        while(true){
            int parent[MAX], dist[MAX];
            Edge *path[MAX];
            queue<int> Q;
            bool inQ[MAX] = {};
            memset(parent, -1, sizeof(parent));
            fill(dist, dist+MAX, INF);
            dist[S] = 0;
            Q.push(S);
            inQ[S] = true;
            while(!Q.empty()){
                int here = Q.front(); Q.pop();
                inQ[here] = false;
                for(auto e : adj[here]){

```

```
                    int there = e->next;
                    if(e->remain() > 0 && dist[there] > dist[here] + e->cost){
                        dist[there] = dist[here] + e->cost;
                        parent[there] = here;
                        path[there] = e;
                        if(!inQ[there]){
                            Q.push(there);
                            inQ[there] = true;
                        }
                    }
                }
                if(parent[E] == -1) break;
                int flow = INF;
                for(int i=E; i!=S; i=parent[i])
                    flow = min(flow, path[i]->remain());
                for(int i=E; i!=S; i=parent[i]){
                    minCost += flow * path[i]->cost;
                    path[i]->push(flow);
                }
                maxFlow += flow;
            }
            return {minCost, maxFlow};
        }
    }F;
```

5 수학

5.1 빠른 조합(nCr) + 나눗셈 역원

Usage: 1~N의 역원을 전처리.

```
const int MAX, MOD; //MAX = n의 최대값
ll inv[MAX+1], fac[MAX+1], facInv[MAX+1];
void combInit(){
    fac[0] = 1;
    for(int i=1; i<=MAX; ++i) fac[i] = (fac[i-1]*i)%MOD;
    inv[1] = 1;
    for(int i=2; i<=MAX; ++i) inv[i] = (MOD-MOD/i)*inv[MOD%i]%MOD;
    facInv[1] = 1;
    for(int i=2; i<=MAX; ++i) facInv[i] = (facInv[i-1]*inv[i])%MOD;
}
```

```

inline ll comb(int n, int r){
    if(r == 0 || n == r) return 1;
    return (((fac[n]*facInv[r])%MOD)*facInv[n-r])%MOD;
}
inline ll divmod(ll a, ll b){ // return (a/b)%MOD, MOD = prime
    return (a*fastPow(b, MOD-2))%MOD;
}

```

5.2 채(sieve) : 소수, 소인수분해, 오일러 피

Usage: sqMAX까지의 소수만 표시하므로, MAX까지 구하려면 i반복문 수정
 complexity: $\mathcal{O}(N \log \log N)$

```

//소수
bool sieve[MAX+1];
void find_prime(){
    sieve[0] = sieve[1] = 1;
    for(ll i=2; i*i<=MAX; ++i){ // i<=MAX if want prime vector
        if(sieve[i] == 0)
            for(ll j=i*i; j<=MAX; j+=i)
                sieve[j] = 1;
    }
}

//소인수분해
const int MAX = 1000000;
int sieve[MAX+1];
void find_prime(){
    memset(sieve, -1, sizeof(sieve));
    for(ll i=2; i*i<=MAX; ++i)
        if(sieve[i] == -1)
            for(ll j=i*i; j<=MAX; j+=i)
                if(sieve[j] == -1)
                    sieve[j] = i;
}

void print_factor(int n){
    while(sieve[n] != -1){
        cout << sieve[n] << '\n';
        n /= sieve[n];
    }
    cout << n << '\n';
}

//1~N의 약수의 개수 구하기 O(NlogN)
void num_of_divisors(int n, int ret[]) {

```

```

    for (int i = 1; i <= n; ++i)
        for (int j = i; j <= n; j += i)
            ret[j] += 1;
}

//오일러 피 함수 O(n*loglogn)
// 0 < x < n && gcd(n, x) = 1 인 x의 개수
void euler_phi(int n, int ret[]) {
    for (int i = 1; i <= n; ++i) ret[i] = i;
    for (int i = 2; i <= n; ++i)
        if (ret[i] == i)
            for (int j = i; j <= n; j += i)
                ret[j] -= ret[j] / i;
}

```

5.3 소수 찾기 - Fast (밀러-라빈)

complexity: $\mathcal{O}(\log N \log N)$

```

typedef unsigned long long ull;
ll large_mod_mul(ll a, ll b, ll m) { //64bits
    return ll((__int128)a*(__int128)b%m);
}

ll large_mod_mul(ll a, ll b, ll m) { // |m| < 2^62, 32bits
    a %= m; b %= m; ll r = 0, v = a;
    while (b) {
        if (b&1) r = (r + v) % m;
        b >= 1;
        v = (v << 1) % m;
    }
}

ll modpow(ll n, ll k, ll m) { // calculate n^k % m
    ll ret = 1;
    n %= m;
    while (k) {
        if (k & 1) ret = large_mod_mul(ret, n, m);
        n = large_mod_mul(n, n, m);
        k /= 2;
    }
    return ret;
}

bool test_witness(ull a, ull n, ull s) {
    if (a >= n) a %= n;
    if (a <= 1) return true;
    ull d = n >> s;

```

```

ull x = modpow(a, d, n);
if (x == 1 || x == n-1) return true;
while (s-- > 1) {
    x = large_mod_mul(x, x, n);
    if (x == 1) return false;
    if (x == n-1) return true;
}
return false;
}
bool is_prime(ull n) { //test prime, O(logn*logn)
if (n == 2) return true;
if (n < 2 || n % 2 == 0) return false;
ull d = n >> 1, s = 1;
for(; (d&1) == 0; s++) d >>= 1;
#define T(a) test_witness(a##ull, n, s)
    if (n < 4759123141ull) return T(2) && T(7) && T(61);
    return T(2) && T(325) && T(9375) && T(28178)
    && T(450775) && T(9780504) && T(1795265022);
#define T(a)
#endif
}

```

5.4 소인수분해 - Fast (플라드-로)

Usage: 밀리 라빈 필요. 소인수분해 결과를 무작위 순서로 반환

complexity: $\mathcal{O}(N^{0.25} \log N)$

```

ll pollard_rho(ll n) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<ll> dis(1, n - 1);
    ll x = dis(gen);
    ll y = x;
    ll c = dis(gen);
    ll g = 1;
    while (g == 1) {
        x = (large_mod_mul(x, x, n) + c) % n;
        y = (large_mod_mul(y, y, n) + c) % n;
        y = (large_mod_mul(y, y, n) + c) % n;
        g = gcd(abs(x - y), n);
    }
    return g;
}
// integer factorization O(n^0.25 * logn)

```

```

void factorize(ll n, vector<ll>& fl) {
    if (n == 1) {
        return;
    }
    if (n % 2 == 0) {
        fl.push_back(2);
        factorize(n / 2, fl);
    }
    else if (is_prime(n)) {
        fl.push_back(n);
    }
    else {
        ll f = pollard_rho(n);
        factorize(f, fl);
        factorize(n / f, fl);
    }
}

```

5.5 FFT (고속 푸리에 변환)

Usage: A,B를 크기가 2^K 꼴이 될때까지 0을 채워넣고 뒤집은 후,
B의 꼬리가 i에 있을 때 A와 B를 곱한 값

complexity: $\mathcal{O}(N \log N)$

```

void fft(vector<complex<double>> &a, bool invert){
    int n = a.size();
    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);
        complex<double> wlen(cos(ang),sin(ang));
        for (int i=0;i<n;i+=len){
            complex<double> w(1);
            for (int j=0;j<len/2;j++){
                complex<double> 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(vector<int> &a, vector<int> &b, vector<int> &ret){
    vector<complex<double>> fa(all(a)), fb(all(b));
    int n = 1;
    while (n < max((int)a.size(),(int)b.size())) 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);
    ret.resize(n);
    for (int i=0;i<n;i++) ret[i] =
        int(fa[i].real()+(fa[i].real()>0?0.5:-0.5));
}

```

5.6 중국인의 나머지 정리 + 확장 유클리드

Usage: $n[]$ 은 모두 소수여야 한다. $a \% (n\text{의 모든 곱})$ 을 구해준다.

```

ll extended_gcd(ll a, ll b) { //ac + bd = gcd(a,b)인 (c,d)
    if(b == 0) return make_pair(1, 0);
    ll t = extended_gcd(b, a % b);
    return make_pair(t.second, t.first - t.second * (a / b));
}
ll modinverse(ll a, ll m) { //ax = gcd(a,m)%m인 x
    return (extended_gcd(a, m).first % m + m) % m;
}
//a[i] \% n[i] 가 동일한 최소 x를 찾아줌
ll chinese_remainder(ll *a, ll *n, int size) {
    if (size == 1) return *a;
    ll tmp = modinverse(n[0], n[1]);
    ll tmp2 = (tmp * (a[1] - a[0]) \% n[1] + n[1]) \% n[1];
    ll ora = a[1]; ll tgcd = gcd(n[0], n[1]);
    a[1] = a[0] + n[0] / tgcd * tmp2; n[1] *= n[0] / tgcd;
    ll ret = chinese_remainder(a + 1, n + 1, size - 1);
    n[1] /= n[0] / tgcd; a[1] = ora; return ret;
}
// mod( $\pi(M)$ )에서  $x = B[i] \% M[pos+i]$  인 x를 구해준다
ll CRT_nocoprime(vector<ll> &B, vector<ll> &M) {

```

```

    ll retB=B[0], retM=M[0];
    for(int i=1; i<M.size(); ++i){
        ll m1=M[i], b1=B[i], m2=retM, b2=retB;
        ll g=gcd(m1, m2); if((b1-b2)%g) return -1;
        retM=m1*m2/g; ll m=m1/g; ll v=((b1-b2)/g%m+m)%m;
        ll w=(extended_gcd(m2/g%m, m1/g).first%m+m)%m;
        retB=b2+v*w%m*m2;
    } return retB;
}

```

5.7 mint - 빠른 모듈러 연산

```

const int mod = 1000000007;
struct mi{
    int v;
    mi() { v = 0; }
    mi(const ll& x) {
        v = (-mod <= x && x < mod) ? x : x % mod;
        if (v < 0) v += mod;
    }
    friend ostream& operator<<(ostream& os, const mi& a) { return os << a.v; }
    friend bool operator==(const mi& a, const mi& b) { return a.v == b.v; }
    friend bool operator!=(const mi& a, const mi& b) { return !(a == b); }
    friend bool operator<(const mi& a, const mi& b) { return a.v < b.v; }
    mi operator-() const { return mi(-v); }
    mi& operator+=(const mi& m) { if((v+m.v)>=mod) v-=mod; return *this; }
    mi& operator-=(const mi& m) { if((v - m.v) < 0) v += mod; return *this; }
    mi& operator*=(const mi& m) { v = (ll)v*m.v%mod; return *this; }
    friend mi ipow(mi a, ll p) {
        mi ans = 1; for (; p; p /= 2, a *= a) if (p&1) ans *= a; return ans;
    }
    friend mi inv(const mi& a) { assert(a.v); return ipow(a, mod - 2); }
    mi& operator/=(const mi& m) { return (*this) *= inv(m); }
    friend mi operator+(mi a, const mi& b) { return a += b; }
    friend mi operator-(mi a, const mi& b) { return a -= b; }
    friend mi operator*(mi a, const mi& b) { return a *= b; }
    friend mi operator/(mi a, const mi& b) { return a /= b; }
    operator int64_t() const {return v; }
};

```

6 문자열

6.1 해싱

```

const int MAX = 1000010, base = 257, m = 1e9+7;
vector<ll> ppow;
void init(){
    ppow.resize(MAX);
    ppow[0] = 1;
    for(int i=1; i<MAX; ++i)
        ppow[i] = (ppow[i - 1] * base) % m;
}
void hash(vector<ll> &h, string &s){
    h.clear(); h.resize(s.size()+1);
    for(int i=0; i<s.size(); ++i)
        h[i+1] = (h[i] + (s[i]+1) * ppow[i]) % m;
}
inline ll get(vector<ll> &h, int l, int len){
    return (h[l+len] - h[l] + m) % m;
}
bool compare(vector<ll> &h1, int l1, vector<ll> &h2, int l2, int len){
    ll v1 = get(h1, l1, len), v2 = get(h2, l2, len);
    if(l1 < l2) v1 = (v1 * ppow[l2 - l1]) % m;
    if(l1 > l2) v2 = (v2 * ppow[l1 - l2]) % m;
    return v1 == v2;
}

```

6.2 트라이 (문자열 전용 set)

complexity: $\mathcal{O}(|S|)$

```

const int MAXN = 100*1000 +1, MAXC = 10; // 트라이 노드 개수, 알파벳 개수
int cnt, trie[MAXN][MAXC]; //트라이 번호 매기기, 트라이 노드
bool term[MAXN]; //N번째 트라이가 종말 노드인가?
void clear(){
    memset(trie, 0, sizeof(trie));
    memset(term, 0, sizeof(term));
    cnt = 0;
}
void insert(string &x){
    int here = 0;
    for(char &i : x){
        if(!trie[here][i-'0']) trie[here][i-'0'] = ++cnt;
        here = trie[here][i-'0'];
    }
}

```

```

    }
    term[here] = 1;
}
bool query(string &x){
    int here = 0;
    for(char &i : x){
        if(!trie[here][i-'0']) return 0;
        here = trie[here][i-'0'];
        if(term[here]) return 1;
    }
    return 0;
}

```

6.3 KMP

Usage: fail[i] = str[0:i]의 접두사이며 접미사인 최대 부분 문자열의 길이)
complexity: $\mathcal{O}(N + M)$

```

void KMP(string &text, string &pat){
    vector<int> fail(pat.size(), 0);
    //pat과 pat끼리 KMP를 실행한다
    for(int i=1, match=0; i<pat.size(); ++i){
        while(match && pat[i] != pat[match])
            match = fail[match-1];
        if(pat[i] == pat[match])
            fail[i] = ++match;
    }
    for(int i=0, match=0; i<text.size(); ++i){
        while(match && text[i] != pat[match])// 매칭 중에 불일치 발생
            match = fail[match-1]; //맞았었던 부분의 실패함수 적용
        if(text[i] == pat[match]) //한 글자 매칭 성공
            if(++match == pat.size()){//완전 매칭 성공
                cout << "match at " << (i-match+1);
                match = fail[match-1];
            }
    }
}

```

6.4 접미사 배열 (Suffix Array + LCP)

Usage: suffix array - S에 대한 접미사 배열을 계산한 벡터를 반환
i번째 값은 사전순으로 i번째인 접미사의 시작 인덱스
LCP - 가장 긴 공통 접두사. 사전순으로 i번째 접미사와 i-1번째 접미사의 가장 긴 공통 접두
사의 길이. i=0일때는 정의되지 않으므로 벡터의 크기가 1 작다

complexity: $\mathcal{O}(N \log N)$

```

vector<int> suffix_array(const vector<char>& in) {
    int n = (int)in.size(), c = 0;
    vector<int> temp(n), pos2bckt(n), bckt(n), bpos(n), out(n);
    for (int i = 0; i < n; i++) out[i] = i;
    sort(out.begin(), out.end(), [&](int a, int b) { return in[a] < in[b]; });
    for (int i = 0; i < n; i++) {
        bckt[i] = c;
        if (i + 1 == n || in[out[i]] != in[out[i + 1]]) c++;
    }
    for (int h = 1; h < n && c < n; h <= 1) {
        for (int i = 0; i < n; i++) pos2bckt[out[i]] = bckt[i];
        for (int i = n - 1; i >= 0; i--) bpos[bckt[i]] = i;
        for (int i = 0; i < n; i++)
            if (out[i] >= n - h) temp[bpos[bckt[i]]++] = out[i];
        for (int i = 0; i < n; i++)
            if (out[i] >= h) temp[bpos[pos2bckt[out[i] - h]]++] = out[i] - h;
        c = 0;
        for (int i = 0; i + 1 < n; i++) {
            int a = (bckt[i] != bckt[i + 1]) || (temp[i] >= n - h)
                || (pos2bckt[temp[i + 1] + h] != pos2bckt[temp[i] + h]);
            bckt[i] = c;
            c += a;
        }
        bckt[n - 1] = c++;
        temp.swap(out);
    }
    return out;
}

vector<int> lcp(const vector<char>& in, const vector<int>& sa) {
    int n = (int)in.size();
    if (n == 0) return vector<int>();
    vector<int> rank(n), height(n - 1);
    for (int i = 0; i < n; i++) rank[sa[i]] = i;
    for (int i = 0, h = 0; i < n; i++) {
        if (rank[i] == 0) continue;
        int j = sa[rank[i] - 1];
        while (i + h < n && j + h < n && in[i + h] == in[j + h]) h++;
        height[rank[i] - 1] = h;
        if (h > 0) h--;
    }
}

```

```

        return height;
}

```

6.5 모든 팰린드롬 찾기 - Manacher

Usage: i번째 글자를 중심으로 하는 팰린드롬의 반지름을 구해준다.

같은 더미를 끼워서 사용. $\mathcal{O}(N)$

complexity: $\mathcal{O}(N)$

```

// s=".a.b.c.d.e." -> m[i] = s[i]가 중심인 '실제' 팰린드롬의 길이
void manacher(const string& s, int m[]) {
    int r = -1, p = -1;
    for (int i=0; i<s.size(); ++i) {
        if (i <= r) m[i] = min((2*p-i)>= 0) ? m[2*p-i] : 0, r-i); else m[i]
        = 0;
        while(i-m[i]-1>=0 && i+m[i]+1<s.size() && s[i-m[i]-1] ==
        s[i+m[i]+1]) ++m[i];
        if (i + m[i] > r) r = i + m[i], p = i;
    }
}

```

6.6 Z Algorithm

Usage: Z[i] : S의 prefix로면서 S[i]의 prefix인 최대 길이

complexity: $\mathcal{O}(N \log N)$

```

vector<int> Zalgorithm(string &S) {
    int n = S.size(), l=0, r=0;
    vector<int> Z(n);
    for (int i = 1; i < n; i++) {
        if (i > r) {
            l = r = i;
            while (r < n && S[r - 1] == S[r]) r++;
            Z[i] = r - l; r--;
        }
        else {
            int k = i - l;
            if (Z[k] < r - i + 1) Z[i] = Z[k];
            else {
                l = i;
                while (r < n && S[r - 1] == S[r]) r++;
                Z[i] = r - l; r--;
            }
        }
    }
}

```

```

}
return Z;
}

```

6.7 아호코라식 (한 문자열 : 여러 패턴)

```

const int MAXN = 100*1000 +1, MAXC = 26; // 트라이 노드 개수, 알파벳 개수
int piv, trie[MAXN][MAXC], fail[MAXN]; //트라이 번호 매기기, 트라이, 실패
함수 링크
bool term[MAXN]; //N번째 트라이가 종말 노드인가?
void init(vector<string> &v){
    memset(trie, 0, sizeof(trie));
    memset(fail, 0, sizeof(fail));
    memset(term, 0, sizeof(term));
    piv = 0;
    for(string &i : v){
        int p = 0;
        for(char &j : i){
            if(!trie[p][j-'a']) trie[p][j-'a'] = ++piv;
            p = trie[p][j-'a'];
        }
        term[p] = 1;
    }
    queue<int> que;
    for(int i=0; i<MAXC; i++)
        if(trie[0][i]) que.push(trie[0][i]);
    while(!que.empty()){
        int x = que.front(); que.pop();
        for(int i=0; i<MAXC; i++){
            if(trie[x][i]){
                int p = fail[x];
                while(p && !trie[p][i]) p = fail[p];
                p = trie[p][i];
                fail[trie[x][i]] = p;
                if(term[p]) term[trie[x][i]] = 1;
                que.push(trie[x][i]);
            }
        }
    }
}

bool query(string &s){
    int p = 0;
    for(char &i : s){

```

```

        while(p && !trie[p][i-'a']) p = fail[p];
        p = trie[p][i-'a'];
        if(term[p]) return 1;
    }
    return 0;
}

```

7 DP

7.1 컨벡스 헬 트릭 (Convex Hull Trick; CHT)

Usage: $dp[i] = \min(0 \leq j < i) A(i)B(j) + C(j) + D(i)$

// O(logN)에 $ax + b$ 꼴의 직선들에 대해 최대값을 구한다.
// 실수 사용을 원하면, inf = 1/.0, div(a,b) = a/b
// 최소값 쿼리를 원하면 add/query시 a, b에 *= -1

```

struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
    const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) { x->p = inf; return false; }
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll a, ll b) {
        auto z = insert({a, b, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
}

```

```

    }
}CHT;

```

7.2 리차오 트리(Li Chao Tree)

Usage: $y = ax + b \leq \max\{f(x)\}$ 를 세그트리로

```

struct Line{ ll a, b; ll f(ll x){return a*x+b;} };
struct Node{ int nl, nr; ll xl, xr; Line line; };
vector<Node> seg;
void clear(ll xl, ll xr){//x의 최소, 최대값
    seg.clear();
    seg.push_back({-1, -1, xl, xr, {0, INF}});
}

void update(int no, const Line &v){ //update(0, v) : 직선 v 추가
    ll xl = seg[no].xl, xr = seg[no].xr, xm = (xl+xr)/2;
    Line lo = seg[no].line, hi = v;
    if(lo.f(xl) > hi.f(xl)) swap(lo, hi); // f(nl)의 값이 hi가 크다.
    if(lo.f(xr) <= hi.f(xr)){ //1) hi가 언제나 위
        seg[no].line = hi; return;
    }
    if (lo.f(xm) < hi.f(xm)){//2) 오른쪽에서 lo가 역전
        seg[no].line = hi;
        if (seg[no].nr == -1){ //오른쪽에 새 노드
            seg[no].nr = seg.size();
            seg.push_back({-1, -1, xm+1, xr, {0, INF}});
        }
        update(seg[no].nr, lo);
    }
    else{//3) 왼쪽에서 lo가 역전
        seg[no].line = lo;
        if (seg[no].nl == -1){ //왼쪽에 새 노드
            seg[no].nl = seg.size();
            seg.push_back({-1, -1, xl, xm, {0, INF}});
        }
        update(seg[no].nl, hi);
    }
}
ll query(int no, ll x){ //query(0, x) : f(x)의 최대값은?
    if (no == -1) return INF;
    ll xl = seg[no].xl, xr = seg[no].xr, xm = (xl+xr)/2;
    if (x <= xm) return max(seg[no].line.f(x), query(seg[no].nl, x));
}

```

```

    else return max(seg[no].line.f(x), query(seg[no].nr, x));
}

```

7.3 벌레캠프(Berlekamp-Massey)

Usage: 점화식 길이 k 의 3배(2배?)의 초항을 넣어준다.

complexity: $\mathcal{O}(nk + n\log m), nth_term = \mathcal{O}(k^2 \log n)$

```

const int mod = 998244353;
ll ipow(ll x, ll p){
    ll 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++){
        ll 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;
        }
        ll 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, ll 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] += 111 * 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] += 111 * 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;
    }
    ll ret = 0;
    for(int i=0; i<m; i++) ret += 111 * s[i] * dp[i] % mod;
    return ret % mod;
}
int guess_nth_term(vector<int> x, ll 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);
    }
}
```

7.4 Monotone Queue (덱으로 최소값)

```

deque<pii> q; // {value, idx}
for(int i=0; i<n; ++i){
    int a = v[i];
    while(!q.empty()){
        if(q.back().xx >= a) q.pop_back();
        else break;
    }
    while(!q.empty()){
        if(q.front().yy < i-m+1) q.pop_front();
        else break;
    }
    q.emplace_back(a, i);
    cout << q.front().xx << ' ';
}
}
```

8 기하

8.1 기본 기하 라이브러리 (CCW)

```

// Integer & Float
typedef pair<geom_t, geom_t> Point;
Point operator- (const Point &a, const Point &b){ return {a.xx-b.xx,
a.yy-b.yy}; }
inline geom_t outer(const Point& a, const Point& b) { return a.xx*b.yy -
a.yy*b.xx; }

Point operator+ (const Point &a, const Point &b){ return {a.xx+b.xx,
a.yy+b.yy}; }
Point rot45(Point p) { return {p.xx+p.yy, p.yy-p.xx}; }
inline geom_t inner(const Point& a, const Point& b) { return a.xx*b.xx +
a.yy*b.yy; }
inline geom_t sign(const geom_t& x) { return (x ? x / abs(x) : 0); }
inline geom_t euclid2(const Point& a, const Point& b) {
    Point ret = a - b;
    return ret.xx*ret.xx + ret.yy*ret.yy;
} // a와 b의 직선 거리
inline geom_t manhattan(const Point& a, const Point& b) {
    Point ret = a - b;
    return abs(ret.xx) + abs(ret.yy);
}
```

```

        return abs(ret.xx) + abs(ret.yy);
    } //a와 b의 택시 거리
    struct Line {
        Point st, ed;
    };
    bool angle_cmp(const Point &a, const Point &b){
        if( (Point(0,0) < a) ^ (Point(0,0) < b) ) return a > b;
        return ccw(Point(0,0), a, b) > 0; // 0일때 같은 각도
    }

    // Integer Geometry
    typedef geom_t ll;
    inline ll sign(ll x) { return (x ? x/abs(x) : 0); }
    inline ll ccw(const Point& a, const Point& b, const Point& c) { return
sign(outer(b - a, c - a)); }
    inline bool intersect(const Line &a, const Line &b){
        auto ab = ccw(a.st, a.ed, b.st) * ccw(a.st, a.ed, b.ed);
        auto cd = ccw(b.st, b.ed, a.st) * ccw(b.st, b.ed, a.ed);
        if(!ab && !cd) // c a-b d 혹은 a c-d b
            return (min(b.st, b.ed) <= max(a.st, a.ed)) &&
(min(a.st, a.ed) <= max(b.st, b.ed));
        return (ab <= 0) && (cd <= 0);
    } //직선 a와 b가 교차? (정수)

    // Float Geometry
    typedef geom_t double
    const double EPS = 1e-9; //const ld LEPS = 1e-16;
    inline int diff(double lhs, double rhs) {
        if (lhs - EPS < rhs && rhs < lhs + EPS) return 0;
        return (lhs < rhs) ? -1 : 1;
    } //실수 lhs < rhs
    inline bool is_between(double check, double a, double b) {
        if (a < b) return (a - EPS < check && check < b + EPS);
        else return (b - EPS < check && check < a + EPS);
    } //실수 a < check < b
    inline int sign(double x) { return diff(x, 0); }
    inline int ccw(const Point& a, const Point& b, const Point& c) {
        return diff(outer(b - a, c - a), 0);
    }

    // Equation Geomentry (with Float)
    inline double norm(Point p) { return sqrt(p.xx*p.xx + p.yy*p.yy); }
    inline Point rot90(Point p) { return {-p.y, p.x}; }

```

```

    struct Circle {
        Point cp; double r;
    };
    pair<int, Point> intersect(Circle c1, Circle c2) {
        Point p1 = c1.cp, p2 = c2.cp;
        double r1 = c1.r, r2 = c2.r;
        Point lineVec = p2 - p1;
        double L = norm(lineVec);
        if (L > r1 + r2) { // no-intersection
            return {0, {0, 0}};
        } else if (L <= abs(r1 - r2)) { // include
            return {-1, {0, 0}};
        }
        double x = (r1 * r1 - r2 * r2 + L * L) / (2 * L);
        Point mp = p1 + lineVec * (x / L);
        double y = sqrt(r1 * r1 - x * x);
        if (y <= 0) { // meet 1
            return {1, {mp, mp}};
        } else { // meet 2
            return {2, {
                mp + rot90(lineVec) * (y / L),
                mp - rot90(lineVec) * (y / L)
            }};
        }
    }
    pair<int, Point> intersect(Line ln, Circle cc) {
        Point lineVec = ln.p2 - ln.p1;
        double a = inner(lineVec, lineVec);
        double b = 2 * inner(lineVec, ln.p1 - cc.cp);
        double c = inner(cc.cp, cc.cp) + inner(ln.p1, ln.p1) - 2 *
inner(cc.cp, ln.p1) - cc.r * cc.r;
        const auto D = b*b - 4*a*c;
        if (sign(D) < 0) { // no-intersection
            return {0, {0, 0}};
        } else if (sign(D) == 0) { // meet 1
            double ret = -b / (2 * a);
            return {1, {ret, ret}};
        } else { // meet 2
            double ret1 = (-b + sqrt(D)) / (2 * a);
            double ret2 = (-b - sqrt(D)) / (2 * a);
            if (ret1 > ret2) swap(ret1, ret2);
            return {2, {ret1, ret2}};
        }
    }

```

```

}

bool incLine(Line ln, vector<Circle> &cir) {
    vector<pair<ld, int>> pene;
    pene.emplace_back(0, 0); // Segment start-point
    pene.emplace_back(1, 0); // Segment end-point
    for(auto &c : cir) {
        Circle tmp = c;
        tmp.r += (EPS * tmp.r);
        auto ret = intersect(ln, tmp);
        if (ret.xx == 2) {
            pene.emplace_back(ret.yy.xx, -1); // -1,1 : cover point
            pene.emplace_back(ret.yy.yy, 1); // 1,-1 : exclude point
        }
    }
    sort(all(pene)); // Through the line
    int depth = 0;
    bool flag = false;
    for(auto &s : pene) {
        depth -= s.yy;
        if (s.yy == 0) flag = !flag; // Line segment start or end
        if (flag && depth <= 0) return false;
    }
    return true;
}

struct Form { // ax+b (with Integer Geometry)
    // y = (dir)x + intY
    double dir, intY;
    int flag; // 0: normal, 1: vertical, 2: horizon
    Form() {}
    Form(const Line& ln) {
        if (diff(ln.st.xx, ln.ed.xx) == 0) {
            intY = ln.st.xx;
            flag = 1;
        } else if (diff(ln.st.yy, ln.ed.yy) == 0) {
            intY = ln.st.yy;
            flag = 2;
        } else {
            dir = (ln.st.yy - ln.ed.yy) / (ln.st.xx - ln.ed.xx);
            intY = ln.st.yy - dir * ln.st.xx;
            flag = 0;
        }
    }
}

```

```

    bool assn(Point pp) {
        switch (flag) {
            case 0: return diff(pp.yy, dir*pp.xx + intY) == 0;
            case 1: return diff(pp.xx, intY) == 0;
            case 2: return diff(pp.yy, intY) == 0;
        }
    }
};

pair<bool, Point> intersect(Form f1, Form f2) {
    if (f1.flag > f2.flag) swap(f1, f2);
    int flag = (1 << f1.flag) + (1 << (f2.flag + 3));
    switch (flag) {
        case 18:
        case 36: { // vertical & vertical // horizon & horizon
            return {!diff(f1.intY, f2.intY), {f1.intY, f2.intY}};
        }
        case 34: { // vertical & horizon
            return {true, {f1.intY, f2.intY}};
        }
        case 17: { // normal & vertical
            return {true, {
                f2.intY,
                f1.dir * f2.intY + f1.intY // assign
            }};
        }
        case 33: { // normal & horizon
            return {true, {
                (f2.intY - f1.intY) / f1.dir, // assign
                f2.intY
            }};
        }
        case 9: break; // normal & normal
    }
    if (diff(f1.dir, f2.dir) == 0) { // parallel
        return {diff(f1.intY, f2.intY) == 0, {f1.intY, f2.intY}};
    }
    Point ret;
    assert(diff(f1.dir, f2.dir));
    ret.xx = (f2.intY - f1.intY) / (f1.dir - f2.dir);
    ret.yy = f1.dir * ret.xx + f1.intY;
    return {true, ret};
}

```

8.2 볼록 껍질(컨벡스 헬)

Usage: 가장 왼쪽, 같을시 아래쪽 점을 시작으로 시계방향으로 탐색. 일직선의 점을 남기고 싶으면, 등호를 뺀다

complexity: $\mathcal{O}(N \log N)$

```
vector<Point> convex_hull(vector<Point>& dat) {
    if (dat.size() <= 2) return dat;
    vector<Point> upper, lower;
    sort(dat.begin(), dat.end(), [](const Point& a, const Point& b) {
        return (a.xx == b.xx) ? a.yy < b.yy : a.xx < b.xx;
    });
    for (const auto& p : dat) {
        while (upper.size() >= 2 && ccw(*++upper.rbegin(), *upper.rbegin(), p) >= 0) upper.pop_back();
        while (lower.size() >= 2 && ccw(*++lower.rbegin(), *lower.rbegin(), p) <= 0) lower.pop_back();
        upper.emplace_back(p);
        lower.emplace_back(p);
    }
    upper.insert(upper.end(), ++lower.rbegin(), --lower.rend());
    return upper;
}
```

8.3 회전하는 캘리퍼스 - 볼록 껍질에서 가장 먼 점

Usage: 볼록 껍질의 모든 점에 대해 가장 먼 점을 구해준다

complexity: $\mathcal{O}(N)$

```
//+ 정수 컨벡스 헬
void calipers(vector<point>& pt) {
    sort(pt.begin(), pt.end(), [](const point& a, const point& b) {
        return (a.xx == b.xx) ? a.yy < b.yy : a.xx < b.xx;
    });
    vector<point> up, lo;
    for (const auto& p : pt) {
        while (up.size() >= 2 && ccw(*++up.rbegin(), *up.rbegin(), p) >= 0) up.pop_back();
        while (lo.size() >= 2 && ccw(*++lo.rbegin(), *lo.rbegin(), p) <= 0) lo.pop_back();
        up.emplace_back(p);
        lo.emplace_back(p);
    }
    for (int i = 0, j = (int)lo.size() - 1; i + 1 < up.size() || j > 0; ) {
```

```
// DO WHAT YOU WANT with up[i], lo[j]
if (i + 1 == up.size()) --j;
else if (j == 0) ++i;
else if ((ll)(up[i+1].yy-up[i].yy)*(lo[j].xx-lo[j-1].xx)
    > (ll)(up[i+1].xx-up[i].xx)*(lo[j].yy-lo[j-1].yy)) ++i;
else --j;
}
}
```

9 기타

9.1 Mo's Algorithm + Sqrt Decomposition

Usage: $\mathcal{O}(N + Q)\sqrt{N}$

complexity: $\mathcal{O}(N + Q)\sqrt{N}$

```
int n, ans[MAX];
vector<pair<pii, int>> query;
inline void add(int k){//현 구간에 k를 하나 추가해준다.
    //do something : ex) if(++cnt[k] == 1) ++ret;
}
inline void pop(int k){//현 구간에서 k를 하나 제거해준다.
    //do something : ex) if(--cnt[k] == 0) --ret;
}
void MOquery(){
    int sqrtN = sqrt(n);
    sort(query.begin(), query.end(), [](auto &a, auto &b){
        int Ln = a.first.first/sqrtN, Rn = b.first.first/sqrtN;
        if(Ln != Rn) return Ln < Rn;
        //부등호를 뒤집으면 더 빨리질수도 (휴리스틱)
        if(Ln&1) return a.first.second > b.first.second;
        else return a.first.second < b.first.second;
    });
    int L = 0, R = 0; // [0,0]부터 시작해 스위핑 해주자.
    add(arr[0]);
    for(auto &P : query){
        int nL = P.first.first, nR = P.first.second;
        if(nL < L){//왼쪽으로 밀면서 추가 [nL, L] 추가
            for(int i=L-1; i>=nL; --i)
                add(arr[i]);
        }
        if(R < nR){//오른쪽으로 밀면서 (R, nR] 추가
            for(int i=R+1; i<=nR; ++i)
                add(arr[i]);
```

```

        add(arr[i]);
    }
    if(L < nL){//오른쪽으로 밀면서 [L, nL) 제거
        for(int i=L; i<nL; ++i)
            pop(arr[i]);
    }
    if(nR < R){//왼쪽으로 밀면서 (nR, R] 제거
        for(int i=R; i>nR; --i)
            pop(arr[i]);
    }
    L = nL; //구간, 쿼리의 답 업데이트
    R = nR;
    ans[P.second] = ret;
}
}

```

9.2 삼분 탐색

Usage: 순증가-순감소 혹은 순감소 - 순증가 하는 볼록 함수

```

int l=0, r=n-1; //아래로 볼록 (최소값) : >, 위로 볼록 (최대값) : <
while(l+3<=r) { //실수는 이 부분을 반복문으로
    int x = (2*l+r)/3, y = (l+2*r)/3;
    if(f(x) > f(y)) l = x;
    else r = y;
}
int ans = INT_MAX;
for(int i=l; i<=r; ++i) ans = min(ans, f(i));

```

9.3 유용한 이론들

- 카탈란 수

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900

$$C_n = \binom{n}{2} / (n+1);$$

- 길이가 $2n$ 인 올바른 팔호 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 0이 아니면 첫번째, 0이면 두번째 플레이어가 승리.
- Grundy Number : 어떤 상황의 Grundy Number는, 가능한 다음 상황들의 Grundy Number를 모두 모은 다음, 그 집합에 포함되지 않는 가장 작은 수가 현재 state의 Grundy Number가 된다. 만약 다음 state가 독립된 여러개의 state들로 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
- 훌의 결혼 정리 : 이분그래프(L-R)에서, 모든 L을 매칭하는 필요충분 조건 = L에서 임의의 부분집합 S를 골랐을 때, 반드시 (S 의 크기) \leq (S 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
- 맨하탄 기하에서 모든 점과 거리가 최소인 점 : (x중앙값, y중앙값) (짝수개면 둘 사이 아무거나)

9.4 자주 쓰이는 테크닉들

```

namespace fio {
    //FAST I/O
    const int BSIZE = 524288; char buffer[BSIZE]; int p = BSIZE;
    inline char readChar() {
        if(p == BSIZE) { fread(buffer, 1, BSIZE, stdin); p = 0; }
        return buffer[p++];
    }
}

```

```

int readInt() {
    char c = readChar();
    while ((c < '0' || c > '9') && c != '-') c = readChar();
    int ret = 0; bool neg = c == '-';
    if (neg) c = readChar();
    while (c >= '0' && c <= '9') {
        ret = ret * 10 + c - '0'; c = readChar();
    }
    return neg ? -ret : ret;
}
int x = fio::readInt();

```

//strtok 파싱

```

string input = "how to,split!string";
char *token = strtok(input, " ,!");
while(token){ cout << token << '\n'; token = strtok(0, " ,!"); }

```

//신발끈 정리 - 변이 교차하지 않는 다각형의 넓이

```

for(int i=0; i<n; ++i)
    area += X[i]*Y[(i+1)%len] - X[i]*Y[(i+len-1)%len];
return abs(area/2);

```

//비트마스크

```

__builtin_popcount(bits) // 켜져있는 비트의 개수
__builtin_ctz(bits) //최하위 비트의 인덱스
(bits & -bits) //최하위 비트의 실제 값
for(int i=bits; i; i=(i-1)&bits) //모든 부분집합 순회

```

```

bitset<64> B(42); // ...00101010
string S = B.to_string();
bitset<64> B2("101010");
cout << B2.to_ullong(); // 42
//string stream
int num;
stringstream stream;
string string = "1 2 3 a 4";
stream.str(string);
while( stream >> num )
    cout << num << ' '; //1 2 3

```

//cout 자리수 설정

```

cout.precision(X); cout << fixed;

```

```

// comparator overload
auto cmp = [] (seg a, seg b){return a.func() < b.func(); };
set<seg, decltype(cmp)> s(cmp);
map<seg, int, decltype(cmp)> mp(cmp);
priority_queue<seg, vector<seg>, decltype(cmp)> pq(cmp); // max heap

// hash func overload
struct hasher {
    size_t operator()(const pii &x) const{
        return hash<ll>()((ll)x.first)^((ll)x.second)<<32));
    }
    size_t operator()(const state &s) const {
        size_t res = 17;
        res = res * 31 + hash<string>()(s.pro);
        res = res * 31 + hash<int>()(s.x);
        res = res * 31 + hash<int>()(s.y);
        res = res * 31 + hash<char>()(s.prev_p);
        return res;
    }
};
unordered_map<state, int, hasher> dp;

//safe rand
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
cout << uniform_int_distribution<int>(0, 10)(rng); //return 0,...,10

//0.9초동안 돌리는 랜덤
double start_time = clock();
while(1){
    //...
    if((clock() - start_time) / CLOCKS_PER_SEC >= 0.99) break; // 0.99s
}

//gcc 최적화
#pragma GCC target ("avx2")
#pragma GCC optimization ("O3")
#pragma GCC optimization ("unroll-loops")

```