Sogang University
Team: BlackWeasel

Coach : Prof. Jung Min So
Contestant : Jihwan Yim, Yunje Lee, Gihyeon Lee

Table of Contents

Table of Contents 1
Algorithm Checklist 2
Data Structures 2
- Biconnected Components & Cactus 2
- Fenwick tree with range update & point query 2
- SCC & 2-SAT 3
- Segment tree & lazy propagation (by recursion) 4
- Splay Tree 4

Graph Algorithms 7
- Shortest path(Bellman-ford's, Dijkstra's, 0-1 BFS) 7
- Centroid decomposition & tree isomorphism 7
- Euler path / circuit 8
- Diamater of tree 8

Mathematical Stuff 9
- Euler phi function 9
- Extended euclidean 9
- Fast fourier transform 9
- Fast $_nC_k$ 9
- Matrix 10
- Miller-rabin primality test & Pollard-rho factorization 10
- Mobius inversion 11

Geometry 12
- General Geometry Library Header 12
- Area series(Shoelace, Brahmagupta's, Heron's) 13
- Convex hull (Graham scan, monotone chain) 13
- Rotating Calipers 13

Flows 14
- Dinic's Algorithm 14
- Hopcroft-karp maximum matching 15
- Hungarian method 15
- MCMF 16

String 17
- Manacher & Z 17
- KMP 17
- Regular expression usage 18
- Trie & aho-corasick 18
- Suffix array & LCP 18

Queries 19
- Heavy-Light decomposition 19
- Offline dynamic connectivity 20
- Persistent segment tree 21
- Square-root decomposition & MO's 21

Dynamic programming optimization 21
- Convex hull trick 21
- DP optimization with deque 22
- Knuth optimization 22

Problem specified techniques 22
- histogram 22
- Dynamic programming 22
- Greedy 23
- sweeping with segment tree 23
- Graph 24

Algorithm Checklist

뒤집어서 생각하기
Greedy
DP – Knuth, Convex hull trick, Divide & Conquer Optimization
Binary Search (+with extra weight)
Divide & Conquer
Centroid Decomposition, Tree Diameter
Un/Unidirected Spanning tree
Hashing, KMP, Suffix Array, Manacher, Z-Algorithm
Min cut - Max flow, MCMF, Bipartite matching
Min cut – Resonable한 하한들로 고찰하기
a = b: a만 움직이기, b만 움직이기, 두 개 동시에 움직이기
답의 상한이 Resonable하게 작은가?
문제가 안풀릴 때 "당연하다고 생각한 것"을 의심하기.
말도 안되는 것을 한 번은 생각해보기.
Random Algorithm
LIS, LCS, Based on DP
Q Sqrt(N), ..
HLD, BCC, SCC
Double 안쓰기 (싫다아..)
Set, Map 쓰기 전에 생각하셨나요? 아슬아슬하면 unordered 얹어볼까요?
투포인터, sliding window

Data Structures

Biconnected components & Cactus

```
/*
Cactus 판별 source
cactus : 양방향 그래프에서 모든 노드로 돌아오는 사이클이 최대 1개인 그래프
dfs 트리에서 깊이를 관리하고 low 값을 이용하여 2개 이상의 자기자신의 dep 이 아닌 두 개 이상의 low 후보를 가질 수 있다면 not-cactus
*/
vector<vector<int>> adj;
vector<int> dep, low;
bool cactus = true;

int dfs(int cur, int prv) {
  dep[cur] = dep[prv] + 1;
  for (int nxt : adj[cur]) {
    if (nxt == prv) continue;
    // forward edge - pass 어차피 backward edge 에서 볼 예정
    else if (dep[nxt] && dep[nxt] > dep[cur]) continue;
    // backward edge
    else if (dep[nxt] && dep[nxt] < dep[cur]) {
      // 기존 low 값이 있다면 이미 포함된 사이클이 존재하는 것
      if (low[cur]) {
        cactus = false;
        low[cur] = -1;
      }
      // 처음으로 속하는 cycle 을 발견
      else low[cur] = nxt;
    }
    else {
      // child 의 low 값을 받아온다.
      int res = dfs(nxt, cur);
      // child 의 low 가 0보다 작으면 이미 cactus 되기는 글렀음
      if (res < 0) low[cur] = -1;
      // chilld 의 low 가 0이면 child 아래로는 사이클이 없다고 봐도 됨
      else if (res == 0);
      // child 의 low 가 존재 , cur 외에 더 위로 가는 사이클이 존재,
      // 그러므로 cur도 child 를 통해서 사이클이 포함됨
      else {
        // 마찬가지로 이미 cycle 포함으로 조짐
        if (low[cur]) {
          cactus = false;
          low[cur] = -1;
        }
        else low[cur] = res;
      }
    }
  }
  return low[cur] == cur ? 0 : low[cur];
}
```

Fenwick tree with range update & point query

```
struct fenwick {
  vector<int> tree;
  fenwick() {}
  fenwick(int N) { tree.resize(N+1); }
  void update(int idx, int val) {
    while (idx <= M) tree[idx] += val, idx += idx & -idx;
  }
  lint query(int idx) {
    lint ret = 0;
    while (idx) ret += tree[idx], idx -= idx & -idx;
    return ret;
  }
};

...
```

```
// range update(l, r, val)
fw.update(l, val), fw.update(r+1, -val);
// point query
fw.query(idx);
```

SCC & 2-SAT

```
#include <bits/stdc++.h>
using namespace std;
using ii = pair<int, int>;
int N, M;
vector<int>adj[20001];
vector<int> vis, sccId;
int sccNUM, visitcnt;
stack<int>S;

inline int oppo(int a) {
  return a % 2 ? a - 1 : a + 1;
}
int dfs(int cur) {
  S.push(cur);
  int ret = vis[cur] = visitcnt++;
  for (int &nxt : adj[cur]) {
    if (vis[nxt] == -1) {
      ret = min(ret, dfs(nxt));
    }
    else if (sccId[nxt] == -1) {
      ret = min(ret, vis[nxt]);
    }
  }
  if (vis[cur] == ret) {
    while (1) {
      int t = S.top();
      S.pop();
      sccId[t] = sccNUM;
      if (t == cur) break;
    }
    sccNUM++;
  }
  return ret;
}
vector<int> getSCC() {
  vis = sccId = vector<int>(2 * N, -1);
  for (int i = 0; i < 2 * N; i++)
    if (vis[i] == -1) dfs(i);
  return sccId;
}
vector<int>solve2SAT() {
  vector<int>label = getSCC();
  for (int i = 0; i < 2 * N; i += 2)
    if (label[i] == label[i + 1]) return vector<int>();
  vector<ii>ord;
  for (int i = 0; i < 2 * N; i++)
    ord.push_back(ii(-label[i], i));

  sort(ord.begin(), ord.end());
  vector<int>variable(N, -1);
  for (int i = 0; i < 2 * N; i++) {
    int var = ord[i].second;
    bool isTrue = (var % 2 == 0);
    if (variable[var / 2] != -1) continue;
    variable[var / 2] = !isTrue;
  }
  return variable;
}

int main() {
  cin >> N >> M;
  for (int i = 0; i < M; i++) {
    int u, v; cin >> u >> v;
    if (u > 0) u = (u - 1) * 2;
    else u = -2 * u - 1;
    if (v > 0) v = (v - 1) * 2;
    else v = -2 * v - 1;
    adj[oppo(u)].push_back(v); adj[oppo(v)].push_back(u);
  }
  vector<int>ans = solve2SAT();
  if (ans.empty()) return cout << "0", 0;
  for (int &x : ans) cout << x << ' ';
}
```

SCC with kosaraju’s algorithm

```
const int mx = 101010;
int n, m;
int deg[mx], who[mx];
bool vis1[mx], vis2[mx];
vector<int> adj[mx], radj[mx], scc_adj[mx];
stack<int> s;

void dfs1(int u) {
   vis1[u] = true;
   for (int v : adj[u])
      if (!vis1[v]) dfs1(v);
```

```
   s.push(u);
}

void dfs2(int u, int rep) {
   vis2[u] = true;
   who[u] = rep;
   for (int v : radj[u])
      if (!vis2[v]) dfs2(v, rep);
}

void make_SCC() {
   for (int i = 0; i < m; i++) {
      int u, v; cin >> u >> v;
      adj[u].pb(v);
      radj[v].pb(u);
   }
   for (int i = 0; i < m; i++) {
      int u, v; cin >> u >> v;
      adj[u].pb(v);
      radj[v].pb(u);
   }
   for (int i = 1; i <= n; i++)
      if (!vis1[i]) dfs1(i);
   while (!s.empty()) {
      int u = s.top(); s.pop();
      if (!vis2[u]) dfs2(u, u);
   }
   for (int i = 1; i <= n; i++) {
      for (int j : adj[i]) {
         if (who[i] != who[j]) scc_adj[who[i]].pb(who[j]);
      }
   }
}
```

Segment Tree & lazy propagation (by recursion)

```
using lint = long long;
const int mxn = 100010;
int N;
struct seg {
  lint a[mxn*4], lz[mxn*4], lim;
  void init() {   //don't forget to call init func
    for (lim = 1; lim < N; lim <<= 1);
    memset(a, 0, sizeof(a));
    memset(lz, 0, sizeof(lz));
    for (int i = 0; i < N; i++)
      cin >> a[i + lim];
    for (int i = lim - 1; i; i--)
      a[i] = a[i * 2] + a[i * 2 + 1];
  }
  void lazy(int n, int nl, int nr) {
    if (lz[n]) {
      a[n] += lz[n] * (nr - nl + 1);
      if (n < lim) {
        lz[n * 2] += lz[n];
        lz[n * 2 + 1] += lz[n];
      }
      lz[n] = 0;
    }
  }
  void update(lint l, lint r, lint diff) { update(l, r, 1, 0, lim - 1, diff); };
  void update(lint l, lint r, int n, int nl, int nr, lint diff) {
    lazy(n, nl, nr);
    if (r < nl || nr < l) return;
    if (l <= nl && nr <= r) {
      lz[n] += diff;
      lazy(n, nl, nr);
      return;
    }
    int mid = (nl + nr) / 2;
    update(l, r, n * 2, nl, mid, diff);
    update(l, r, n * 2 + 1, mid + 1, nr, diff);
    a[n] = a[n * 2] + a[n * 2 + 1];
  }
  lint sum(lint l, lint r) { return sum(l, r, 1, 0, lim - 1); }
  lint sum(lint l, lint r, int n, int nl, int nr) {
    lazy(n, nl, nr);
    if (r < nl || nr < l) return 0;
    if (l <= nl && nr <= r) return a[n];
    int mid = (nl + nr) / 2;
    return sum(l, r, n * 2, nl, mid) + sum(l, r, n * 2 + 1, mid + 1, nr);
  }
}seg;
```

Splay Tree

```
const lint INF = 1e22 + 7, mxn = 100010;
struct node {
  node *l, *r, *p;
  bool inv, dummy;
  lint v, cnt, sum, lazy, mx, mn;
} *tree;

int a[mxn], N, Q;
node* ptr[mxn];
```

```
void Update(node *x) {
  x->cnt = 1;
  x->sum = x->v;
  x->mn = x->mx = x->v;
  if (x->l) {
    x->cnt += x->l->cnt;
    x->sum += x->l->sum;
    x->mn = min(x->mn, x->l->mn);
    x->mx = max(x->mx, x->l->mx);
  }
  if (x->r) {
    x->cnt += x->r->cnt;
    x->sum += x->r->sum;
    x->mn = min(x->mn, x->r->mn);
    x->mx = max(x->mx, x->r->mx);
  }
}

void Lazy(node *x) {
  if (!x->inv) return;
  node *t = x->l;
  x->l = x->r;
  x->r = t;
  x->inv = false;
  if (x->l) x->l->inv = !x->l->inv;
  if (x->r) x->r->inv = !x->r->inv;
}

void Rotate(node *x) {
  node *p = x->p;
  node *b;
  Lazy(p);
  Lazy(x);
  if (x == p->l) {
    p->l = b = x->r;
    x->r = p;
  }
  else {
    p->r = b = x->l;
    x->l = p;
  }
  x->p = p->p;
  p->p = x;
  if (b) b->p = p;
  (x->p ? p == x->p->l ? x->p->l : x->p->r : tree) = x;
  Update(p);
  Update(x);
}
```

```
void Splay(node *x) {
  while (x->p) {
    node *p = x->p;
    node *g = p->p;
    if (g) Rotate((x == p->l) == (p == g->l) ? p : x);
    Rotate(x);
  }
}

void Initialize(int N) {
  node *x;
  if (tree) delete tree;
  ptr[0] = tree = x = new node;
  x->l = x->r = x->p = NULL;
  x->dummy = x->cnt = 1; x->inv = 0;
  x->sum = x->v = -INF;

  for (int i = 1; i <= N; i++) {
    ptr[i] = x->r = new node;
    x->r->p = x;
    x = x->r;
    x->l = x->r = NULL;
    x->cnt = 1; x->inv = x->dummy = 0;
    x->sum = x->v = a[i];
    //x = x->r;
  }
  ptr[N + 1] = x->r = new node;
  x->r->p = x;
  x = x->r;
  x->l = x->r = NULL;
  x->cnt = 1; x->dummy = 1;
  x->sum = x->v = INF;
  for (int i = N; i >= 1; i--)
    Update(ptr[i]);
  Splay(ptr[1]);
}

void Insert(int v) {
  node *p = tree, **pp;
  if (!p) {
    node *x = new node;
    tree = x;
    x->l = x->r = x->p = NULL;
    x->v = v;
    return;
  }
  while (1) {
    if (v == p->v) return;
    if (v < p->v) {
```

```
      if (!p->l) {
        pp = &p->l;
        break;
      }
      p = p->l;
    }
    else {
      if (!p->r) {
        pp = &p->r;
        break;
      }
      p = p->r;
    }
  }
  node *x = new node;
  *pp = x;
  x->l = x->r = NULL;
  x->p = p;
  x->v = v;
  Splay(x);
}

bool Find(lint v) {
  node *p = tree;
  if (!p) return false;
  while (p) {
    if (v == p->v) break;
    if (v < p->v) {
      if (!p->l) break;
      p = p->l;
    }
    else {
      if (!p->r) break;
      p = p->r;
    }
  }
  Splay(p);
  return v == p->v;
}

void Find_Kth(int k) {
  node *x = tree;
  Lazy(x);
  while (1) {
    while (x->l && x->l->cnt > k)
      x = x->l, Lazy(x);
    if (x->l) k -= x->l->cnt;
    if (!k--) break;
    x = x->r;
    Lazy(x);
  }
  Splay(x);
}

node* Interval(int l, int r) {
  Find_Kth(l - 1);
  node *x = tree;
  tree = x->r;
  tree->p = NULL;
  Find_Kth(r - l + 1);
  x->r = tree;
  tree->p = x;
  tree = x;
  return tree->r->l;
}

void Add(int l, int r, int z) {
  Interval(l, r);
  node *x = tree->r->l;
  x->sum += x->cnt * z;
  x->lazy += z;
}

lint Sum(int l, int r) {
  Interval(l, r);
  return tree->r->l->sum;
}

void Reverse(int l, int r) {
  Interval(l, r);
  node *x = tree->r->l;
  x->inv = !x->inv;
}

void revolve(int l, int r, int T) {  // rotate l~r T times
  if (l >= r || !T) return;
  int l = (r - l + 1);
  T = (T%l + l) % l;
  Reverse(l, r - T);
  Reverse(r - T + 1, r);
  Reverse(l, r);
}

void print(node*cur) {
  Lazy(cur);
  if (cur->l) print(cur->l);
  if (!cur->dummy) cout << cur->v << ' ';
  if (cur->r) print(cur->r);
```

```
}
```

Graph Algorithms

Shortest path (Bellman-ford, Dijkstra, 0-1 BFS)

```
void bellman-Ford() {
  for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= N; j++) {
      if (dist[j] == INF) continue;
      for (auto it : adj[j]) {
        int next = it.first, d = it.second;
        if (dist[j] != INF && dist[next] > dist[j] + d) {
          dist[next] = dist[j] + d;
          if (i == N) {
            isminus = true;
            return;
} } } } } }
void dijkstra(int S) {
  priority_queue<pll> pq;
  fill(dist, dist + N, INF);
  dist[S] = 0;
  pq.push({ 0,S });
  while (!pq.empty()) {
    int cur = pq.top().second;
    lint d = -pq.top().first;
    pq.pop();
    if (dist[cur] < d) continue;
    for (auto &it : adj[cur]) {
      int nxt = it.first, w = it.second;
      if (dist[nxt] > dist[cur] + w) {
        prv[nxt] = cur;
        dist[nxt] = dist[cur] + w;
        pq.push({ -dist[nxt], nxt });
} } } }
void z1BFS(int S) {
  fill(dist, dist + N, INF);
  dist[S] = 0;
  deque<int> q;
  q.push_front(S);
  while (!q.empty()) {
    int cur = q.front();
    q.pop_front();
    for (auto &e : adj[cur]) {
      int nxt = e.first, w = e.second;
      int w = e.second;
      if (dist[cur] + w < dist[nxt]) {
        dist[nxt] = dist[cur] + w;
        if (w == 1) q.push_back(nxt);
        else q.push_front(nxt);
} } } }
```

Centroid decomposition & tree isomorphism

```
#include <bits/stdc++.h>
using namespace std;
#define all(v) (v).begin(),(v).end()
const int mxn = 100010;
typedef long long ll;
typedef vector<int> vi;
typedef vector<vi> vvi;
typedef pair<vi, int> pvi;

vvi adj[2]; //입력 트리
vvi g[2]; //센트로이드를 루트로 하는 트리에서 깊이별로 정점 분류
int sz[mxn]; //sz[i] = 입력받은 트리에서 i를 루트로 하는 서브트리의 크기
int par[2][mxn]; //par[id][i] = 센트로이드를 루트로 하는 트리에서 i의 부모
int label[2][mxn]; //label[id][i] = i를 renumbering 할 때의 번호d
vi cent[2]; //트리의 센트로이드(1 or 2개)
int N; //트리 정점 개수

//centroid 구하기, 1개 혹은 2개
int getCent(int id, int v, int p) { //tree id, vertex, parent
  int ch = 0;
  for (auto i : adj[id][v]) if (p != i) {
    int now = getCent(id, i, v);
    if (now > (N / 2)) break;
    ch += now;
  }
  if (N - ch - 1 <= N / 2) cent[id].push_back(v);
  return sz[v] = ch + 1;
}

//센트로이드를 루트로 하는 트리 생성, 깊이 반환
int dfs(int id, int v, int p, int d) { //tree id, vertex, parent, depth
  par[id][v] = p; g[id][d].push_back(v);
  int mx = d;
  for (auto i : adj[id][v]) if (i != p) {
    mx = max(mx, dfs(id, i, v, d + 1));
  }
  return mx;
}

int chk(int _lv) {
  for (int lv = _lv - 1; lv >= 0; lv--) {
    vector<pvi> tup[2];
```

```
    for (int id = 0; id < 2; id++) {
      for (auto i : g[id][lv]) {
        //깊이가 lv인 i의 자식들로 튜플 생성 - renumbering된 값을 넣어줌
        tup[id].emplace_back(vi(), i);
        for (auto j : adj[id][i])
          if (par[id][i] != j) tup[id].back().first.push_back(label[id][j]);
      }
    }
    //튜플 크기 다르면 false
    if (tup[0].size() != tup[1].size()) return 0;

    for (int id = 0; id < 2; id++) {
      for (auto &i : tup[id]) sort(i.first.begin(), i.first.end());
      sort(tup[id].begin(), tup[id].end());
    }

    int pv = 0;
    for (int i = 0; i < tup[0].size(); i++) {
      if (tup[0][i].first != tup[1][i].first) return 0;

      //이전 값과 같다면 같은 숫자로 renumbering
      if (i != 0 && tup[0][i].first == tup[0][i - 1].first)
        label[0][tup[0][i].second] = label[1][tup[1][i].second] = pv;
      else label[0][tup[0][i].second] = label[1][tup[1][i].second] = ++pv;
    }
  }
  return 1;
}

void init() {
  memset(sz, 0, sizeof(int) * (N + 2));
  for (int i = 0; i < 2; i++) {
    adj[i].clear(), cent[i].clear(), g[i].clear();
    memset(label[i], 0, sizeof(int) * (N + 2));
    memset(par[i], 0, sizeof(int) * (N + 2));
  }
}

int solve() {
  for (int id = 0; id < 2; id++)
    getCent(id, 1, -1);
  if (cent[0].size() != cent[1].size())
    return 0;
  if (cent[0].size() == 2) {
    N++;
    for (int id = 0; id < 2; id++) {
      for (int j = 0; j < 2; j++) {
        auto it = remove(all(adj[id][cent[id][j]]), cent[id][!j]);
        adj[id][cent[id][j]].erase(it, adj[id][cent[id][j]].end());
        adj[id][cent[id][j]].push_back(N);
        adj[id][N].push_back(cent[id][j]);
      }
      cent[id][0] = N;
    }
  }
  int t[2];
  for (int id = 0; id < 2; id++)
    t[id] = dfs(id, cent[id][0], -1, 0);
  if (t[0] != t[1]) return 0;

  if (chk(t[0])) return 1;
  return 0;
}
```

Euler path / circuit

```
vector<int> euler;
void findEulerianCircuit(int from) {
  for (int to = 0; to < N; to++) {
    while (adj[from][to]) {
      adj[from][to]--;
      adj[to][from]--;
      findEulerianCircuit(to);
    }
  }
  euler.push_back(from);
}
```

Diamater of tree

```
int N, x, y, mxd;
vector<ii> adj[mxn];
void dfs(int cur, int prev, int dist) {
  if (mxd < dist) {
    mxd = dist, x = cur;
  }
  for (auto&it : adj[cur]) {
    int next = it.first, d = it.second;
    if (next == prev) continue;
    dfs(next, cur, dist + d);
  }
}
dfs(1, -1, 0);  //1 or arbitrary node
y = x; dfs(x, -1, 0);  diameter = mxd
```

Mathematical Stuff

Euler phi function

```
vector<ll> prime;
ll euler(ll n) {
   ll ret = 1;
   for (ll p : prime) {
      ll pow = 1;
      while (n%p == 0) {
         n /= p;
         pow *= p;
      }
      if (pow != 0) {
         ret *= (pow - (pow / p));
      }
   }
   if (n != 1) ret *= (n - 1);
   return ret;
}
```

Extended euclidean

p*a + q*b = 1 을 만족하는 (p,q) 특수해를 찾는다.
a,b가 서로소여야 한다는 것에 주의

```
pll euclidean(ll a, ll b) {
  if ((a - 1) % b == 0) return { 1, -((a - 1) / b) };
  else {
    pll p = euclidean(b, a%b);
    ll k = a / b, c1 = p.first, c2 = p.second;
    return { c2, c1 - c2 * k };
  }
}
```

Fast fourier transform

$c_k = \sum_{i+j=k} a_i b_i$.

```
typedef complex <double> base;

void fft(vector <base> &a, bool invert) {
  int n = sz(a);
  for(int i=1, j=0; i<n; i++) {
    int bit = n >> 1;
    for(; j>=bit; bit>>=1) j -= bit;
    j += bit;
    if(i < j) swap(a[i], a[j]);
  }
  for(int len=2; len<=n; len<<=1) {
    double ang = 2*M_PI/len*(invert?-1:1);
    base wlen(cos(ang), sin(ang));
    for(int i=0; i<n; i+=len) {
      base w(1);
      for(int j=0; j<len/2; j++) {
        base u = a[i+j], v = a[i+j+len/2]*w;
        a[i+j] = u+v;
        a[i+j+len/2] = u-v;
        w *= wlen;
      }
    }
  }
  if(invert) for(int i=0; i<n; i++) a[i] /= n;
}

void multiply(const vector<int> &a, const vector<int> &b, vector<int> &res) {
  vector <base> fa(all(a)), fb(all(b));
  int n = 1;
  while(n < max(sz(a), sz(b))) n <<= 1;
  n<<=2;
  fa.resize(n); fb.resize(n);
  fft(fa, false); fft(fb, false);
  for(int i=0; i<n; i++) fa[i] *= fb[i];
  fft(fa, true);
  res.resize(n);
  for(int i=0; i<n; i++) res[i] = int(fa[i].real()+(fa[i].real()>0?0.5:-0.5));
}
```

Fast nCk

```
const int MAX = 4000000;
ll fact[MAX + 1], inv[MAX + 1];

ll fastpow(ll a, ll pow) {
  if (pow == 0) return 1LL;
  ll ret = fastpow(a, pow / 2);
  ret = (ret*ret) % DIV;
  return pow % 2 == 1 ? (ret*a) % DIV : ret;
}

void init() {
  fact[0] = fact[1] = 1;
  for (ll i = 2; i <= MAX; i++) fact[i] = (fact[i - 1] * i) % DIV;
```

```
  inv[MAX] = fastpow(fact[MAX], DIV - 2);
  for (ll i = MAX - 1; i >= 0; i--) inv[i] = (inv[i + 1] * (i + 1)) % DIV;
}

ll comb(int n, int k) {
  ll res = fact[n];
  res *= inv[k]; res %= DIV;
  res *= inv[n - k];
  return res % DIV;
}
```

Matrix

```
struct Matrix {
  vector<vector<double>> v;
  int N;
  Matrix(int n) : N(n) { v.resize(N, vector<double>(N, 0)); }
  ~Matrix() {
    for (int i = 0; i < N; i++) v[i].clear();
    v.clear();
  }
  Matrix identity(int n) {
    Matrix ret = Matrix(n);
    for (int i = 0; i < n; i++)
      ret.v[i][i] = 1;
    return ret;
  }
  void swapRow(int i, int j) {
    if (i == j) return;
    for (int k = 0; k < N; k++)
      swap(v[i][k], v[j][k]);
  }
  double *operator[](int i) { return &v[i][0]; }
};

//rref of xor-ing bits, xor maximization
int64_t basis[60], x;

void computeREF(int64_t x) {
  for (int i = 59; i >= 0; i--) {
    if ((x >> i) & 1) {
      if (!basis[i]) {
        basis[i] = x;
        return;
      }
      else x ^= basis[i]; // elimination
} } }
for (int i = 59; i >= 0; i--)
  x = max(x, x ^ basis[i]);
```

Miller-rabin primality test & Pollard-rho factorization

```
using lint = unsigned long long;
vector<lint> a = { 2,3,5,7,11,13,17,19,23,29, 31, 37, 9780504, 12, 325, 9375,
                   28178, 450775, 1795265022 };
vector<int> plist;
bool isprime[1000100];

lint abs2(lint a, lint b) {
  if (a > b) return a - b;
  return b - a;
}

lint gcd(lint a, lint b) {
  return b == 0 ? a : gcd(b, a%b);
}

lint fac(lint n, lint mod) {
  if (n == 1) return 1;
  return (n * fac(n - 1, mod))%mod;
}

void sieve() {
  fill(isprime, isprime + 1000100, true);
  isprime[0] = isprime[1] = false;
  for (int i = 2; i <= 1000000; i++) {
    if (!isprime[i]) continue;
    plist.push_back(i);
    for (int j = i * 2; j <= 1000000; j += i)
      isprime[j] = false;
  }
}

inline lint addmod(lint x, lint y, lint m) {
  x %= m;
  y %= m;
  return (x >= m - y ? x - (m - y) : x + y);
}

inline lint mulmod(lint x, lint y, lint m) {
  x %= m;
  y %= m;
```

```
  lint r = 0;
  while (y > 0) {
    if (y % 2 == 1)
      r = addmod(r, x, m);
    x = addmod(x, x, m);
    y /= 2;
  }
  return r;
}

lint exp(lint a, lint b, lint mod) {
  lint ret = 1;
  while (b) {
    if (b % 2) ret = mulmod(ret, a, mod);
    //if (b % 2) ret = (ret * a) % mod;
    a = mulmod(a, a, mod);
    //a = (a*a) % mod;
    b /= 2;
  }

  return ret;
}

bool miller_labin(lint n, lint a) {
  lint d = n - 1;
  while (d % 2 == 0) {
    if (exp(a, d, n) == n - 1) return true;
    d /= 2;
  }
  lint val = exp(a, d, n);
  if (val == 1 || val == n - 1) return true;

  return false;
}

bool chk(lint n, vector<lint> alist) {
  if (n <= 1'000'000 && isprime[n]) return true;
  for (lint it : alist)
    if (!miller_labin(n, it))
      return false;
  return true;
}

lint PollardRho(lint n) {
  lint x = rand() % (n - 2) + 2;
  lint y = x;
  lint 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;
    lint d = gcd(abs2(x, y), n);
    if (d == 1) continue;
    //if (!chk(d, a)) return PollardRho(d);
    return d;
  }
}

int main() {
  //freopen("input.txt", "r", stdin);
  vector<lint> ans;
  lint n;
  cin >> n;
  sieve();
  for (lint div : plist) {
    if (n < div) break;
    int tmp = 1;
    if (n%div == 0) {
      while (n%div == 0) {
        ans.push_back(div);
        n /= div;
        tmp++;
      }
    }
  }
  if (n == 1);
  else if (chk(n, a)) ans.push_back(n);
  else {
    lint d = PollardRho(n);
    ans.push_back(min(d, n / d));
    ans.push_back(max(d, n / d));
  }
  for (lint x : ans)
    cout << x << '\n';
}
```

Mobius inversion

mobius[n] = 1 (n == 1) 주의!
0 (n이 어떤 소수 p에 대해 p^2|n 일 때)
$(-1)^r$ (n=$p_1 * p_2 * \ldots * p_r$ 꼴일 때, $p_1, \ldots, p_r$은 prime)

```
const int mx = 10000001;
```

```
int mobius[mx];
void init() {
  fill(mobius, mobius + mx, 1);
  for (int i = 2; i * i <= mx; i++)
    if (mobius[i] == 1) {
      for (int j = i; j <= mx; j += i) mobius[j] *= -i;
      for (int j = i * i; j <= mx; j += i * i) mobius[j] = 0;
    }

  for (int i = 2; i <= mx; i++) {
    if (mobius[i] == i) mobius[i] = 1;
    else if (mobius[i] == -i) mobius[i] = -1;
    else if (mobius[i] < 0) mobius[i] = 1;
    else if (mobius[i] > 0) mobius[i] = -1;
  }
}
```

Geometry

General Geometry Library Header

```
using ll = long long;
using pdd = pair<double, double>;

struct Point {
    int x, y;
    Point (int xx, int yy): x(xx), y(yy) {}
    Point () { x = 0, y = 0; }
    //
    Point operator + (const Point& rhs) { return Point(x + rhs.x, y + rhs.y); }
    Point operator - (const Point& rhs) { return Point(x - rhs.x, y - rhs.y); }
    Point operator * (const int& rhs) { return Point(x * rhs, y * rhs); }
    Point operator / (const int& rhs) { return Point(x / rhs, y / rhs); }
    bool operator < (const Point& rhs) { return x == rhs.x ? y < rhs.y : x <
rhs.x; }
    // inner product
    ll inner (const Point& rhs) { return 1LL * x * rhs.x + 1LL * y * rhs.y; }
    // outer product
    ll cross (const Point& rhs) { return 1LL * x * rhs.y - 1LL * y * rhs.x; }
    // distance between two point
    ll dist (const Point& rhs) {
        Point tmp = *this - rhs;
        return 1LL * tmp.x * tmp.x + 1LL * tmp.y * tmp.y;
    }
};
struct Line {
    Point s, e;
    Line (Point ss, Point ee): s(ss), e(ee) {}
    Line () { s = Point(), e = Point(); }
};

// ccw
// counter-clock-wise : 1
// parallel : 0
// clock-wise : -1

// using 3 point a->b->c
int ccw (Point a, Point b, Point c) {
    Point ab = b - a, bc = c - b;
    ll res = ab.cross(bc);
    if (res > 0) return 1;
    else if (res == 0) return 0;
    else return -1;
}
// using 4 point a->b , c->d
int ccw (Point& a, Point& b, Point& c, Point& d) {
    return ccw(a, b, d - c + b);
}
// using 2 line a:a.s->a.e , b:b.s->b.e
int ccw (Line a, Line b) {
    return ccw(a.s, a.e, b.e - b.s + a.e);
}

// intersect
// return flag which line intersect
bool intersect (Line a, Line b) {
    int ccw_ab = ccw(a.s, a.e, b.s) * ccw(a.s, a.e, b.e);
    int ccw_ba = ccw(b.s, b.e, a.s) * ccw(b.s, b.e, a.e);
    if (ccw_ab <= 0 && ccw_ba <= 0) {
        if (!ccw_ab && !ccw_ba) {
            if (a.s.x == b.s.x) {
                if (max(a.s.y, a.e.y) < min(b.s.y, b.e.y)) return false;
                if (max(b.s.y, b.e.y) < min(a.s.y, a.e.y)) return false;
            }
            else {
                if (max(a.s.x, a.e.x) < min(b.s.x, b.e.x)) return false;
                if (max(b.s.x, b.e.x) < min(a.s.x, a.e.x)) return false;
            }
        }
        return true;
    }
```

```
    return false;
}

vector<Point> P;
vector<int> Convex_Hull;

// Angular sort using pivot point(P[0])
bool angle_cmp(Point& a, Point& b) {
    if (ccw(P[0], a, b) == 0) {
        if (a.x == b.x)
            return a.y - P[0].y < b.y - P[0].y;
        else return a.x - P[0].x < b.x - P[0].x;
    }
    else return ccw(P[0], a, b) == 1;
}
```

Area series (Shoelace, Brahmagupta's, Heron's)

```
double shoelace(vector<P> a) {
  double ret = 0; int N = a.size();
  for (int i = 0; i < N; i++)
    ret += a[i].x * a[(i + 1) % N].y - a[(i + 1) % N].x * a[i].y;
  return abs(ret / 2);
}
```

$$S = \sqrt{(s-a)(s-b)(s-c)(s-d)} \quad where \quad s = \frac{a+b+c+d}{2}$$

$$c^2 = a^2 + b^2 - 2ab\cos\theta$$

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R$$

Convex Hull (Graham scan, Monotone chain)

```
void graham_scan(int n) {
    swap(P[0], *min_element(P.begin(), P.end()));

    sort(P.begin() + 1, P.end(), angle_cmp);

    for (int i = 0; i < n; ++i) {
        while (Convex_Hull.size() >= 2) {
            int sz = Convex_Hull.size();
            if (ccw(P[Convex_Hull[sz - 2]], P[Convex_Hull[sz - 1]], P[i]) < 1)
                Convex_Hull.pop_back();
            else break;
        }
        Convex_Hull.push_back(i);
    }
}
// monotone chain
vector<P> arr, Low, Upp;
vector<P> Monotone_chain(vector<P>&a) {
  Low.clear(); Upp.clear();
  sort(all(a));
  for (int i = 0; i < sz(a); i++) {
    while (sz(Low) >= 2 && ccw(Low[sz(Low) - 2], Low[sz(Low) - 1], a[i]) <= 0)
      Low.pop_back();
    Low.push_back(a[i]);
  }
  for (int i = sz(a) - 1; i >= 0; i--) {
    while (sz(Upp) >= 2 && ccw(Upp[sz(Upp) - 2], Upp[sz(Upp) - 1], a[i]) <= 0)
      Upp.pop_back();
    Upp.push_back(a[i]);
  }
  Low.pop_back(); Upp.pop_back();
  Low.insert(Low.end(), all(Upp));

  return Low;
}
```

Rotating Calipers

```
ll rotating_calipers() {
    int m = Convex_Hull.size();
    ll ret = -1;

    for (int i = 0, j = 1; i < m; ++i) {
        while (ccw(P[Convex_Hull[i]], P[Convex_Hull[(i + 1) % m]],
P[Convex_Hull[j]], P[Convex_Hull[(j + 1) % m]]) > 0)
            j = (j + 1) % m;

        ll tmp = P[Convex_Hull[i]].dist(P[Convex_Hull[j]]);
        ret = max(ret, tmp);
    }
    return ret;
}
```

Flows

Dinic's Algorithm

```
#include <bits/stdc++.h>
using namespace std;

const int mxn = 1818, INF = 1e9 + 7;
int N, M, C, sr, sc;

namespace flow {
    struct edge {
        int u, r, c, f;
        edge() {}
        edge(int u, int r, int c, int f = 0) :u(u), r(r), c(c), f(f) {}
        inline int residual() { return c - f; }
    };
    vector<edge> adj[mxn];
    int S, T, lv[mxn], vis[mxn];
    bitset<mxn> viss;
    inline void addedge(int u, int v, int c) {
        edge e1(v, adj[v].size(), c);
        edge e2(u, adj[u].size(), c);
        adj[u].push_back(e1);
        adj[v].push_back(e2);
    }
    void init(int id) {
        memset(vis, 0, sizeof vis);
        memset(lv, 0, sizeof lv);

    }
    int bfs() {
        memset(lv, 0, sizeof lv);
        queue<int> q;
        q.push(S);
        lv[S] = 1;
        while (!q.empty()) {
            int cur = q.front(); q.pop();
            for (auto&e : adj[cur]) {
                int nxt = e.u;
                if (e.residual() > 0 && !lv[nxt]) {
                    lv[nxt] = lv[cur] + 1;
                    if (nxt != T) q.push(nxt);
                }
            }
        }
        return lv[T];
    }
    int dfs(int cur, int flow) {
        if (cur == T) return flow;
        int sz = (int)adj[cur].size();
        for (int&i = vis[cur]; i < sz; i++) {
            edge &e = adj[cur][i];
            int nxt = e.u;
            if (e.residual() > 0 && lv[nxt] == lv[cur] + 1) {
                int ret = dfs(nxt, min(e.residual(), flow));
                if (ret > 0) {
                    e.f += ret;
                    adj[nxt][e.r].f -= ret;
                    return ret;
                }
            }
        }
        return 0;
    }
    int dinic() {
        int ret = 0, fl;
        while (bfs()) {
            memset(vis, 0, sizeof vis);
            while (fl = dfs(S, INF)) ret += fl;
        }
        return ret;
    }
    void getAB() {    // 0 for T sets, 1 for S sets
        viss.reset();
        queue<int> q;
        viss[S] = 1;
        q.push(S);
        while (!q.empty()) {
            int cur = q.front(); q.pop();
            for (auto&it : adj[cur]) {
                if (!viss[it.u] && it.residual() > 0) {
                    viss[it.u] = 1;
                    q.push(it.u);
                }
            }
        }
        for (int i = 1; i <= N; i++)
            if (viss[i]) cout << i << ' ';
        cout << '\n';
        for (int i = 1; i <= N; i++)
            if (!viss[i]) cout << i << ' ';
        cout << '\n';
    }
}
```

Hopcroft-karp Maximum matching

```
namespace Matching{
//matching [1...n] <-> [1...m]
const int MX = 40040, MY = 40040;
vector <int> E[MX];
int xy[MX], yx[MY];
int n, m;

void addE(int x, int y) { E[x].pb(y); }
void setnm(int sn, int sm) { n = sn; m = sm; }

int tdis[MX], que[MX], *dis = tdis + 1;
int bfs() {
  int *fr = que, *re = que;
  for(int i=1;i<=n;i++) {
    if(xy[i] == -1) *fr++ = i, dis[i] = 0;
    else dis[i] = -1;
  }
  dis[-1] = -1;
  while(fr != re) {
    int t = *re++;
    if(t == -1) return 1;
    for(int e : E[t]) {
      if(dis[yx[e]] == -1) dis[yx[e]] = dis[t] + 1, *fr++ = yx[e];
    }
  }
  return 0;
}

int dfs(int x) {
  for(int e : E[x]) {
    if(yx[e] == -1 || (dis[yx[e]] == dis[x] + 1 && dfs(yx[e]))) {
      xy[x] = e;
      yx[e] = x;
      return 1;
    }
  }
  dis[x] = -1;
  return 0;
}

int Do() {
  memset(xy, -1, sizeof xy);
  memset(yx, -1, sizeof yx);

  int ans = 0;
  while(bfs()) {
    for(int i=1;i<=n;i++) if(xy[i] == -1 && dfs(i)) ++ans;
  }
  return ans;
}

void solve(){
  int n, m;
  scanf("%d%d", &n, &m);
  Matching::setnm(n, m);
  for(int i=1;i<=n;i++) {
    int x; scanf("%d", &x);
    while(x--) {
      int y; scanf("%d", &y);
      Matching::addE(i, y);
    }
  }
  printf("%d\n", Matching::Do());
}
```

Hungarian Method

```
int in[505][505];

int mats[505], matt[505];
int Ls[505], Lt[505];
int revs[505], revt[505];
int valt[505];

bool chks[505], chkt[505];

vector <int> Vu;
void vpush(int p, int N) {
  chks[p] = true;
  for (int i = 1; i <= N; i++) {
    if (!valt[i]) continue;
    if (valt[i] > Ls[p] + Lt[i] - in[p][i]) {
      valt[i] = Ls[p] + Lt[i] - in[p][i];
      revt[i] = p;
      if (!valt[i]) Vu.push_back(i);
    }
  }
}
int main() {
  int N, i, j, k;
  scanf("%d", &N);
  for (i = 1; i <= N; i++) {
    for (j = 1; j <= N; j++) {
      scanf("%d", &in[i][j]);
      in[i][j] = -in[i][j];
```

```
    }
  }

  for (i = 1; i <= N; i++) Lt[i] = -INF;
  for (i = 1; i <= N; i++) for (j = 1; j <= N; j++) Lt[j] = max(Lt[j], in[i][j]);
  for (i = 1; i <= N; i++) {
    for (j = 1; j <= N; j++) chks[j] = chkt[j] = false;
    for (j = 1; j <= N; j++) valt[j] = INF;
    for (j = 1; j <= N; j++) revs[j] = revt[j] = 0;

    int p = 0;
    for (j = 1; j <= N; j++) if (!mats[j]) break;
    p = j;
    vpush(p, N);

    while (1) {
      if (!Vu.empty()) {
        int t = Vu.back();
        Vu.pop_back();
        chkt[t] = true;

        if (!matt[t]) {
          vector <int> Vu2;
          Vu2.push_back(t);
          while (1) {
            Vu2.push_back(revt[Vu2.back()]);
            if (Vu2.back() == p) break;
            Vu2.push_back(revs[Vu2.back()]);
          }
          reverse(all(Vu2));
          for (j = 0; j < Vu2.size(); j += 2) {
            int s = Vu2[j], t = Vu2[j + 1];
            mats[s] = t;
            matt[t] = s;
          }
          break;
        }
        else {
          int s = matt[t];
          revs[s] = t;
          vpush(s, N);
        }
      }
      else {
        int mn = INF;
        for (j = 1; j <= N; j++) if (!chkt[j]) mn = min(mn, valt[j]);
        for (j = 1; j <= N; j++) {
          if (chks[j]) Ls[j] -= mn;
          if (chkt[j]) Lt[j] += mn;
          else {
            valt[j] -= mn;
            if (valt[j] == 0) Vu.push_back(j);
          }
        }
      }
    }
    Vu.clear();
  }

  int ans = 0;
  for (i = 1; i <= N; i++) ans += Ls[i] + Lt[i];
  return !printf("%d\n", -ans);
}
```

Minimum cost Maximum flow

```
const int mxn = 802, INF = 1e9;
int N, M, S, E, cnt;
int c[mxn][mxn], f[mxn][mxn], d[mxn][mxn];
vector<int> adj[mxn];

void addedge(int u, int v, int cap, int dist) {
  c[u][v] = cap, d[u][v] = dist, d[v][u] = -dist;
  adj[u].push_back(v);
  adj[v].push_back(u);
}

inline int residual(int u, int v) { return c[u][v] - f[u][v]; }

int MCMF() {     //maximum cost를 구한다면 간선에 dist, ret 부호 반대로
  int s=S, e=E, ret = 0;
  while (1) {
    int prev[mxn], dist[mxn];
    fill(prev, prev + mxn, -1);
    fill(dist, dist + mxn, INF);
    bool inq[mxn] = { 0 };
    queue<int> q;
    dist[s] = 0, inq[s] = true;
    q.push(s);
    while (!q.empty()) {
      int cur = q.front(); q.pop();
      inq[cur] = false;
      for (int next : adj[cur]) {
        if (residual(cur, next) > 0 && dist[next] > dist[cur] + d[cur][next]) {
          dist[next] = dist[cur] + d[cur][next];
          prev[next] = cur;
          if (!inq[next]) {
            q.push(next);
            inq[next] = true;
          }
```

```
      }
    }
  }

  if (prev[e] == -1) break;

  for (int i = e; i != s; i = prev[i]) {
    ret += d[prev[i]][i];
    f[prev[i]][i]++;
    f[i][prev[i]]--;
  }
  cnt++;
 }
 return ret;
}
```

String

Manacher & Z

가장 긴 팰린드롬 부분 문자열
L(i) = s[i-x,i+x]가 팰린드롬이 되는 최대의 x
z(i) : s와 s[i..]의 공통 접두사의 길이
Both O(N)

```
vector<int> manachers(const string &s){
 int n = (int)s.length();
 vector<int> L(n);
 int r = 0, p = 0;
 for(int i = 0; i < n; i++){
  if(i <= R) L[i] = min(L[2*p-i],r-i);
  while(i - L[i] - 1 >= 0 && i + L[i] + 1 < n
       && s[i - L[i] - 1] == s[i + L[i] + 1]) L[i]++;
  if(R < i + L[i]){
    r = i + L[i];
    p = i;
  }
 }
 return L;
}

int main(){
 int n; cin>>n;
 vector<int> p(n*2+1); p[0] = -1;
 for(int i = 0;i<n;i++)          // |A|B|C|C|B|A|
  cin>>p[i*2+1], p[i*2+2] = -1;
 const auto s = manachers(p);
 int q,l,r; cin>>q;
 while(q--){
  cin>>l>>r; l=l*2-1,r=r*2-1;
  int m=(l+r)/2,d=(r-m);
  cout<<(d<=s[m])<<'\n';
 }
}

vector<int> z(const string &s){
 int n = (int)s.length();
 vector<int> z(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[z[[i]] == s[i+z[i]]) z[i]++;
  if(i + z[i] - 1 > R) L = i, r = i + z[i] - 1;
 }
 return z;
}
```

KMP

```
char str[mxn + 1], pat[mxn + 1];
int strLen, patternLen, fail[mxn];
int fail[mxn];

void getfail() {
 int j = 0;
 for (int i = 1; i < patternLen; i++) {
  while (j > 0 && pat[i] != pat[j]) j = fail[j - 1];
  if (pat[i] == pat[j]) fail[i] = ++j;
 }
}
vector<int> solve() {
 vector<int> ret;
 int j = 0;
 for (int i = 0; i < strLen; i++) {
  while (j > 0 && str[i] != pat[j]) j = fail[j - 1];
  if (str[i] == pat[j]) {
   if (j == patternLen - 1) {
    ret.push_back(i - patternLen + 2);
    j = fail[j];
   }
   else j++;
  }
 }
 return ret;
}
```

Regular expression usage

```
#include <iostream>
#include <string>
#include <regex>
#include <vector>
using namespace std;

int tc;
string pat, str;
smatch m;
// also can use as regex_match(str, regpat);
bool solve() {
  for (int i = 0; i < pat.size(); i++) {
    if (pat[i] == '_') {    // for wild card
      for (char j = 'A'; j <= 'Z'; j++) {
        pat[i] = j;
        regex regpat(pat);
        if (regex_match(str, m, regpat)) {
          return cout << j << '\n', 1;
} } } }
  return false;
}

int main() {
  //freopen("input.txt", "r", stdin);
  for (cin >> tc; tc--;) {
    cin >> pat >> str;
    regex regpat(pat);
    if (regex_match(str, m, regpat)) {
      cout << "_\n";
      continue;
    }
    if (solve()) continue;
    cout << "!\n";
  }
  return 0;
};
```

Trie & aho-corasick

```
const int mxn = 100010, mxc = 26;

namespace aho {
  int ctoi(char c) { return c - 'a'; };
  int trie[mxn][mxc], idx, fail[mxn], term[mxn];
  void init(vector<string>&ts) {
    idx = 0;
    memset(trie, 0, sizeof trie);
    memset(fail, 0, sizeof fail);
    memset(term, 0, sizeof term);
    for(auto &t: ts) {  //insert
      int p = 0;
      for(auto &i : t) {
        int ch = ctoi(i);
        if(!trie[p][ch]) trie[p][ch] = ++idx;
        p = trie[p][ch];
      }
      term[p] = 1;
    }
    queue<int> q;  //get failure function
    for(int i = 0; i < mxc; i++)
      if(trie[0][i]) q.push(trie[0][i]);
    while(!q.empty()) {
      int x = q.front(); q.pop();
      for(int i = 0; i < mxc; 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;
        term[trie[x][i]] += term[p];
        q.push(trie[x][i]);
      }
    }
  }
  int qry(char s[]) {
    int p = 0;
    for(int it = 0; s[it]; it++) {
      int ch = ctoi(s[it]);
      while(p && !trie[p][ch]) p = fail[p];
      p = trie[p][ch];
      if(term[p]) return 1;
    } return 0;
  }
}
```

Suffix array & LCP

```
char S[MAX];
int N, d, sa[MAX], pos[MAX]; // pos: 그룹 번호
bool cmp(int i, int j) {
  if (pos[i] != pos[j]) return pos[i] < pos[j];
  i += d; j += d;
  return (i < N && j < N) ? (pos[i] < pos[j]) : (i > j);
```

```
}

void constructSA() {
  N = strlen(S);
  for (int i = 0; i<N; i++) {
    sa[i] = i; pos[i] = S[i];
  }
  // d를 2배씩 늘려가면서 매번 앞에서부터 d*2글자만 보고 접미사 정렬
  for (d = 1; ; d *= 2) {
    sort(sa, sa + N, cmp);
    int temp[MAX] = { 0 };          // temp: 새로운 그룹 번호

    // 앞에서부터 훑으면서 각 접미사가 서로 다른 그룹에 속할 때마다 그룹 번호 증가시킴
    for (int i = 0; i<N - 1; i++)
      temp[i + 1] = temp[i] + cmp(sa[i], sa[i + 1]);
    for (int i = 0; i<N; i++)       // pos 배열을 temp 배열로 대체
      pos[sa[i]] = temp[i];
                                    // 모든 접미사가 다른 그룹으로 나뉘어졌다면 종료
    if (temp[N - 1] == N - 1) break;
  }
}
void constructLCP() {
                  // pos[i] = S[i:]가 sa의 몇 번째에 있는가 (pos[sa[i]] = i)
                  // 제일 긴 접미사(S)부터 시작한다.
                  // 매 루프마다 k>0이면 k--
  for (int i = 0, k = 0; i<N; i++, k = max(k - 1, 0)) {
    if (pos[i] == N - 1) continue; // 마지막 접미사(길이 1)면 아무것도 안 함
    // 바로 아래 인접한 접미사와 비교하여 앞에서부터 몇 개의 글자가 일치하는지 센다
    for (int j = sa[pos[i] + 1]; S[i + k] == S[j + k]; k++);
    lcp[pos[i]] = k;
  }
}
```

Queries

Heavy-Light decomposition

```
//segment tree required
struct hld {
  vector<int> par, dep, heavy, head, pos;
  int cur_pos = 0;
  void init() {
    par = dep = head = pos = vector<int>(N);
    heavy = vector<int>(N, -1);
    int p;
    for (int i = 1; i < N; i++) {
      cin >> p; p--;
      adj[p].push_back(i);
    }
    dfs(0);
    decompose(0, 0);
    seg.init();
  }
  int dfs(int cur) {
    int ret = 1, mx = 0;
    for (int next : adj[cur]) {
      if (next == par[cur]) continue;
      par[next] = cur, dep[next] = dep[cur] + 1;
      int next_sz = dfs(next);
      ret += next_sz;
      if (mx < next_sz)
        mx = next_sz, heavy[cur] = next;
    }
    return ret;
  }
  void decompose(int cur, int top) {
    head[cur] = top, pos[cur] = cur_pos++;
    if (heavy[cur] != -1)
      decompose(heavy[cur], top);
    for (int next : adj[cur]) {
      if (next != par[cur] && next != heavy[cur])
        decompose(next, next);
    }
  }
  int query(int l, int r) {
    int ret = 0;
    for (; head[l] != head[r]; r = par[head[r]]) {
      if (dep[head[l]] > dep[head[r]]) swap(l, r);
      int tmp = seg.query(pos[head[r]], pos[r]);
      ret = max(ret, tmp);
    }
    if (dep[l] > dep[r]) swap(l, r);
    if (pos[l] != pos[r]) {
      int tmp = seg.query(pos[l]+1, pos[r]);
      ret = max(ret, tmp);
    }
    return !ret;
  }
  void update(int i, int val) {
    int diff = val - seg.tree[pos[i] + seg.lim];
    seg.update(pos[i], diff);
  }
};
```

Offline dynamic connectivity

어떤 간선을 추가해, 제거해, 현재 시점에 어떤 두 점이 연결되어 있는지 여부 판단
분할정복으로 한다.
solve(l,r,edgeset)에서는
timestamp[l,r] 동안 내내 살아있는 edge들에 대해
연결 -> 처리 -> 분할정복 -> 연결 끊기(rollback)
edge set을 만드는 과정은 segment tree의 recursive update와 유사.
seg[node]: node번째가 가리키는 range [nl, nr] 내내 연결되어 있는 간선의 집합

```
typedef struct query {
   int type, u, v;
}query;

typedef struct edge {
   int u, v, s, e;
}edge;

vector<edge> e;
query q[404040];

vector<edge> seg[404040];

int n, m, sz = 1;
int par[101010], dep[101010];
stack<pair<pii, int>> st; //{{u,v},0}: u에 v를 dep증가없이 붙였다.

void update(edge e, int node, int nodel, int noder) {
   int l = e.s, r = e.e;
   if (r < nodel || l > noder) return;
   else if (l <= nodel && noder <= r) {
      seg[node].pb(e);
      return;
   }
   int mid = (nodel + noder) / 2;
   update(e, node * 2, nodel, mid);
   update(e, node * 2 + 1, mid + 1, noder);
   return;
}

int find(int u) {
   if (par[u] == u) return u;
   return find(par[u]);
}

bool merge(int u, int v) {
   u = find(u), v = find(v);
   if (u == v) return false;

   if (dep[u] > dep[v]) {
      par[v] = u;
      st.push({ { u,v },0 });
   }
   else if (dep[u] == dep[v]) {
      par[v] = u;
      st.push({ { u,v },1 });
      dep[u]++;
   }
   else {
      par[u] = v;
      st.push({ { v,u },0 });
   }
   return true;
}

void rollback(int cnt) {
   for (int i = 0; i < cnt; i++) {
      pair<pii, int> cur = st.top(); st.pop();
      int u = cur.first.first, v = cur.first.second, type = cur.second;
      par[v] = v;
      if (type == 1) dep[u]--;
   }
   return;
}

void solve(int node, int nodel, int noder) {
   //live edge connect
   int cnt = 0;
   for (edge e : seg[node]) cnt += merge(e.u, e.v);

   //do something at specific time
   if (nodel == noder) {
      if (q[nodel].type == 3) {
         if (find(q[nodel].u) == find(q[nodel].v)) cout << "1\n";
         else cout << "0\n";
      }
      rollback(cnt);
      return;
   }

   //divide & conquer
   int mid = (nodel + noder) / 2;
   solve(node * 2, nodel, mid);
   solve(node * 2 + 1, mid + 1, noder);

   //rollback
```

```
   rollback(cnt);
}

void ODC() {
   while (sz < m) sz *= 2;
   for (int i = 1; i <= n; i++) {
      par[i] = i;
      dep[i] = 1;
   }
   //edge timestamp 전처리

   for (edge ee : e) update(ee, 1, 1, sz);
   solve(1, 1, sz);
}
```

Persistent segment tree

```
int rt[mxn + 1];
struct Node {
  int l, r, val;
};
vector<Node> tree;

void update(int idx, int x, int n = 1, int nl = 1, int nr = mxn) {
  if (nl==nr) return;
  int mid = (nl + nr) / 2;
  if (idx <= mid) {
    int lidx = tree[n].l;
    tree.push_back({ tree[lidx].l, tree[lidx].r, tree[lidx].val + x });
    tree[n].l = (int)tree.size() - 1;
    update(idx, x, tree[n].l, nl, mid);
  }
  else {
    int ridx = tree[n].r;
    tree.push_back({ tree[ridx].l, tree[ridx].r, tree[ridx].val + x });
    tree[n].r = (int)tree.size() - 1;
    update(idx, x, tree[n].r, mid + 1, nr);
  }
}

int sum(int l, int r, int n = 1, int nl = 1, int nr = mxn) {
  if (r < nl || nr < l) return 0;
  if (l <= nl && nr <= r) return tree[n].val;
  int mid = (nl + nr) / 2;
  return sum(l, r, tree[n].l, nl, mid) + sum(l, r, tree[n].r, mid + 1, nr);
}
vector<int> ys[mxn + 1];

...
```

```
tree = vector<Node> (4);   //2D pst
tree[1].l = 2, tree[1].r = 3; rt[0] = 1;
for (int i = 1; i <= mxn; i++) {
  tree.push_back({ tree[rt[i - 1]].l, tree[rt[i - 1]].r, tree[rt[i - 1]].val });
  rt[i] = (int)tree.size() - 1;
  for (int &y : ys[i]) {
    tree[rt[i]].val += 1;
    update(y, 1, rt[i]);
  }
}
```

Square-root decomposition & MO's

append때는 선 idx 후 처리, reduce때는 선 처리 후 idx

```
sort(all(v), [&](query a, query b) {
  if (a.r / rt == b.r / rt) return a.l < b.l;
  return a.r / rt < b.r / rt;
});
```

Dynamic programming optimization

Convex hull trick

$dp[i] = \min_{0 \le j < i} (A[i]B[j] + C[j]) + D[i]$ where $i \le j \rightarrow B[i] \ge B[j]$

```
using ll = long long;
struct line { ll a, b; };
vector<line> f;  // f = ax + b형태로 관리
double inter(line f1, line f2) { return 1.0 * (f2.b - f1.b) / (f1.a - f2.a); }
void push_line(ll a, ll b) {
  while(f.size() >= 2 && inter (*++f.rbegin(), f.back()) > inter(f.back(), { a,b }))
    f.pop_back();
  f.push_back({ a,b });
}
ll findval(int x) {  //어떤 x좌표에서 함수값의 최솟값 반환
  int s = 0, e = f.size() - 1;
  while(s < e) {
    int m = (s+e)/2;
    if(inter(f[m], f[m+1]) < x) s = m+1;
    else e = m;
  }
  return f[s].a * x + f[s].b;
}
ll l[100], r[100], N, dp[100];
ll solve() {
  push_line(l[0], r[0]);
```

```
  for(int i = 1; i < N; i++) {
    dp[i] = findval(l[i]);
    push_line(r[i], dp[i]);
  }
  return dp[N-1];
}
```

Dynamic programming optimization with deque

```
deque<ii> dq;
dq.push_back(ii(0, 0));
for(int i = 1; i <= N; i++) {
  while(!dq.empty() && i - dq.front().second > D) dq.pop_front();
  dp[i] = sth;
  ans = max(ans, dp[i]);
  while(!dq.empty() && dq.back().first < dp[i]) dq.pop_back();
  dq.push_back(ii(dp[i], i));
}
```

Knuth optimization

1) $D[i][j] = \min_{i<k<j}(D[i][k] + D[k][j]) + C[i][j]$
2) C[a][c]+C[b][d] ≤ C[a][d]+C[b][c], $a \le b \le c \le d$
3) $C[b][c] \le C[a][d], a \le b \le c \le d$

if discrete range, let E[i][j]=D[i+1][j], then
$E[i][j] = \min_{i<k<j}(E[i][k] + E[k][j]) + C[i+1][j]$.

```
for (int len = 1; len <= N; len++) {
  for (int i = 1; i + len <= N; i++) {
    int j = i + len;
    dp[i][j] = INF;

    for (int k = opt[i][j - 1]; k <= opt[i + 1][j]; k++) {
      lint cand = dp[i][k] + dp[k + 1][j] + psum[j] - psum[i - 1];
      if (dp[i][j] > cand) {
        dp[i][j] = cand;
        opt[i][j] = k;
} } } }
```

Problem specified techniques

히스토그램 with stack

```
int st[100010], top = -1;
for (int i = 0; i < N + 1; i++) {
  while (top != -1 && a[st[top]] >= a[i]) {
    int idx = st[top]; top--;
    ans = max(ans, a[idx] * (top == -1 ? i : (i - 1) - st[top]));
  }
  st[++top] = i;
}
```

Dynamic programming

```
// two machine
int N, a[1001], b[1001], dp[2][100010];
for (int i = 1; i <= N; i++) {
  fill_n(dp[1], 100010, INF);
  for (int w = 0; w < 100010; w++) {
    dp[1][w] = min(dp[1][w], dp[0][w] + b[i]);
    if (w >= a[i]) dp[1][w] = min(dp[1][w], dp[0][w - a[i]]);
  }
  swap(dp[0], dp[1]);
}

// 경찰차
int go(int l, int r) {
  int curr = max(l, r);
  if (curr == M + 1) return 0;
  int&ret = dp[l][r];
  if (ret != -1) return ret;
  int d1 = go(curr + 1, r) + dist(arr[curr + 1], arr[l]);
  int d2 = go(l, curr + 1) + dist(arr[curr + 1], arr[r]);
  return ret = min(d1, d2);
}

// 사수아탕
int solve(int i, int j, int rem) {
  if (!rem) return 0;
  int&ret = dp[i][j], l = min(i,j), r = max(i,j);
  if (ret != -1) return dp[i][j];
  dp[i][j] = INF;
  if (r != n) ret = min(ret, solve(l, r + 1, rem - 1) + rem * (x[r + 1] - x[j]));
  if (l != 0) ret = min(ret, solve(r, l - 1, rem - 1) + rem * (x[j] - x[l - 1]));

  return ret;
}

...

for (int i = 0; i <= n; ++i) {
  memset(dp, -1, sizeof(dp));
  res = max(res, i * m - solve(s, s, i));
}
```

```
//코끼리
//max seg for { len, count }
seg.update(0, ii(0, 1));
for (int i = 0; i < N; i++) {
    ii x = seg.query(0, a[i].y - 1); x.first++;
    seg.update(a[i].y, x);
}

//외판원 순회
lint solve(int vis, int cur) {
  if (vis == (1 << n) - 1) return w[cur][0];
  lint&ret = dp[vis][cur];
  if (ret != -1) return ret;
  ret = INF;
  for (int next = 0; next < n; next++) {
    if (vis & (1 << next)) continue;
    ret = min(ret, solve(vis | (1 << next), next) + w[cur][next]);
  }

  return ret;
}

//trie + dp(dp[i] = min(dp[j]) 1
for (int i = N - 1; i >= 0; i--) {
  int idx = 0;
  for (int len = 1; len <= 1000 && i + len - 1 < N; len++) {
    idx = _find(s[i + len - 1], T[idx]);
    if (idx == -1) break;
    if (!terminal[idx]) continue;
    if (dp[i] > dp[i + len] + 1) {
      dp[i] = dp[i + len] + 1;
      sidx[i] = terminal[idx];
      nxt[i] = i + len;
    }
    else if (dp[i] == dp[i + len] + 1 && sidx[i] > terminal[idx]) {
      sidx[i] = terminal[idx];
      nxt[i] = i + len;
    }
  }
}
```

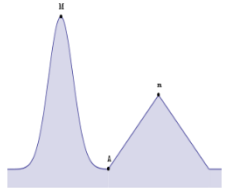
Greedy

```
//마스크
for (int i = 0; i < N; i++)
  cin >> a[i].second >> a[i].first;  // a[i] = {R, L}
for (int i = 0; i < M; i++) {
  lint x, y;
  cin >> x >> y;
  mp[x] += y;
}
sort(a, a + N);
int ans = 0, idx = 0;
for (int i = 0; i < N; i++) {
  lint l = a[i].second, r = a[i].first;
  auto it = mp.lower_bound(l);
  if (it != mp.end() && it->first <= r) {
    ans++;
    if (--(it->second) == 0)
      mp.erase(it);
  }
}

//강의실 배정 (최소 강의실 개수, pq.size())
sort(arr, arr + n);  //{L, R}, 자연빵 sort
for (int i = 0; i < n; i++) {
  if (!pq.empty() && -pq.top() <= arr[i].first)
    pq.pop();
  pq.push(-arr[i].second);
}

//scheduling with deadline
struct P {
  int d, w;  //deadline, 값어치
  bool operator < (P&rhs) {
    if (w != rhs.w) return w > rhs.w;
    return d < rhs.d;
  }
} a[mxn];
sort(a, a + N);
for (int i = 0; i < N; i++) {
  int tmp = find(a[i].d);
  if (!tmp) continue;
  par[tmp] = tmp - 1;
  ans += a[i].w;
}

//rest stop
```



sweeping with segment tree

```
//직사각형의 둘레 합
#include <iostream>
#include <algorithm>
using namespace std;

struct P {
```

```
  int x, ymn, ymx, add;
  bool operator < (P&rhs) {
    if (x != rhs.x) return x < rhs.x;
    return add > rhs.add;
  }
}x[10001], y[10001];

int N, cnt[80008][2], tree[80008][2];

void update(int l, int r, int n, int nl, int nr, int f, int add) {
  if (r < nl || nr < l) return;
  if (l <= nl && nr <= r) cnt[n][f] += add;
  else {
    int mid = (nl + nr) / 2;
    update(l, r, n * 2, nl, mid, f, add);
    update(l, r, n * 2 + 1, mid + 1, nr, f, add);
  }
  if (cnt[n][f]) tree[n][f] = nr - nl + 1;
  else {
    if (nl == nr) tree[n][f] = 0;
    else tree[n][f] = tree[n * 2][f] + tree[n * 2 + 1][f];
  }
}

x[i] = { x1, y1, y2 - 1,  1 }; x[i + N] = { x2, y1, y2 - 1, -1 };
y[i] = { y1, x1, x2 - 1,  1 }; y[i + N] = { y2, x1, x2 - 1, -1 };
sort(x, x + N * 2), sort(y, y + N * 2);
long long ans = 0, prvx = 0, prvy = 0;
for (int i = 0; i < 2 * N; i++) {
  update(x[i].ymn, x[i].ymx, 1, 0, mxrange, 0, x[i].add);
  update(y[i].ymn, y[i].ymx, 1, 0, mxrange, 1, y[i].add);
  ans += abs(tree[1][0] - prvy), ans += abs(tree[1][1] - prvx);
  prvy = tree[1][0], prvx = tree[1][1];
}

//직사각형의 면적 합
int N, tree[4 * 30030], cnt[4 * 30030], lim;
struct P {  //x
  int x, ymn, ymx, add;
  bool operator <(P&rhs) {
    if (x != rhs.x) return x < rhs.x;
    return add < rhs.add;
  }
} a[20001];
void update(int l, int r, int add, int n, int nl, int nr) {
  if (r < nl || nr < l) return;
  if (l <= nl && nr <= r) cnt[n] += add;
  else {
    int mid = (nl + nr) / 2;
    update(l, r, add, n * 2, nl, mid);
    update(l, r, add, n * 2 + 1, mid + 1, nr);
  }
  if (cnt[n]) tree[n] = nr - nl + 1;
  else {
    if (nl == nr) tree[n] = 0;
    else tree[n] = tree[n * 2] + tree[n * 2 + 1];
  }
}
a[i] = { x1, y1, y2 - 1, 1 };
a[i + N] = { x2, y1, y2 - 1, -1 };
for (int i = 0; i < 2 * N; i++) {
  if (i) ans += (a[i].x - a[i - 1].x) * tree[1];
  update(a[i].ymn, a[i].ymx, a[i].add, 1, 0, max_range);
}
```

Graph

```
// 산만한 고양이 (정점 하나 지워서 사이클 존재하지 않도록)
// 해당 정점 번호 합
vector<vector<int> > adj, tree; // bidirectional
vector<int> chk, par, sub_out, sub_in;
void dfs(int cur, int prev = 0) {
  chk[cur] = 1;
  par[cur] = prev;
  for (int nxt : adj[cur]) {
    if (nxt == prev) continue;
    if (chk[nxt] == 2) continue;  // cross, forward
    if (chk[nxt] == 1) {  // back
      sub_out[cur]++;
      sub_in[nxt]++;
    }
    else {  // tree
      int t = sub_in[cur];
      dfs(nxt, cur);
      par[nxt] = sub_in[cur] - t;
      sub_out[cur] += sub_out[nxt];
      sub_in[cur] += sub_in[nxt];
      tree[cur].push_back(nxt);
    }
  }
  chk[cur] = 2;
}
  dfs(1);
  int back = m - (n - 1);
  long long ans = 0;
  for (int i = 1; i <= n; ++i) {
  bool f = true;
  if (sub_out[i] < back) continue;
```

```
  for (int j : tree[i])
    if (sub_in[j] || sub_out[j] - par[j] >= 2) {
      f = false; break;
    }
    if (f) ans += i;
  }
```

GL, HF

BlackWeasel