Team Note of 팀명

이름1, 이름2, 이름3

Compiled on October 20, 2023

Contents

1 정렬 / 이분 탐색 관련

1.1 정렬 비교 함수 구현

```
struct Point{
    int x, y;
    // 방법 1: 연산자 오버로딩
    // x좌표 오름차순, x좌표 같으면 y좌표 오름차순
    // sort(시작 주소, 끝 주소)
    bool operator < (const Point &p) const {
        if(x != p.x) return x < p.x;
        else return y < p.y;
    }
};

// 방법 2: 비교 함수 구현
// sort(시작 주소, 끝 주소, Compare)
bool Compare(const Point &a, const Point &b){
    if(a.x != b.x) return a.x < b.x;
    else return a.y < b.y;
}

vector<Point> V;
V.push_back({1, 2});
V.push_back({1, 1});
V.push_back({2, 3});
sort(V.begin(), V.end()); // 방법 1
sort(V.begin(), V.end(), Compare); // 방법 2
```

1.2 좌표 압축

Time Complexity: $O(N \log N)$

```
// 원소의 대소관계를 유지하면서 [0, N) 범위의 수로 압축함
// ex. {50, 31, 24, 10, 46, 10} -> {4, 2, 1, 0, 3, 0}
int N = 6, A[6] = {50, 31, 24, 10, 46, 10};
vector<int> C;
for(int i=0; i<N; i++) C.push_back(A[i]);
sort(C.begin(), C.end());
C.erase(unique(C.begin(), C.end()), C.end());
for(int i=0; i<N; i++){
    A[i] = lower_bound(C.begin(), C.end(), A[i]) - C.begin();
}
for(int i=0; i<N; i++) cout << A[i] << " "; // 4 2 1 0 3 0
```

1.3 이분 탐색 관련 STL

```
vector<int> v = {3, 5, 7, 7, 7, 10};

// lower_bound: x 이상인 가장 빠른 위치
// 배열은 정렬되어 있어야 함, O(log N)
for(int i=3; i<=8; i++){
    cout << lower_bound(v.begin(), v.end(), i) - v.begin() << " ";
} // 0 1 1 2 2 5
cout << "\n";

// upper_bound: x 초과인 가장 빠른 위치
// 배열은 정렬되어 있어야 함, O(log N)
for(int i=3; i<=8; i++){
    cout << upper_bound(v.begin(), v.end(), i) - v.begin() << " ";
} // 1 1 2 2 5 5
cout << "\n";

// binary_search: x가 있으면 true, 없으면 false
// 배열은 정렬되어 있어야 함, O(log N)
for(int i=3; i<=8; i++){
    cout << binary_search(v.begin(), v.end(), i) << " ";
} // 1 0 1 0 1 0
cout << "\n";

vector<int> a = {1, 3, 5, 2, 4, 6};

// nth_element(a.begin(), a.begin()+k, a.end())
// 정렬했을 때 a[k]에 와야 하는 수가 a[k]에 옴
// a[k] 미만의 수는 모두 a[0..k-1]으로 이동
// a[k] 초과의 수는 모두 a[k+1..]으로 이동
// 평균 시간 복잡도 O(N), 최악 시간 복잡도 O(N log N)
nth_element(a.begin(), a.begin()+3, a.end());
for(auto i : a) cout << i << " "; // a b c 4 d e
// a b c <= 4, d e >= 4
```

2 그래프/트리 (Easy)

2.1 인접 리스트, DFS, BFS

Time Complexity: $O(V + E)$

```
int N, M, C[1010];
vector<int> G[1010];
void AddEdge(int s, int e){ G[s].push_back(e); }
void DFS(int v){
    cout << v << " ";
    C[v] = 1;
    for(auto i : G[v]) if(!C[i]) DFS(i);
}
void BFS(int s){
    queue<int> Q;
    Q.push(s); C[s] = 1;
    while(!Q.empty()){
        int v = Q.front(); Q.pop();
        cout << v << " ";
        for(auto i : G[v]) if(!C[i]) Q.push(i), C[i] = 1;
    }
}
```

2.2 위상 정렬

Time Complexity: $O(V + E)$

```
int N, M, In[1010];
vector<int> G[1010];
void AddEdge(int s, int e){ G[s].push_back(e); In[e]++; }
void TopSort(){
    queue<int> Q;
    for(int i=1; i<=N; i++) if(!In[i]) Q.push(i);
    while(!Q.empty()){
        int v = Q.front(); Q.pop();
        cout << v << " ";
        for(auto i : G[v]) if(!--In[i]) Q.push(i);
    }
}
```

2.3 최단 거리 - Floyd Warshall

Time Complexity: $O(V^3)$

```
int N, G[111][111];
void Init(){ // 시작 전에 호출해야 함
    memset(G, 0x3f, sizeof G);
    for(int i=1; i<=N; i++) G[i][i] = 0;
}
// s에서 e로 가는 가중치 w 간선 추가
void AddEdge(int s, int e, int w){
    G[s][e] = min(G[s][e], w);
}
void Run(){
    for(int k=1; k<=N; k++)
        for(int i=1; i<=N; i++)
            for(int j=1; j<=N; j++)
                G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
}
```

2.4 최단 거리 - Dijkstra

Time Complexity: $O(E \log E)$

```
ll D[505050], P[505050];
vector<pair<ll,ll>> G[505050]; // {정점, 가중치}
// 주의: 가중치 >= 0, 음수 있으면 bellman ford 사용

// s -> t 최단 경로 출력
void Dijkstra(int s, int t){
    memset(D, 0x3f, sizeof D);
    priority_queue<pair<ll,ll>, vector<pair<ll,ll>>, greater<>> Q;
    Q.emplace(D[s]=0, s);
    while(!Q.empty()){
        auto [c,v] = Q.top(); Q.pop();
        if(c == D[v]) for(auto [i,w] : G[v]) if(D[i] > c + w) Q.emplace(D[i]=c+w, i), P[i]
        = v;
    }
    vector<int> path;
    for(int i=t; i!=s; i=P[i]) path.push_back(i);
    path.push_back(s);
    reverse(path.begin(), path.end());
    for(auto i : path) cout << i << " ";
}
```

2.5 최단 거리 - Bellman Ford

Time Complexity: $O(VE)$

```
int N, M; ll D[555]; // 주의: 웬만하면 long long으로 잡는 게 좋음
vector<tuple<int,int,ll>> E; // {from, to, weight}
void AddEdge(int s, int e, int w){
    E.emplace_back(s, e, w);
}
// s에서 도달 가능한 음수 사이클 있으면 false 반환
bool Run(int s){
    memset(D, 0x3f, sizeof D);
    ll INF = D[0];
    D[s] = 0;
    for(int iter=1; iter<=N; iter++){
        bool changed = false;
        for(auto [u,v,w] : E){
            if(D[u] == INF) continue;
            if(D[v] > D[u] + w) D[v] = D[u] + w, changed = true;
        }
        if(iter == N && changed) return false;
    }
    return true;
}
```

2.6 유니온 파인드 + 최소 신장 트리(Kruskal)

Time Complexity: UF: 연산마다 $O(logN)$, MST: $O(E \log E)$

```
int N, M, P[10101];
int Find(int v){ return v == P[v] ? v : P[v] = Find(P[v]); }
bool Merge(int u, int v){
    u = Find(u); v = Find(v);
    if(u == v) return false;
    P[u] = v; return true;
}
int main(){
    cin >> N >> M;
    vector<tuple<int,int,int>> E; // {weight, from, to}
    for(int i=1,u,v,w; i<=M; i++){
        cin >> u >> v >> w;
        E.emplace_back(w, u, v);
    }
    sort(E.begin(), E.end());
    for(int i=1; i<=N; i++) P[i] = i;
    long long res = 0;
    for(auto [w,u,v] : E) if(Merge(u, v)) res += w;
    cout << res;
}
```

3 자료구조

3.1 세그먼트 트리

Time Complexity: $O(\log N)$

```
// SZ: N보다 크거나 같은 2^k 꼴의 수
// 13만 -> 1 << 17 (131072), 26만 -> 1 << 18 (262144)
// 52만 -> 1 << 19 (524288), 100만 -> 1 << 20 (1048576)
constexpr int SZ = 1 << 20;
ll T[SZ<<1];

void Set(int x, ll v){ // x번째 수를 v로 지정, x는 0 이상 SZ 미만
    x += SZ; T[x] = v;
    while(x /= 2) T[x] = T[x*2] + T[x*2+1];
}

ll Sum(int l, int r){ // [l, r] 구간의 합
    ll res = 0;
    for(l+=SZ, r+=SZ; l<=r; l/=2, r/=2){
        if(l % 2 == 1) res += T[l++];
        if(r % 2 == 0) res += T[r--];
    }
    return res;
}
```

3.2 세그먼트 트리 + 레이지 프로퍼게이션

Time Complexity: $O(\log N)$

```
// SZ: N보다 크거나 같은 2^k 꼴의 수
// 13만 -> 1 << 17 (131072), 26만 -> 1 << 18 (262144)
// 52만 -> 1 << 19 (524288), 100만 -> 1 << 20 (1048576)
constexpr int SZ = 1 << 20;
ll T[SZ<<1], L[SZ<<1];

void Push(int node, int s, int e){
    if(L[node] == 0) return;
    T[node] += (e - s + 1) * L[node];
    if(s != e) L[node*2] += L[node], L[node*2+1] += L[node];
    L[node] = 0;
}

// [l, r]번째 수에 v를 더함, 0 <= l <= r < SZ
void RangeAdd(int l, int r, ll v, int node=1, int s=0, int e=SZ-1){
    Push(node, s, e);
    if(r < s || e < l) return;
    if(l <= s && e <= r){ L[node] += v; Push(node, s, e); return; }
    int m = (s + e) / 2;
    RangeAdd(l, r, v, node*2, s, m);
    RangeAdd(l, r, v, node*2+1, m+1, e);
    T[node] = T[node*2] + T[node*2+1];
}

// [l, r]번째 수의 합을 구함
ll RangeSum(int l, int r, int node=1, int s=0, int e=SZ-1){
    Push(node, s, e);
    if(r < s || e < l) return 0;
    if(l <= s && e <= r) return T[node];
    int m = (s + e) / 2;
    return RangeSum(l, r, node*2, s, m) + RangeSum(l, r, node*2+1, m+1, e);
}
```

3.3 Convex Hull Trick

Usage: call init() before use

```
// 직선 개수 N, 쿼리 횟수 Q일 때 O(N+Q)
// (최댓값 쿼리) 삽입하는 직선의 기울기는 단조 증가해야 함
// (최솟값 쿼리) 삽입하는 직선의 기울기는 단조 감소해야 함
// 쿼리를 하는 x좌표는 단조 증가해야 함
struct Line{
  ll a, b, c; // y = ax + b, c = line index
  Line(ll a, ll b, ll c) : a(a), b(b), c(c) {}
  ll f(ll x){ return a * x + b; }
};
vector<Line> v; int pv;
void init(){ v.clear(); pv = 0; }
int chk(const Line &a, const Line &b, const Line &c) const {
  return (__int128_t)(a.b - b.b) * (b.a - c.a) <= (__int128_t)(c.b - b.b) * (b.a - a.a);
}
void insert(Line l){
  if(v.size() > pv && v.back().a == l.a){
    // if min query, then if(l.b > v.back().b)
    if(l.b < v.back().b) l = v.back();
    v.pop_back();
  }
  while(v.size() >= pv+2 && chk(v[v.size()-2], v.back(), l)) v.pop_back();
  v.push_back(l);
}
p query(ll x){ // if min query, then v[pv].f(x) >= v[pv+1].f(x)
  while(pv+1 < v.size() && v[pv].f(x) <= v[pv+1].f(x)) pv++;
  return {v[pv].f(x), v[pv].c};
}
///// line container start (max query) /////
// 직선 개수 N 쿼리 개수 Q일 때 O((N+Q) log N)
// 단조성 제약 조건 없음
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; }
}; // (for doubles, use inf = 1/.0, div(a,b) = a/b)
struct LineContainer : multiset<Line, less<>> {
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { return a / b - ((a ^ b) < 0 && a % b); } // floor
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    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 k, ll m) {
    auto z = insert({k, m, 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;
  }
};
```

3.4 퍼시스턴트 세그먼트 트리

Usage: call init(root[0], s, e) before use

```
struct PSTNode{
  PSTNode *l, *r; int v;
  PSTNode(){ l = r = nullptr; v = 0; }
};
PSTNode *root[101010];
PST(){ memset(root, 0, sizeof root); } // constructor
void init(PSTNode *node, int s, int e){
  if(s == e) return;
  int m = s + e >> 1;
  node->l = new PSTNode; node->r = new PSTNode;
  init(node->l, s, m); init(node->r, m+1, e);
}
void update(PSTNode *prv, PSTNode *now, int s, int e, int x){
  if(s == e){ now->v = prv ? prv->v + 1 : 1; return; }
  int m = s + e >> 1;
  if(x <= m){
    now->l = new PSTNode; now->r = prv->r;
    update(prv->l, now->l, s, m, x);
  }
  else{
    now->r = new PSTNode; now->l = prv->l;
    update(prv->r, now->r, m+1, e, x);
  }
  int t1 = now->l ? now->l->v : 0;
  int t2 = now->r ? now->r->v : 0;
  now->v = t1 + t2;
}
int kth(PSTNode *prv, PSTNode *now, int s, int e, int k){
  if(s == e) return s;
  int m = s + e >> 1, diff = now->l->v - prv->l->v;
  if(k <= diff) return kth(prv->l, now->l, s, m, k);
  else return kth(prv->r, now->r, m+1, e, k-diff);
}
```

4 그래프/트리 (Hard)

4.1 SCC - Kosaraju

Time Complexity: $O(V+E)$

```
int N, M, C[10101]; // C[i] = i번 정점이 속한 SCC 번호
vector<int> G[10101], R[10101], V;
vector<vector<int>> S; // 각 SCC에 속한 정점 목록
void AddEdge(int s, int e){
    G[s].push_back(e);
    R[e].push_back(s);
}
void DFS1(int v){
    C[v] = -1;
    for(auto i : G[v]) if(!C[i]) DFS1(i);
    V.push_back(v);
}
void DFS2(int v, int c){
    C[v] = c; S.back().push_back(v);
    for(auto i : R[v]) if(C[i] == -1) DFS2(i, c);
}
int GetSCC(){ // SCC 개수 반환
    for(int i=1; i<=N; i++) if(!C[i]) DFS1(i);
    reverse(V.begin(), V.end());
    int cnt = 0;
    for(auto i : V) if(C[i] == -1) S.emplace_back(), DFS2(i, cnt++);
    return cnt;
} // 각 SCC는 위상 정렬 순서대로 번호 매겨져 있음
```

4.2 BCC - Tarjan

Time Complexity: $O(V+E)$

```
// 1-based, 다른 거 호출하기 전에 tarjan 먼저 호출해야 함
vector<int> G[MAX_V]; int In[MAX_V], Low[MAX_V], P[MAX_V];
void addEdge(int s, int e){ G[s].push_back(e); G[e].push_back(s); }
void tarjan(int n){ /// Pre-Process
  int pv = 0;
  function<void(int,int)> dfs = [&pv,&dfs](int v, int b){
    In[v] = Low[v] = ++pv; P[v] = b;
    for(auto i : G[v]){
      if(i == b) continue;
      if(!In[i]) dfs(i, v), Low[v] = min(Low[v], Low[i]); else Low[v] = min(Low[v], In[i]);
    }
  };
  for(int i=1; i<=n; i++) if(!In[i]) dfs(i, -1);
}
vector<int> cutVertex(int n){
  vector<int> res; array<char,MAX_V> isCut; isCut.fill(0);
  function<void(int)> dfs = [&dfs,&isCut](int v){
    int ch = 0;
    for(auto i : G[v]){
      if(P[i] != v) continue; dfs(i); ch++;
```

```
      if(P[v] == -1 && ch > 1) isCut[v] = 1; else if(P[v] != -1 && Low[i] >= In[v])
      isCut[v]=1;
    }
  };
  for(int i=1; i<=n; i++) if(P[i] == -1) dfs(i);
  for(int i=1; i<=n; i++) if(isCut[i]) res.push_back(i);
  return move(res);
}
vector<PII> cutEdge(int n){
  vector<PII> res;
  function<void(int)> dfs = [&dfs,&res](int v){
    for(int t=0; t<G[v].size(); t++){
      int i = G[v][t]; if(t != 0 && G[v][t-1] == G[v][t]) continue;
      if(P[i] != v) continue; dfs(i);
      if((t+1 == G[v].size() || i != G[v][t+1]) && Low[i] > In[v])
      res.emplace_back(min(v,i), max(v,i));
    }
  };
  for(int i=1; i<=n; i++) sort(G[i].begin(), G[i].end()); // multi edge -> sort
  for(int i=1; i<=n; i++) if(P[i] == -1) dfs(i);
  return move(res); // sort(all(res));
}
vector<int> BCC[MAX_V]; // BCC[v] = components which contains v
void vertexDisjointBCC(int n){ // allow multi edge, not allow self loop
  int cnt = 0; array<char,MAX_V> vis; vis.fill(0);
  function<void(int,int)> dfs = [&dfs,&vis,&cnt](int v, int c){
    vis[v] = 1; if(c > 0) BCC[v].push_back(c);
    for(auto i : G[v]){
      if(vis[i]) continue;
      if(In[v] <= Low[i]) BCC[v].push_back(++cnt), dfs(i, cnt); else dfs(i, c);
    }
  };
  for(int i=1; i<=n; i++) if(!vis[i]) dfs(i, 0);
  for(int i=1; i<=n; i++) if(BCC[i].empty()) BCC[i].push_back(++cnt);
}
```

4.3 최대 유량 - Dinic

Time Complexity: $O(V^2E)$, 모든 간선의 용량이 1 이면 $O(\min(V^{2/3}, E^{1/2})E)$

```
// Directed Graph이면 add_edge(s, e, c)
// Undirected Graph이면 add_edge(s, e, c, c)
template<typename flow_t, flow_t MAX_U=(1<<30)>
struct Dinic{ // 0-based
  struct edge_t{ int v, r; flow_t c, f; };
  int n;
  vector<vector<edge_t>> g;
  vector<int> lv, idx;
  Dinic(int n) : n(n) { clear(); }
  void clear(){
    g = vector<vector<edge_t>>(n);
    lv = vector<int>(n, 0);
    idx = vector<int>(n, 0);
  }
  void add_edge(int s, int e, flow_t c1, flow_t c2=flow_t(0)){
    g[s].push_back({e, (int)g[e].size(), c1, 0});
    g[e].push_back({s, (int)g[s].size()-1, c2, 0});
  }
  bool bfs(int s, int t, flow_t limit=1){
    fill(lv.begin(), lv.end(), 0);
    queue<int> que; que.push(s); lv[s] = 1;
    while(!que.empty()){
      int v = que.front(); que.pop();
      for(const auto &e : g[v]) if(!lv[e.v] && e.c - e.f >= limit) que.push(e.v), lv[e.v] =
      lv[v] + 1;
    }
    return lv[t] != 0;
  }
  flow_t dfs(int v, int t, flow_t fl=MAX_U){
    if(v == t || fl == flow_t(0)) return fl;
    for(int &i=idx[v]; i<g[v].size(); i++){
      auto &e = g[v][i];
      if(lv[e.v] != lv[v] + 1 || e.c - e.f == flow_t(0)) continue;
      flow_t now = dfs(e.v, t, min(fl, e.c - e.f));
      if(now == flow_t(0)) continue;
      e.f += now; g[e.v][e.r].f -= now;
      return now;
    }
    return 0;
  }
  flow_t maximum_flow(int s, int t){
    flow_t flow = 0, augment = 0;
    while(bfs(s, t)){
      fill(idx.begin(), idx.end(), 0);
      while((augment=dfs(s, t)) != flow_t(0)) flow += augment;
    }
    return flow;
  }
  // {최소 컷 비용, s와 같은 집합, t와 같은 집합, 절단 간선}
  tuple<flow_t, vector<int>, vector<int>, vector<pair<int,int>>> minimum_cut(int s, int t){
    flow_t flow = maximum_flow(s, t);
    vector<int> a, b;
    vector<pair<int,int>> edges;
    bfs(s, t, 1);
    for(int i=0; i<n; i++) (lv[i] ? a : b).push_back(i);
    for(auto i : a) for(auto e : g[i]) if(e.c != flow_t(0) && !lv[e.v])
    edges.emplace_back(i, e.v);
    return {flow, a, b, edges};
  }
};
```

4.4 MCMF

```
// 유량을 k 만큼 흘리는 경우: run(src, snk, k)
// 유량을 최대한 많이 흘리는 경우: run(src, snk)
template<typename flow_t=int, typename cost_t=long long, flow_t MAX_U=(1<<30), cost_t
MAX_C=(1LL<<60)>
```

```
struct MinCostFlow{ // 0-based
    struct edge_t{ int v, r; flow_t c; cost_t d; };
    int n;
    vector<vector<edge_t>> g;
    vector<int> prv, idx, chk;
    vector<cost_t> dst;
    MinCostFlow(int n) : n(n) { clear(); }
    void clear(){
        g = vector<vector<edge_t>>(n);
        prv = idx = chk = vector<int>(n);
        dst = vector<cost_t>(n);
    }
    void add_edge(int s, int e, flow_t c, cost_t d){
        g[s].push_back({e, (int)g[e].size(), c, d});
        g[e].push_back({s, (int)g[s].size()-1, 0, -d});
    }
    bool find_path(int s, int t){
        fill(chk.begin(), chk.end(), 0);
        fill(dst.begin(), dst.end(), MAX_C);
        queue<int> que; que.push(s); dst[s] = 0; chk[s] = 1;
        while(!que.empty()){
            int v = que.front(); que.pop(); chk[v] = 0;
            for(int i=0; i<g[v].size(); i++){
                const auto &e = g[v][i];
                if(e.c > 0 && dst[e.v] > dst[v] + e.d){
                    dst[e.v] = dst[v] + e.d; prv[e.v] = v; idx[e.v] = i;
                    if(!chk[e.v]) que.push(e.v), chk[e.v] = 1;
                }
            }
        }
        return dst[t] < MAX_C;
    }
    pair<flow_t, cost_t> augment(int s, int t, flow_t k=-1){
        if(!find_path(s, t)) return {0, 0};
        flow_t fl = MAX_U;
        for(int i=t; i!=s; i=prv[i]) fl = min(fl, g[prv[i]][idx[i]].c);
        if(k != -1) fl = min(fl, k);
        for(int i=t; i!=s; i=prv[i]){
            g[prv[i]][idx[i]].c -= fl;
            g[i][g[prv[i]][idx[i]].r].c += fl;
        }
        return {fl, fl * dst[t]};
    }
    pair<flow_t, cost_t> run(int s, int t, flow_t k=-1){
        flow_t flow = 0; cost_t cost = 0;
        while(true){
            auto [fl,cst] = augment(s, t, k);
            if(fl == 0) break;
            flow += fl; cost += cst;
            if(k != -1) k -= fl;
        }
        return {flow, cost};
    }
};
```

4.5 이분 매칭 - Hopcroft Karp

Time Complexity: $O(E\sqrt{V})$

```
// n: 왼쪽 정점 개수, m: 오른쪽 정점 개수, 0-based
struct HopcroftKarp{
  int n, m;
  vector<vector<int>> g;
  vector<int> dst, le, ri;
  vector<char> visit, track;
  HopcroftKarp(int n, int m) : n(n), m(m) { clear(); }
  void clear(){
    g = vector<vector<int>>(n); dst = vector<int>(n, 0);
    le = vector<int>(n, -1); ri = vector<int>(m, -1);
    visit = vector<char>(n, 0); track = vector<char>(n+m, 0);
  }
  void add_edge(int s, int e){ g[s].push_back(e); }
  bool bfs(){
    bool res = false;
    queue<int> que;
    fill(dst.begin(), dst.end(), 0);
    for(int i=0; i<n; i++) if(le[i] == -1) que.push(i), dst[i] = 1;
    while(!que.empty()){
      int v = que.front(); que.pop();
      for(auto i : g[v]){
        if(ri[i] == -1) res = true;
        else if(!dst[ri[i]]) dst[ri[i]] = dst[v] + 1, que.push(ri[i]);
      }
    }
    return res;
  }
  bool dfs(int v){
    if(visit[v]) return false;
    visit[v] = 1;
    for(auto i : g[v]){
      if(ri[i] == -1 || !visit[ri[i]] && dst[ri[i]] == dst[v] + 1 && dfs(ri[i])){
        le[v] = i; ri[i] = v; return true;
      }
    }
    return false;
  }
  int maximum_matching(){
    int res = 0;
    fill(le.begin(), le.end(), -1);
    fill(ri.begin(), ri.end(), -1);
    while(bfs()){
      fill(visit.begin(), visit.end(), 0);
      for(int i=0; i<n; i++) if(le[i] == -1) res += dfs(i);
    }
    return res;
  }
  vector<pair<int,int>> maximum_matching_edges(){
```

```
    int matching = maximum_matching();
    vector<pair<int,int>> edges; edges.reserve(matching);
    for(int i=0; i<n; i++) if(le[i] != -1) edges.emplace_back(i, le[i]);
    return edges;
  }
  void dfs_track(int v){
    if(track[v]) return; track[v] = 1;
    for(auto i : g[v]) track[n+i] = 1, dfs_track(ri[i]);
  }
  tuple<vector<int>, vector<int>, int> minimum_vertex_cover(){
    int matching = maximum_matching();
    fill(track.begin(), track.end(), 0);
    for(int i=0; i<n; i++) if(le[i] == -1) dfs_track(i);
    vector<int> lv, rv;
    for(int i=0; i<n; i++) if(!track[i]) lv.push_back(i);
    for(int i=0; i<m; i++) if(track[n+i]) rv.push_back(i);
    assert(lv.size() + rv.size() == matching);
    return {lv, rv, lv.size() + rv.size()};
  }
};
```

4.6 최소 공통 조상(LCA)

Time Complexity: 전처리 $O(N \log N)$, 쿼리 $O(\log N)$

```
int N, Q, D[101010], P[22][101010];
vector<int> G[101010];
void Connect(int u, int v){
    G[u].push_back(v); G[v].push_back(u);
}
void DFS(int v, int b=-1){
    for(auto i : G[v]) if(i != b) D[i] = D[v] + 1, P[0][i] = v, DFS(i, v);
}
int LCA(int u, int v){
    if(D[u] < D[v]) swap(u, v);
    int diff = D[u] - D[v];
    for(int i=0; diff; i++, diff>>=1) if(diff & 1) u = P[i][u];
    if(u == v) return u;
    for(int i=21; i>=0; i--) if(P[i][u] != P[i][v]) u = P[i][u], v = P[i][v];
    return P[0][u];
}
////
// 1. Connect로 간선 추가
// 2. DFS(1) 호출
// 3. 아래 코드 실행
for(int i=1; i<22; i++) for(int j=1; j<=N; j++) P[i][j] = P[i-1][P[i-1][j]];
// 4. LCA(u, v)로 최소 공통 조상 구할 수 있음
```

4.7 Heavy Light Decomposition

Time Complexity: 전처리 $O(N)$, 쿼리 $O(T(N) \log N)$

```
int N, Q, A[SZ], Top[SZ], Par[SZ], Dep[SZ], Sz[SZ], In[SZ];
vector<int> Inp[SZ], G[SZ];

void Connect(int u, int v){
    Inp[u].push_back(v); Inp[v].push_back(u);
}

void DFS0(int v, int b=-1){
    for(auto i : Inp[v]) if(i != b)
        Dep[i] = Dep[v] + 1, Par[i] = v, G[v].push_back(i), DFS0(i, v);
}

void DFS1(int v){
    Sz[v] = 1;
    for(auto &i : G[v]){
        DFS1(i); Sz[v] += Sz[i];
        if(Sz[i] > Sz[G[v][0]]) swap(i, G[v][0]);
    }
}

void DFS2(int v){
    static int pv = 0; In[v] = ++pv;
    for(auto i : G[v]) Top[i] = i == G[v][0] ? Top[v] : i, DFS2(i);
}

void VertexUpdate(int x, int v){
    Update(In[x], v);
}

long long PathQuery(int u, int v){
    long long res = 0;
    for(; Top[u] != Top[v]; u=Par[Top[u]]){
        if(Dep[Top[u]] < Dep[Top[v]]) swap(u, v);
        res += Query(In[Top[u]], In[u]);
    }
    if(In[u] > In[v]) swap(u, v);
    res += SegQuery(In[u], In[v]); // 정점 쿼리는 In[u], 간선 쿼리는 In[u]+1
    return res;
}
/////
// 1. Connect로 간선 추가
// 2. DFS0(1); DFS1(1); DFS2(Top[1]=1); 호출
// 3. VertexUpdate, PathQuery로 연산 수행
//  3-1. Update, Query는 배열의 구간 쿼리를 지원하는 자료구조(ex. 세그먼트 트리) 사용
```

5 수학

5.1 나눗셈, 최대공약수, 최소공배수

```
// floor(p / q)
int floor(int p, int q){
    if(q < 0) p = -p, q = -q;
    return p >= 0 ? p / q : (p - q + 1) / q;
}
```

```
// ceil(p / q)
int ceil(int p, int q){
    if(q < 0) p = -p, q = -q;
    return p >= 0 ? (p + q - 1) / q : p / q;
}

// a, b >= 0, O(log max(a,b))
int gcd(int a, int b){ return b ? gcd(b, a % b) : 0; }
int lcm(int a, int b){ return a / gcd(a, b) * b; }
```

5.2 빠른 거듭제곱

Usage: $a^b \pmod{c}$를 구하는 함수
Time Complexity: $O(\log b)$

```
ll PowMod(ll a, ll b, ll c){
    if(c == 1) return 0;
    ll res = 1;
    for(a%=c; b; b >>= 1, a = a * a % c) if(b & 1) res = res * a % c;
    return res;
}
```

5.3 소수 판별, 소인수분해

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

```
// IsPrime(2) = true, IsPrime(4) = false
// Factorize(72) = { {2, 3}, {3, 2} }
bool IsPrime(ll n){
    if(n < 2) return false;
    for(ll i=2; i*i<=n; i++) if(n % i == 0) return false;
    return true;
}
vector<pair<ll,ll>> Factorize(ll n){
    if(n == 1) return {};
    vector<pair<ll,ll>> res;
    for(ll i=2; i*i<=n; i++){
        if(n % i != 0) continue;
        int cnt = 0;
        while(n % i == 0) n /= i, cnt++;
        res.emplace_back(i, cnt);
    }
    if(n != 1) res.emplace_back(n, 1);
    return res;
}
```

5.4 에라토스테네스의 체, 소인수분해

Time Complexity: Sieve: $O(N \log\log N)$, Factorize: $O(\log N)$

```
int SP[5050505]; // SP[i] = i의 가장 작은 소인수
vector<int> Primes;

// n 이하의 모든 소수를 구함
void Sieve(int n){
  for(int i=2; i<=n; i++){
    if(SP[i]) continue;
    for(int j=i; j<=n; j+=i) if(!SP[j]) SP[j] = i;
  }
}

// Sieve 먼저 호출해야 함
vector<pair<int,int>> Factorize(int n){
  vector<pair<int,int>> res;
  while(n != 1){
    if(res.empty() || res.back().first != SP[n]) res.emplace_back(SP[n], 1);
    else res.back().second++;
    n /= SP[n];
  }
  return res;
}
```

5.5 선형 시간 체, 곱셈적 함수 전처리

```
// sigma 계산하는 부분 제외하면 O(n), sigma 계산은 O(n log log n)
// pw(j, e[i*j]) 모두 전처리하면 O(n)에 계산할 수 있음
// prime: n 이하 소수 목록
// sp : 최소 소인수, 소수라면 0
// tau : 약수 개수, sigma : 약수 합
// phi : n 이하 자연수 중 n과 서로소인 개수
// mu : 2번 이상 곱해진 소인수가 있으면 0, 그렇지 않으면 (-1)^(소인수 개수)
// e[i] : i에 sp[i]가 곱해진 횟수
vector<int> prime;
int sp[sz], e[sz], phi[sz], mu[sz], tau[sz], sigma[sz];
phi[1] = mu[1] = tau[1] = sigma[1] = 1;
for(int i=2; i<=n; i++){
  if(!sp[i]){
    prime.push_back(i);
    e[i] = 1; phi[i] = i-1; mu[i] = -1; tau[i] = 2; sigma[i] = i+1;
  }
  for(auto j : prime){
    if(i*j >= sz) break;
    sp[i*j] = j;
    if(i % j == 0){
      e[i*j] = e[i]+1; phi[i*j] = phi[i]*j; mu[i*j] = 0;
      tau[i*j] = tau[i]/e[i*j]*(e[i*j]+1);
      sigma[i*j] = sigma[i]*(j-1)/(pw(j, e[i*j])-1)*(pw(j, e[i*j]+1)-1)/(j-1);//overflow
      break;
    }
    e[i*j] = 1; phi[i*j] = phi[i] * phi[j]; mu[i*j] = mu[i] * mu[j];
    tau[i*j] = tau[i] * tau[j]; sigma[i*j] = sigma[i] * sigma[j];
  }
}
```

5.6 확장 유클리드 알고리즘

Time Complexity: $O(\log \max(a, b))$

```
// 정수 a, b 주어지면 ax + by = gcd(a, b) = g
// 를 만족하는 정수 {g, x, y} 반환
tuple<ll,ll,ll> ext_gcd(ll a, ll b){
  if(b == 0) return {a, 1, 0};
  auto [g,x,y] = ext_gcd(b, a % b);
  return {g, y, x - a/b * y};
}
```

5.7 중국인의 나머지 정리

Time Complexity: $O(k \log m)$

```
// a = a1 (mod m1), a = a2 (mod m2)를 만족하는 {a, lcm(m1, m2)} 반환
// 만약 a가 존재하면 0 <= a < lcm(m1, m2) 에서 유일하게 존재
// a가 존재하지 않는 경우 {-1, -1} 반환
ll mod(ll a, ll b){ return (a %= b) >= 0 ? a : a + b; }
pair<ll,ll> crt(ll a1, ll m1, ll a2, ll m2){
  ll g = gcd(m1, m2), m = m1 / g * m2;
  if((a2 - a1) % g) return {-1, -1};
  ll md = m2/g, s = mod((a2-a1)/g, m2/g);
  ll t = mod(get<1>(ext_gcd(m1/g%md, m2/g)), md);
  return { a1 + s * t % md * m1, m };
}
// a = a_i (mod m_i)를 만족하는 {a, lcm(m_1, ... , m_k)} 반환
// a가 존재하지 않는 경우 {-1, -1} 반환
pair<ll,ll> crt(const vector<ll> &a, const vector<ll> &m){
  ll ra = a[0], rm = m[0];
  for(int i=1; i<m.size(); i++){
    auto [aa,mm] = crt(ra, rm, a[i], m[i]);
    if(mm == -1) return {-1, -1}; else tie(ra,rm) = tie(aa,mm);
  }
  return {ra, rm};
}
```

5.8 이항 계수를 소수로 나눈 나머지

Time Complexity: 전처리 $O(P)$, 쿼리 $O(\log P)$

```
// Lucas C(13);
// C.calc(5, 3) = 5C3 = 10
// C.calc(10, 2) = 10C2 % 13 = 45 % 13 = 6
// 주의: P는 소수
// P가 크고(약 10억) n,r이 작으면(1000만 이하)
// 생성자에서 fac, inv를 1000만까지만 구해도 됨

struct Lucas{ // init : O(P), query : O(log P)
    const size_t P;
    vector<ll> fac, inv;
    ll Pow(ll a, ll b){
        ll res = 1;
        for(; b; b>>=1, a=a*a%P) if(b & 1) res = res * a % P;
        return res;
    }
    Lucas(size_t P) : P(P), fac(P), inv(P) {
        fac[0] = 1; for(int i=1; i<P; i++) fac[i] = fac[i-1] * i % P;
        inv[P-1] = Pow(fac[P-1], P-2); for(int i=P-2; ~i; i--) inv[i] = inv[i+1] * (i+1) %
        P;
    }
    ll small(ll n, ll r) const { return r <= n ? fac[n] * inv[r] % P * inv[n-r] % P : 0LL;
    }
    ll calc(ll n, ll r) const {
        if(n < r || n < 0 || r < 0) return 0;
        if(!n || !r || n == r) return 1; else return small(n%P, r%P) * calc(n/P, r/P) % P;
    }
};
```

5.9 빠른 소수 판별, 소인수분해 - Miller Rabin, Pollard Rho

Usage: 처음에 Sieve() 호출해야 함
Time Complexity: IsPrime: $O(\log^2 N)$, Factorize: 약 $O(N^{1/4})$

```
constexpr int SZ = 10'000'000;
bool PrimeCheck[SZ+1]; vector<int> Primes;
void Sieve(){
    memset(PrimeCheck, true, sizeof PrimeCheck);
    PrimeCheck[0] = PrimeCheck[1] = false;
    for(int i=2; i<=SZ; i++){
        if(PrimeCheck[i]) Primes.push_back(i);
        for(auto j : Primes){
            if(i*j > SZ) break;
            PrimeCheck[i*j] = false;
            if(i % j == 0) break;
        }
    }
}

using ll = long long;
using ull = unsigned long long;
ull MulMod(ull a, ull b, ull c){ return (__uint128_t)a * b % c; }
ull PowMod(ull a, ull b, ull c){
    ull res = 1; a %= c;
    for(; b; b>>=1, a=MulMod(a,a,c)) if(b & 1) res = MulMod(res,a,c);
    return res;
}
bool MillerRabin(ull n, ull a){
    if(a % n == 0) return true;
    int cnt = __builtin_ctzll(n - 1);
    ull p = PowMod(a, n >> cnt, n);
    if(p == 1 || p == n - 1) return true;
    while(cnt--) if((p=MulMod(p,p,n)) == n - 1) return true;
    return false;
}
bool IsPrime(ll n){
    if(n <= SZ) return PrimeCheck[n];
    if(n <= 2) return n == 2;
    if(n % 2 == 0 || n % 3 == 0 || n % 5 == 0 || n % 7 == 0 || n % 11 == 0) return false;
    // 32bit integer: {2, 7, 61}
```

```cpp
    for(int p : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) if(!MillerRabin(n, p))
    return false;
    return true;
}
ll Rho(ll n){
    while(true){
        ll x = rand() % (n - 2) + 2, y = x, c = rand() % (n - 1) + 1;
        while(true){
            x = (MulMod(x, x, n) + c) % n;
            y = (MulMod(y, y, n) + c) % n;
            y = (MulMod(y, y, n) + c) % n;
            ll d = __gcd(abs(x - y), n);
            if(d == 1) continue;
            if(IsPrime(d)) return d;
            else{ n = d; break; }
        }
    }
}
vector<pair<ll,ll>> Factorize(ll n){
    vector<pair<ll,ll>> v;
    int two = __builtin_ctzll(n);
    if(two > 0) v.emplace_back(2, two), n >>= two;
    if(n == 1) return v;
    while(!IsPrime(n)){
        ll d = Rho(n), cnt = 0;
        while(n % d == 0) cnt++, n /= d;
        v.emplace_back(d, cnt);
        if(n == 1) break;
    }
    if(n != 1) v.emplace_back(n, 1);
    return v;
}
```

5.10 가우스 소거법 - RREF, 랭크, 행렬식, 역행렬

Time Complexity: $O(N^3)$

```cpp
// T Add(T a, T b), Sub, Mul, Div 구현해야 함
// bool IsZero(T x) 구현해야 함
template<typename T> // return {rref, rank, det, inv}
tuple<vector<vector<T>>, int, T, vector<vector<T>>> Gauss(vector<vector<T>> a, bool
square=true){
  int n = a.size(), m = a[0].size(), rank = 0;
  vector<vector<T>> out(n, vector<T>(m, 0)); T det = T(1);
  for(int i=0; i<n; i++) if(square) out[i][i] = T(1);
  for(int i=0; i<m; i++){
    if(rank == n) break;
    if(IsZero(a[rank][i])){
      T mx = T(0); int idx = -1; // fucking precision error
      for(int j=rank+1; j<n; j++) if(mx < abs(a[j][i])) mx = abs(a[j][i]), idx = j;
      if(idx == -1 || IsZero(a[idx][i])){ det = 0; continue; }
      for(int k=0; k<m; k++){
        a[rank][k] = Add(a[rank][k], a[idx][k]);
        if(square) out[rank][k] = Add(out[rank][k], out[idx][k]);
      }
    }
    det = Mul(det, a[rank][i]);
    T coeff = Div(T(1), a[rank][i]);
    for(int j=0; j<m; j++) a[rank][j] = Mul(a[rank][j], coeff);
    for(int j=0; j<m; j++) if(square) out[rank][j] = Mul(out[rank][j], coeff);
    for(int j=0; j<n; j++){
      if(rank == j) continue;
      T t = a[j][i]; // Warning: [j][k], [rank][k]
      for(int k=0; k<m; k++) a[j][k] = Sub(a[j][k], Mul(a[rank][k], t));
      for(int k=0; k<m; k++) if(square) out[j][k] = Sub(out[j][k], Mul(out[rank][k], t));
    }
    rank++;
  }
  return {a, rank, det, out};
}
```

5.11 다항식 곱셈(FFT)

Time Complexity: $O(N \log N)$

```cpp
// 104,857,601   =  25 * 2^22 + 1, w = 3 | 998,244,353   = 119 * 2^23 + 1, w = 3
// 2,281,701,377 =  17 * 2^27 + 1, w = 3 | 2,483,027,969 =  37 * 2^26 + 1, w = 3
// 2,113,929,217 =  63 * 2^25 + 1, w = 5 | 1,092,616,193 = 521 * 2^21 + 1, w = 3
using real_t = double; using cpx = complex<real_t>;
void FFT(vector<cpx> &a, bool inv_fft=false){
  int N = a.size(); vector<cpx> root(N/2);
  for(int i=1, j=0; i<N; i++){
    int bit = N / 2;
    while(j >= bit) j -= bit, bit >>= 1;
    if(i < (j += bit)) swap(a[i], a[j]);
  }
  long double ang = 2 * acosl(-1) / N * (inv_fft ? -1 : 1);
  for(int i=0; i<N/2; i++) root[i] = cpx(cosl(ang*i), sinl(ang*i));
  /*
  NTT : ang = pow(w, (mod-1)/n) % mod, inv_fft -> ang^{-1}, root[i] = root[i-1] * ang
  XOR Convolution : set roots[*] = 1, a[j+k] = u+v, a[j+k+i/2] = u-v
   OR Convolution : set roots[*] = 1, a[j+k+i/2] += inv_fft ? -u : u;
  AND Convolution : set roots[*] = 1, a[j+k  ] += inv_fft ? -v : v;
  */
  for(int i=2; i<=N; i<<=1){
    int step = N / i;
    for(int j=0; j<N; j+=i) for(int k=0; k<i/2; k++){
        cpx u = a[j+k], v = a[j+k+i/2] * root[step * k];
        a[j+k] = u+v; a[j+k+i/2] = u-v;
      }
  }
  if(inv_fft) for(int i=0; i<N; i++) a[i] /= N; // skip for AND/OR convolution.
}
vector<ll> multiply(const vector<ll> &_a, const vector<ll> &_b){
  vector<cpx> a(all(_a)), b(all(_b));
  int N = 2; while(N < a.size() + b.size()) N <<= 1;
  a.resize(N); b.resize(N); FFT(a); FFT(b);
  for(int i=0; i<N; i++) a[i] *= b[i];
```

```
  vector<ll> ret(N); FFT(a, 1); // NTT : just return a
  for(int i=0; i<N; i++) ret[i] = llround(a[i].real());
  while(ret.size() > 1 && ret.back() == 0) ret.pop_back();
  return ret;
}
// 더 높은 정밀도
vector<ll> multiply_mod(const vector<ll> &a, const vector<ll> &b, const ull mod){
  int N = 2; while(N < a.size() + b.size()) N <<= 1;
  vector<cpx> v1(N), v2(N), r1(N), r2(N);
  for(int i=0; i<a.size(); i++) v1[i] = cpx(a[i] >> 15, a[i] & 32767);
  for(int i=0; i<b.size(); i++) v2[i] = cpx(b[i] >> 15, b[i] & 32767);
  FFT(v1); FFT(v2);
  for(int i=0; i<N; i++){
    int j = i ? N-i : i;
    cpx ans1 = (v1[i] + conj(v1[j])) * cpx(0.5, 0);
    cpx ans2 = (v1[i] - conj(v1[j])) * cpx(0, -0.5);
    cpx ans3 = (v2[i] + conj(v2[j])) * cpx(0.5, 0);
    cpx ans4 = (v2[i] - conj(v2[j])) * cpx(0, -0.5);
    r1[i] = (ans1 * ans3) + (ans1 * ans4) * cpx(0, 1);
    r2[i] = (ans2 * ans3) + (ans2 * ans4) * cpx(0, 1);
  }
  vector<ll> ret(N); FFT(r1, true); FFT(r2, true);
  for(int i=0; i<N; i++){
    ll av = llround(r1[i].real()) % mod;
    ll bv = ( llround(r1[i].imag()) + llround(r2[i].real()) ) % mod;
    ll cv = llround(r2[i].imag()) % mod;
    ret[i] = (av << 30) + (bv << 15) + cv;
    ret[i] %= mod; ret[i] += mod; ret[i] %= mod;
  }
  while(ret.size() > 1 && ret.back() == 0) ret.pop_back();
  return ret;
}
```

6 문자열

6.1 문자열 해싱

Time Complexity: build: $O(N)$, get: $O(1)$

```
// 전처리 O(N), 부분 문자열의 해시값을 O(1)에 구함
// Hashing<917, 998244353> H;
// H.build("ABCDABCD");
// assert(H.get(1, 4) == H.get(5, 8));
// 주의: get 함수의 인자는 1-based 닫힌 구간
// 주의: M은 10억 근처의 소수, P는 M과 서로소

// 1e5+3, 1e5+13, 131'071, 524'287, 1'299'709, 1'301'021
// 1e9-63, 1e9+7, 1e9+9, 1e9+103
template<long long P, long long M> struct Hashing {
    vector<long long> h, p;
    void build(const string &s){
        int n = s.size();
        h = p = vector<long long>(n+1); p[0] = 1;
        for(int i=1; i<=n; i++) h[i] = (h[i-1] * P + s[i-1]) % M;
        for(int i=1; i<=n; i++) p[i] = p[i-1] * P % M;
    }
    long long get(int s, int e) const {
        long long res = (h[e] - h[s-1] * p[e-s+1]) % M;
        return res >= 0 ? res : res + M;
    }
};
```

6.2 문자열 매칭 - KMP

Time Complexity: GetFail: $O(|P|)$, $O(|S| + |P|)$

```
// s에서 p가 등장하는 위치 반환
// KMP("ABABCAB", "AB") = {0, 2, 5}
// KMP("AAAA", "AA") = {0, 1, 2}

vector<int> GetFail(const string &p){
    int n = p.size();
    vector<int> fail(n);
    for(int i=1, j=0; i<n; i++){
        while(j && p[i] != p[j]) j = fail[j-1];
        if(p[i] == p[j]) fail[i] = ++j;
    }
    return fail;
}

vector<int> KMP(const string &s, const string &p){
    int n = s.size(), m = p.size();
    vector<int> fail = GetFail(p), ret;
    for(int i=0, j=0; i<s.size(); i++){
        while(j && s[i] != p[j]) j = fail[j-1];
        if(s[i] == p[j]){
            if(j + 1 == m) ret.push_back(i-m+1), j = fail[j];
            else j++;
        }
    }
    return ret;
}
```

6.3 가장 긴 팰린드롬 부분 문자열 - Manacher

Time Complexity: $O(N)$

```
// 각 문자를 중심으로 하는 최장 팰린드롬의 반경을 반환
// Manacher("abaaba") = {0,1,0,3,0,1,6,1,0,3,0,1,0}
// # a # b # a # a # b # a #
// 0 1 0 3 0 1 6 1 0 3 0 1 0
vector<int> Manacher(const string &inp){
    int n = inp.size() * 2 + 1;
    vector<int> ret(n);
    string s = "#";
    for(auto i : inp) s += i, s += "#";
    for(int i=0, p=-1, r=-1; i<n; i++){
```

```
        ret[i] = i <= r ? min(r-i, ret[2*p-i]) : 0;
        while(i-ret[i]-1 >= 0 && i+ret[i]+1 < n && s[i-ret[i]-1] == s[i+ret[i]+1])
        ret[i]++;
        if(i+ret[i] > r) r = i+ret[i], p = i;
    }
    return ret;
}
```

6.4 문자열 매칭 - Z

Time Complexity: $O(N)$

```
// Z[i] = LongestCommonPrefix(S[0:N], S[i:N])
//      = S[0:N]과 S[i:N]이 앞에서부터 몇 글자 겹치는지
vector<int> Z(const string &s){
    int n = s.size();
    vector<int> z(n);
    z[0] = n;
    for(int i=1, l=0, r=0; i<n; i++){
        if(i < r) z[i] = min(r-i-1, z[i-l]);
        while(i+z[i] < n && s[i+z[i]] == s[z[i]]) z[i]++;
        if(i+z[i] > r) r = i+z[i], l = i;
    }
    return z;
}
```

6.5 접미사 배열

Time Complexity: $O(N \log N)$

```
// LCP는 1-based
pair<vector<int>, vector<int>> SuffixArray(const string &s){ // O(N log N)
  int n = s.size(), m = max(n, 256);
  vector<int> sa(n), lcp(n), pos(n), tmp(n), cnt(m);
  auto counting_sort = [&](){
    fill(cnt.begin(), cnt.end(), 0);
    for(int i=0; i<n; i++) cnt[pos[i]]++;
    partial_sum(cnt.begin(), cnt.end(), cnt.begin());
    for(int i=n-1; i>=0; i--) sa[--cnt[pos[tmp[i]]]] = tmp[i];
  };
  for(int i=0; i<n; i++) sa[i] = i, pos[i] = s[i], tmp[i] = i;
  counting_sort();
  for(int k=1; ; k<<=1){
    int p = 0;
    for(int i=n-k; i<n; i++) tmp[p++] = i;
    for(int i=0; i<n; i++) if(sa[i] >= k) tmp[p++] = sa[i] - k;
    counting_sort();
    tmp[sa[0]] = 0;
    for(int i=1; i<n; i++){
      tmp[sa[i]] = tmp[sa[i-1]];
      if(sa[i-1]+k < n && sa[i]+k < n && pos[sa[i-1]] == pos[sa[i]] && pos[sa[i-1]+k] ==
      pos[sa[i]+k]) continue;
      tmp[sa[i]] += 1;
    }
    swap(pos, tmp); if(pos[sa.back()] + 1 == n) break;
  }
  for(int i=0, j=0; i<n; i++, j=max(j-1,0)){
    if(pos[i] == 0) continue;
    while(sa[pos[i]-1]+j < n && sa[pos[i]]+j < n && s[sa[pos[i]-1]+j] == s[sa[pos[i]]+j])
    j++;
    lcp[pos[i]] = j;
  }
  return {sa, lcp};
}
```

7 계산 기하

7.1 2차원 계산 기하 템플릿 + CCW

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using ll = long long;
using Point = pair<ll, ll>;

Point operator + (Point p1, Point p2){ return {p1.x + p2.x, p1.y + p2.y}; }
Point operator - (Point p1, Point p2){ return {p1.x - p2.x, p1.y - p2.y}; }
ll operator * (Point p1, Point p2){ return p1.x * p2.x + p1.y * p2.y; } // 내적
ll operator / (Point p1, Point p2){ return p1.x * p2.y - p2.x * p1.y; } // 외적
int Sign(ll v){ return (v > 0) - (v < 0); } // 양수면 +1, 음수면 -1, 0이면 0 반환
ll Dist(Point p1, Point p2){ return (p2 - p1) * (p2 - p1); } // 두 점 거리 제곱
ll SignedArea(Point p1, Point p2, Point p3){ return (p2 - p1) / (p3 - p1); }
int CCW(Point p1, Point p2, Point p3){ return Sign(SignedArea(p1, p2, p3)); }
```

7.2 360도 각도 정렬

```
/* y축
    ↑
3  2 1
4 -1 0 → x축
5  6 7
원점 기준 각도 정렬
x축 위에 있는 점이 가장 먼저, 그리고 반시계 방향으로
*/
int QuadrantID(const Point p){
  static int arr[9] = { 5, 4, 3, 6, -1, 2, 7, 0, 1 };
  return arr[Sign(p.x)*3+Sign(p.y)+4];
}
sort(points.begin(), points.end(), [&](auto p1, auto p2){
  if(QuadrantID(p1) != QuadrantID(p2)) return QuadrantID(p1) < QuadrantID(p2);
  else return p1 / p2 > 0; // 반시계
});
```

7.3 다각형 넓이

```
// 다각형의 넓이의 2배를 반환, 항상 정수, O(N)
ll PolygonArea(const vector<Point> &v){
```

```
    ll res = 0;
    for(int i=0; i<v.size(); i++) res += v[i] / v[(i+1)%v.size()];
    return abs(res);
}
```

7.4 선분 교차 판정

```
// 선분 교차 - 선분 ab와 선분 cd가 만나면 true
bool Cross(Point s1, Point e1, Point s2, Point e2){
    int ab = CCW(s1, e1, s2) * CCW(s1, e1, e2);
    int cd = CCW(s2, e2, s1) * CCW(s2, e2, e1);
    if(ab == 0 && cd == 0){
        if(s1 > e1) swap(s1, e1);
        if(s2 > e2) swap(s2, e2);
        return !(e1 < s2 || e2 < s1);
    }
    return ab <= 0 && cd <= 0;
}
// 교차하지 않으면 0
// 교점이 무한히 많으면 -1
// 교점이 1개면 1 반환하고 res에 교점 저장
int Cross(Point s1, Point e1, Point s2, Point e2, pair<double, double> &res){
    if(!Cross(s1, e1, s2, e2)) return 0;
    ll det = (e1 - s1) / (e2 - s2);
    if(!det){
        if(s1 > e1) swap(s1, e1);
        if(s2 > e2) swap(s2, e2);
        if(e1 == s2){ res = s2; return 1; }
        if(e2 == s1){ res = s1; return 1; }
        return -1;
    }
    res.x = s1.x + (e1.x - s1.x) * ((s2 - s1) / (e2 - s2) * 1.0 / det);
    res.y = s1.y + (e1.y - s1.y) * ((s2 - s1) / (e2 - s2) * 1.0 / det);
    return 1;
}
```

7.5 다각형 내부 판별

```
// 다각형 내부 또는 경계 위에 p가 있으면 true, O(N)
bool PointInPolygon(const vector<Point> &v, Point p){
    int n = v.size(), cnt = 0;
    Point p2(p.x+1, 1'000'000'000 + 1); // 좌표 범위보다 큰 수
    for(int i=0; i<n; i++){
        int j = i + 1 < n ? i + 1 : 0;
        if(min(v[i],v[j]) <= p && p <= max(v[i],v[j]) && CCW(v[i], v[j], p) == 0) return
        true;
        if(SegmentIntersection(v[i], v[j], p, p2)) cnt++;
    }
    return cnt % 2 == 1;
}
```

7.6 볼록 껍질 - Graham Scan

```
// 모든 점을 포함하는 가장 작은 볼록 다각형, O(N log N)
vector<Point> ConvexHull(vector<Point> points){
    if(points.size() <= 1) return points;
    swap(points[0], *min_element(points.begin(), points.end()));
    sort(points.begin()+1, points.end(), [&](auto a, auto b){
        int dir = CCW(points[0], a, b);
        if(dir != 0) return dir > 0;
        return Dist(points[0], a) < Dist(points[0], b);
    });
    vector<Point> hull;
    for(auto p : points){
        while(hull.size() >= 2 && CCW(hull[hull.size()-2], hull.back(), p) <= 0)
        hull.pop_back();
        hull.push_back(p);
    }
    return hull;
}
```

7.7 가장 먼 두 점 - Rotating Calipers

```
// 가장 먼 두 점을 구하는 함수, O(N)
// 주의: hull은 반시계 방향으로 정렬된 볼록 다각형이어야 함
pair<Point, Point> Calipers(vector<Point> hull){
    int n = hull.size(); ll mx = 0; Point a, b;
    for(int i=0, j=0; i<n; i++){
        while(j + 1 < n && (hull[i+1] - hull[i]) / (hull[j+1] - hull[j]) >= 0){
            ll now = Dist(hull[i], hull[j]);
            if(now > mx) mx = now, a = hull[i], b = hull[j];
            j++;
        }
        ll now = Dist(hull[i], hull[j]);
        if(now > mx) mx = now, a = hull[i], b = hull[j];
    }
    return {a, b};
}
```

7.8 볼록 다각형 내부 판별

```
// 다각형 내부 또는 경계 위에 p가 있으면 true, O(log N)
// 주의: v는 반시계 방향으로 정렬된 볼록 다각형이어야 함
bool PointInConvexPolygon(const vector<Point> &v, const Point &pt){
    if(CCW(v[0], v[1], pt) < 0) return false; int l = 1, r = v.size() - 1;
    while(l < r){
        int m = l + r + 1 >> 1;
        if(CCW(v[0], v[m], pt) >= 0) l = m; else r = m - 1;
    }
    if(l == v.size() - 1) return CCW(v[0], v.back(), pt) == 0 && v[0] <= pt && pt <=
    v.back();
    return CCW(v[0], v[l], pt) >= 0 && CCW(v[l], v[l+1], pt) >= 0 && CCW(v[l+1], v[0], pt)
    >= 0;
}
```

8 기타

8.1 가장 긴 증가하는 부분 수열(LIS)

Time Complexity: $O(N \log N)$

```
// LIS({10, 20, 10, 30, 20, 50}) = {10, 20, 30, 50}
vector<int> LIS(const vector<int> &v){
  int n = v.size();
  vector<int> pos(n), lis, res;
  for(int i=0; i<n; i++){
    if(lis.empty() || lis.back() < v[i]){
      lis.push_back(v[i]); pos[i] = lis.size();
    }
    else{
      int idx = lower_bound(lis.begin(), lis.end(), v[i]) - lis.begin();
      lis[idx] = v[i]; pos[i] = idx + 1;
    }
  }
  int len = lis.size();
  for(int i=n-1; i>=0; i--) if(pos[i] == len) res.push_back(v[i]), len--;
  reverse(res.begin(), res.end());
  return res;
}
```

8.2 이분 탐색

```
// 주의: 구간 범위에 음수가 들어가면 / 2 말고 >> 1로 해야 함

// 1 1 1 ... 1 1 0 0 ... 0 꼴일 때 마지막 1의 위치
int l = 1, r = n;
while(l < r){
    int m = (l + r + 1) / 2;
    if(Check(m)) l = m;
    else r = m - 1;
}

// 0 0 0 ... 0 0 1 1 ... 1 꼴일 때 첫 번째 1의 위치
int l = 1, r = n;
while(l < r){
    int m = (l + r) / 2;
    if(Check(m)) r = m;
    else l = m + 1;
}
```

8.3 삼분 탐색

```
while(s + 3 <= e){ // get minimum / when multiple answer, find minimum `s`
  T l = (s + s + e) / 3, r = (s + e + e) / 3;
  if(Check(l) > Check(r)) s = l; else e = r;
}
T mn = INF, idx = s;
for(T i=s; i<=e; i++) if(T now = Check(i); now < mn) mn = now, idx = i;
```

8.4 C++ 랜덤, GCC 확장, 비트마스킹 트릭

```
mt19937 rd((unsigned)chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<int> rnd_int(l, r); // rnd_int(rd)
uniform_real_distribution<double> rnd_real(0, 1); // rnd_real(rd)
/////
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/rope>
using namespace __gnu_pbds; //ordered_set : find_by_order(order), order_of_key(key)
using namespace __gnu_cxx; //crope : append(str), substr(s, e), at(idx)
template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
/////
int __builtin_clz(int x);// number of leading zero
int __builtin_ctz(int x);// number of trailing zero
int __builtin_popcount(int x);// number of 1-bits in x
lsb(n): (n & -n); // last bit (smallest)
floor(log2(n)): 31 - __builtin_clz(n | 1);
floor(log2(n)): 63 - __builtin_clzll(n | 1);
long long next_perm(long long v){
  long long t = v | (v-1);
  return (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(v) + 1));
}
int frq(int n, int i) { // # of digit i in [1, n]
  int j, r = 0;
  for (j = 1; j <= n; j *= 10) if (n / j / 10 >= !i) r += (n / 10 / j - !i) * j + (n / j %
  10 > i ? j : n / j % 10 == i ? n % j + 1 : 0);
  return r;
}
```

8.5 빠른 입력(Fast Input from STDIN)

```
namespace io {
  const signed IS=1<<20;
  char I[IS+1],*J=I;
  inline void daer(){if(J>=I+IS-64){
    char*p=I;do*p++=*J++;
    while(J!=I+IS);p[read(0,p,I+IS-p)]=0;J=I;}}
  template<int N=10,typename T=int>inline T getu(){
    daer();T x=0;int k=0;do x=x*10+*J-'0';
    while(*++J>='0'&&++k<N);++J;return x;}
  template<int N=10,typename T=int>inline T geti(){
    daer();bool e=*J=='-';J+=e;return(e?-1:1)*getu<N,T>();}
  struct f{f(){I[read(0,I,IS)]=0;}}flu;
};
```

8.6 구간별 약수 최대 개수, 최대 소수

```
< 10^k          number     divisors   2 3 5 71113171923293137
-------------------------------------------------------------
1                    6            4   1 1
2                   60           12   2 1 1
```

```
3                    840             32   3 1 1 1
4                   7560             64   3 3 1 1
5                  83160            128   3 3 1 1 1
6                 720720            240   4 2 1 1 1 1
7                8648640            448   6 3 1 1 1 1
8               73513440            768   5 3 1 1 1 1 1
9              735134400           1344   6 3 2 1 1 1 1
10            6983776800           2304   5 3 2 1 1 1 1 1
11           97772875200           4032   6 3 2 2 1 1 1 1
12          963761198400           6720   6 4 2 1 1 1 1 1 1
13         9316358251200          10752   6 3 2 1 1 1 1 1 1 1
14        97821761637600          17280   5 4 2 2 1 1 1 1 1 1
15       866421317361600          26880   6 4 2 1 1 1 1 1 1 1 1
16      8086598962041600          41472   8 3 2 2 1 1 1 1 1 1 1
17     74801040398884800          64512   6 3 2 2 1 1 1 1 1 1 1 1
18    897612484786617600         103680   8 4 2 2 1 1 1 1 1 1 1 1
```

```
< 10^k     prime    # of prime           < 10^k               prime
-------------------------------------------------------------------
1              7             4           10              9999999967
2             97            25           11             99999999977
3            997           168           12            999999999989
4           9973          1229           13           9999999999971
5          99991          9592           14          99999999999973
6         999983         78498           15         999999999999989
7        9999991        664579           16        9999999999999937
8       99999989       5761455           17       99999999999999997
9      999999937      50847534           18      999999999999999989
```

8.7 카탈란 수, 심슨 적분, 그런디 정리, 픽의 정리, 페르마 포인트, 오일러 정리

- 카탈란 수
  1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012,742900
  $C_n = binomial(n*2, n)/(n+1)$;
  - 길이가 2n인 올바른 괄호 수식의 수
  - n + 1개의 리프를 가진 풀 바이너리 트리의 수
  - n + 2각형을 n개의 삼각형으로 나누는 방법의 수
- Simpson 공식 (적분) : Simpson 공식, $S_n(f) = \frac{h}{3}[f(x_0) + f(x_n) + 4\sum f(x_{2i+1}) + 2\sum f(x_{2i})]$
  - $M = \max|f^4(x)|$이라고 하면 오차 범위는 최대 $E_n \le \frac{M(b-a)}{180}h^4$
- 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 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 + 1로 나눈 나머지를 XOR 합하여 판단한다.
  - Index-k Nim : 한 번에 최대 k개의 더미를 골라 각각의 더미에서 아무렇게나 돌을 제거할 수 있을 때, 각 binary digit에 대하여 합을 k + 1로 나눈 나머지를 계산한다. 만약 이 나머지가 모든 digit에 대하여 0이라면 두번째, 하나라도 0이 아니라면 첫번째 플레이어가 승리.
  - Misere Nim : 모든 돌 무더기가 1이면 N이 홀수일 때 후공 승, 그렇지 않은 경우 XOR 합 0이면 후공 승
- Pick' s Theorem
  격자점으로 구성된 simple polygon이 주어짐. I 는 polygon 내부의 격자점 수, B 는 polygon 선분 위 격자점 수, A는 polygon의 넓이라고 할 때, 다음과 같은 식이 성립한다. $A = I + B/2 - 1$

```
// number of (x, y) : (0 <= x < n && 0 < y <= k/d x + b/d)
ll count_solve(ll n, ll k, ll b, ll d) { // argument should be positive
  if (k == 0) {
    return (b / d) * n;
  }
  if (k >= d || b >= d) {
    return ((k / d) * (n - 1) + 2 * (b / d)) * n / 2 + count_solve(n, k % d, b % d, d);
  }
  return count_solve((k * n + b) / d, d, (k * n + b) % d, k);
}
```

- 페르마 포인트 : 삼각형의 세 꼭짓점으로부터 거리의 합이 최소가 되는 점
  $2\pi/3$ 보다 큰 각이 있으면 그 점이 페르마 포인트, 그렇지 않으면 각 변마다 정삼각형 그린 다음, 정삼각형의 끝점에서 반대쪽 삼각형의 꼭짓점으로 연결한 선분의 교점
  $2\pi/3$ 보다 큰 각이 없으면 거리의 합은 $\sqrt{(a^2 + b^2 + c^2 + 4\sqrt{3}S)/2}$, S는 넓이
- 오일러 정리: 서로소인 두 정수 a, n에 대해 $a^{\phi(n)} \equiv 1 \pmod n$
  모든 정수에 대해 $a^n \equiv a^{n-\phi(n)} \pmod n$
  $m \ge log_2 n$이면 $a^m \equiv a^{m\%\phi(n)+\phi(n)} \pmod n$
- $g^0 + g^1 + g^2 + \cdots g^{p-2} \equiv -1 \pmod p$ iff $g = 1$, otherwise 0.

8.8 경우의 수 - 포함 배제, 스털링 수, 벨 수

- 공 구별 X, 상자 구별 O, 전사함수 : 포함배제 $\sum_{i=1}^{k}(-1)^{k-i} \times kCi \times i^n$
- 공 구별 O, 상자 구별 X, 전사함수 : 제 2종 스털링 수 $S(n,k) = k \times S(n-1,k) + S(n-1,k-1)$
  포함배제하면 $O(K \log N)$, $S(n,k) = 1/k! \times \sum_{i=1}^{k}(-1)^{k-i} \times kCi \times i^n$
- 공 구별 O, 상자 구별 X, 제약없음 : 벨 수 $B(n,k) = \sum_{i=0}^{k} S(n,i)$ 몇 개의 상자를 버릴지 다 돌아보기
  수식 정리하면 $O(\min(N,K)\log N)$에 됨. $B(n,n) = \sum_{i=0}^{n-1}(n-1)Ci \times B(i,i)$
  $B(n,k) = \sum_{j=0}^{k} S(n,j) = \sum_{j=0}^{k} 1/j! \sum_{i=0}^{j}(-1)^{j-i} jCi \times i^n = \sum_{j=0}^{k}\sum_{i=0}^{j}\frac{(-1)^{j-i}}{i!(j-i)!}i^n$
  $= \sum_{i=0}^{k}\sum_{j=i}^{k}\frac{(-1)^{j-i}}{i!(j-i)!}i^n = \sum_{i=0}^{k}\sum_{j=0}^{k-i}\frac{(-1)^{j}}{i!j!}i^n = \sum_{i=0}^{k}\frac{i^n}{i!}\sum_{j=0}^{k-i}\frac{(-1)^j}{j!}$
- Derangement: $D(n) = (n-1)(D(n-1) + D(n-2))$
- Signed Stirling 1: $S_1(n,k) = (n-1)S_1(n-1,k) + S_1(n-1,k-1)$
- Unsigned Stirling 1: $C_1(n,k) = (n-1)C_1(n-1,k) + C_1(n-1,k-1)$
- Stirling 2: $S_2(n,k) = kS_2(n-1,k) + S_2(n-1,k-1)$
- Stirling 2: $S_2(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$
- Partition: $p(n,k) = p(n-1,k-1) + p(n-k,k)$
- Partition: $p(n) = \sum(-1)^k p(n - k(3k-1)/2)$
- Bell: $B(n) = \sum_{k=1}^{n}\binom{n-1}{k-1}B(n-k)$
- Catalan: $C_n = \frac{1}{n+1}\binom{2n}{n}$
- Catalan: $C_n = \binom{2n}{n} - \binom{2n}{n+1}$
- Catalan: $C_n = \frac{(2n)!}{n!(n+1)!}$
- Catalan: $C_n = \sum C_i C_{n-i}$

8.9 삼각형의 오심 - 외심, 내심, 무게중심, 수심, 방심

변 길이 $a, b, c; p = (a+b+c)/2$
넓이 $A = \sqrt{p(p-a)(p-b)(p-c)}$
외접원 반지름 $R = abc/4A$, 내접원 반지름 $r = A/p$
중선 길이 $m_a = 0.5\sqrt{2b^2+2c^2-a^2}$
각 이등분선 길이 $s_a = \sqrt{bc(1-\frac{a}{b+c}^2)}$
사인 법칙 $\frac{sinA}{a} = 1/2R$, 코사인 법칙 $a^2 = b^2+c^2-2bc\cos A$, 탄젠트 법칙 $\frac{a+b}{a-b} = \frac{\tan(A+B)/2}{\tan(A-B)/2}$
중심 좌표 $(\frac{\alpha x_a+\beta x_b+\gamma x_c}{\alpha+\beta+\gamma}, \frac{\alpha y_a+\beta y_b+\gamma y_c}{\alpha+\beta+\gamma})$

이름	α	β	γ	
외심	$a^2\mathcal{A}$	$b^2\mathcal{B}$	$c^2\mathcal{C}$	$\mathcal{A} = b^2+c^2-a^2$
내심	a	b	c	$\mathcal{B} = a^2+c^2-b^2$
무게중심	1	1	1	$\mathcal{C} = a^2+b^2-c^2$
수심	$\mathcal{BC}$	$\mathcal{CA}$	$\mathcal{AB}$	
방심(A)	$-a$	b	c	

8.10 미적분, 뉴턴 랩슨법

- $(\arcsin x)' = 1/\sqrt{1-x^2}$
- $(\tan x)' = 1+\tan^2 x$
- $\int tanax = -\ln|\cos ax|/a$
- $(\arccos x)' = -1/\sqrt{1-x^2}$
- $(\arctan x)' = 1/(1+x^2)$
- $\int x\sin ax = (\sin ax - ax\cos ax)/a^2$
- Newton: $x_{n+1} = x_n - f(x_n)/f'(x_n)$
- $\oint_C (Ldx+Mdy) = \int\int_D (\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y})dxdy$
- where C is positively oriented, piecewise smooth, simple, closed; D is the region inside C; L and M have continuous partial derivatives in D.

8.11 문제 풀이 체크리스트

- 비슷한 문제를 풀어본 적이 있던가?
- 단순한 방법에서 시작할 수 있을까? (Brute Force)
- 내가 문제를 푸는 과정을 수식화할 수 있을까? (예제를 직접 해결해보면서)
- 문제를 단순화할 수 없을까?
- 그림으로 그려볼 수 있을까?
- 수식으로 표현할 수 있을까?
- 문제를 분해할 수 있을까?
- 뒤에서부터 생각해서 풀 수 있을까?
- 순서를 강제할 수 있을까?
- 특정 형태의 답만을 고려할 수 있을까? (정규화)
- 구간을 통째로 가져간다 : 플로우 + 적당한 자료구조 $(i, i+1, k, 0), (s, e, 1, w), (N, T, k, 0)$
- a = b : a만 움직이기, b만 움직이기, 두 개 동시에 움직이기, 반대로 움직이기
- 말도 안 되는 것들을 한 번은 생각해보기 / "당연하다고 생각한 것" 다시 생각해보기
- 확률 : DP, 이분 탐색(NYPC 2019 Finals C)
- 최대/최소 : 이분 탐색, 그리디(Prefix 고정, Exchange Argument), DP(순서 고정)