1. Graph

1.1. dfs iterative

```
/*************************************************************
O(V+E)
깊이 우선 탐색 iterative 버전
*************************************************************/
void dfs_iter(ll x)
{
    stack<ll> cur_s, idx_s;
  ll cur = x, idx = 0;
  do
  {
    ll i, last_idx = adj[cur].size();
    if (!visit[cur])
    {
      visit[cur] = 1;
      printf("%lld ", cur + 1); // doing
    }
    for (i = idx; i < adj[cur].size(); i++)
      if (!visit[adj[cur][i]])
      {
        cur_s.push(cur); idx_s.push(i);
        cur = adj[cur][i]; idx = 0;
        break;
      }
    if (i == last_idx) // there is no next vertex; not break;
    {
       cur = cur_s.top(), cur_s.pop();
       idx = idx_s.top(), idx_s.pop();
    }
  } while (!cur_s.empty());
}
```

1.2. bfs

```
/*************************************************************
O(V+E)
너비 우선 탐색, visit check 타이밍 주의!
*************************************************************/
void bfs(ll x)
{
  queue<ll> q;
  q.push(x);
  visit[x] = 1;
  while (!q.empty())
  {
    ll p = q.front(); q.pop();
    printf("%lld ", p + 1); // doing
    for (ll i = 0; i < adj[p].size(); i++)
      if (!visit[adj[p][i]])
      {
        q.push(adj[p][i]);
        visit[adj[p][i]] = 1;
      }
  }
}
```

1.3. Eulerian Curcuit & trail

```
/*************************************************************
O(VE) // E가 V^2보다 클 수 있음!
음 circuit : reverse order Eulerian Circuit
adj[x][y] : the number of x -> y edges, 즉 인접행렬
trail은 (end, start)간선을 추가하여, circuit으로 변환하여 푼다.
홀수 차수가 2개 존재하면, trail이다.
함정: 간선이 0개인 non-spannig graph도 path가 true다.
*************************************************************/
void getEulerCircuit(ll here, vector<ll>& circuit)
{
    // adj.size() == N
  for(ll there = 0; there < adj.size(); there++)
    while(adj[here][there] > 0)
    {
      adj[here][there]--; // 양쪽 간선을 모두 지운다.(무향)
      adj[there][here]--;
      getEulerCircuit(there, circuit);
    }
  circuit.push_back(here);
}
```

1.4. Dijkstra

```
/**********************************************************
O(ElogV)
fill(dis, dis+N, INF);
adj : pair(vertex, cost)
edge(i, adj[i].first), cost of this edge = adj[i].second;
q.top().first = min-cost, second = vertex;
**********************************************************/
ll dis[N];
void dijk(ll s)
{
  priority_queue<pll, vector<pll>, greater<pll> > q;
  // 주의! q : pair(distance, vertex) <- adj 랑 반대
  dis[s] = 0;
  q.push({dis[s], s});
  while (!q.empty())
  {
    while (!q.empty() && visit[q.top().second]) q.pop();
    if (q.empty()) break;
    ll next = q.top().second; q.pop();
    visit[next] = 1;
    for (ll i = 0; i < adj[next].size(); i++)
      if (dis[adj[next][i].first] > dis[next] + adj[next][i].second)
      {
        dis[adj[next][i].first] = dis[next] + adj[next][i].second;
        q.push({dis[adj[next][i].first], adj[next][i].first});
      }
  }
}
```

1.5. Bellman-Ford

```
/**********************************************************
O(VE)
음수사이클 있으면 0 반환, 없으면 1 반환
fill(dis, dis+N, INF);
**********************************************************/
bool bellman(ll x)
{
  bool cycle = 0;
  dis[x] = 0;
  for (ll i = 0; i < n; i++)
    for (ll j = 0; j < n; j++)
      for (ll k = 0; k < adj[j].size(); k++)
      {
        ll next = adj[j][k].first;
        ll cost = adj[j][k].second;
        if (dis[j] != INF && dis[next] > dis[j] + cost)
        { // INF != 체크하는 이유는,
          아직 방문하지 않은 점으로부터 갱신되는 것을 방지하려고.
          dis[next] = dis[j] + cost;
          if (i == n - 1) cycle = 1;
          // n 번째에 또 갱신된다면, 음의 사이클이 존재한다는 말.
        }
      }
  return !cycle;
}
```

1.6. Floyd-Warshall

```
/**********************************************************
O(V^3)
음수 사이클 있으면 0 반환, 없으면 1 반환
adj 방식 아니고, 바로 dis 배열에 입력받음, dis[N][N]
fill(dis, dis+N, INF);
**********************************************************/
bool floyd()
{
  bool cycle = 0;
  fill(dis, dis+N, INF);
  scanf("%lld %lld", &n, &m);
  for (ll i = 0; i < m; i++) // 입력부
  {
    ll x, y, cost; scanf("%lld %lld %lld", &x, &y, &cost);
    x--; y--; // index 가 1 부터일 경우
    dis[x][y] = min(dis[x][y], cost); // 값이 여러개 들어왔을때, 최소값
  }

  for (ll i = 0; i < n; i++) dis[i][i] = 0;
  for (ll k = 0; k < n; k++)
    for (ll i = 0; i < n; i++)
      for (ll j = 0; j < n; j++)
        dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);

   // 음수사이클이 없다는 전제가 있다면, 아래부분 생략가능, 사이클 체크
  for (ll k = 0; k < n; k++)
```

```
    for (ll i = 0; i < n; i++)
      for (ll j = 0; j < n; j++)
        if (dis[i][j] > dis[i][k] + dis[k][j]) cycle = 1;
  return !cycle;
}
```

2. Tree

2.1. Prim

```
/*********************************************************
O(ElogV)
스패닝 트리를 만들 수 있으면, 모든 cost의 합을 반환한다.
스패닝 트리를 만들 수 없으면, -1 반환한다.
최소 코스트 합만을 반환하는 코드, 직접 간선들을 구하는 것도 가능.
MST의 두 정점 사이의 path는 원래 그래프에서의 minimax path이다.
*********************************************************/
ll prim()
{
  priority_queue<pll, vector<pll>, greater<pll> > q;
  ll count = 0; ll ret = 0;
  q.push(make_pair(0, 0)); // (cost, vertex)
  while (!q.empty())
  {
    ll x = q.top().second; // 이 타이밍에 연결에 사용되는 간선 체크할 수도 있음.
    visit[x] = 1; ret += q.top().first; q.pop(); count++;
    for (ll i = 0; i < adj[x].size(); i++) // adj (vertex, cost)
            q.push(pll(adj[x][i].second, adj[x][i].first));
    while (!q.empty() && visit[q.top().second]) q.pop();
  }
  if (count != n) return -1;
  else return ret;
}
```

2.2. Kruskal

```
/*********************************************************
O(ElogV)
스패닝 트리를 만들 수 있으면, 모든 cost의 합을 반환한다.
스패닝 트리를 만들 수 없으면, -1 반환한다.
최소 코스트 합만을 반환하는 코드, 직접 간선들을 구하는 것도 가능.
MST의 두 정점 사이의 path는 원래 그래프에서의 minimax path이다.
Union-Find 필요
*********************************************************/
ll Kruskal()
{
  ll ret = 0;
  fill(p, p+N, -1);
  vector<pair<ll, pll>> e;
  for(ll i= 0; i < n; i++)
    for(ll j=0; j < adj[i].size(); j++)
      e.push_back({adj[i][j].second, {i, adj[i][j].first}});
  sort(e.begin(), e.end());
  for(ll i=0; i < e.size(); i++)
  {
    ll x = e[i].second.first;
    ll y = e[i].second.second;
    if(find(x) != find(y))
    {
      merge(x, y);
      ret += e[i].first;
    }
  }
  if(-p[find(0)] != n) return -1;
  else return ret;
}
```

2.3. Treap

```
/*********************************************************
O(logN)
삽입, 삭제, 탐색, k번째수, x보다 작은 수의 개수 찾기
*********************************************************/
struct Node
{
  ll key, priority;
  ll data;
  ll size;
```

```
  Node* left;
  Node* right;
  Node(ll _key, ll _data) :data(_data), key(_key),
  size(1), priority(rand()), left(NULL), right(NULL) {}
  void setleft(Node* l)
  {
    size = 1;
    left = l;
    if (left != NULL) size += left->size;
    if (right != NULL) size += right->size;
  }
  void setright(Node* r)
  {
    size = 1;
    right = r;
    if (left != NULL) size += left->size;
    if (right != NULL) size += right->size;
  }
};
pair<Node*, Node*> split(Node* root, ll data)
{
  if (root == NULL) return pair<Node*, Node*>(NULL, NULL);
  if (root->key < data)
  {
    pair<Node*, Node*> tmp = split(root->right, data);
    root->setright(tmp.first);
    return pair<Node*, Node*>(root, tmp.second);
  }
  else
  {
    pair<Node*, Node*> tmp = split(root->left, data);
    root->setleft(tmp.second);
    return pair<Node*, Node*>(tmp.first, root);
  }
}
Node* insert(Node* root, Node* nnode) // root = insert(root, nnode);
{
  if (root == NULL) return nnode;
  if (root->priority < nnode->priority)
  {
    auto child = split(root, nnode->key);
    nnode->setleft(child.first);
    nnode->setright(child.second);
    return nnode;
  }
  else
  {
    if (nnode->key > root->key) root->setright(insert(root->right,
nnode));
    else root->setleft(insert(root->left, nnode));
    return root;
  }
}
Node* merge(Node* l, Node* r)
{
  if (l == NULL) return r;
  if (r == NULL) return l;
  if (l->priority < r->priority)
  {
    r->setleft(merge(l, r->left));
    return r;
  }
  else
  {
    l->setright(merge(l->right, r));
    return l;
  }
}
Node* erase(Node* root, ll data) // root = erase(root, data);
{
  if (root == NULL) return root;
  if (root->key < data) root->setright(erase(root->right, data));
  else if (root->key > data) root->setleft(erase(root->left, data));
  else
  {
    Node* tmp = merge(root->left, root->right);
    delete root;
    return tmp;
  }
  return root;
}
Node* kth(Node* root, ll k) // root = kth(root, k);
{
  ll l = 0;
  if (root->left != NULL) l += root->left->size;
  if (l + 1 == k) return root;
  else if (l + 1 > k) return kth(root->left, k);
  else return kth(root->right, k - l - 1);
```

```
}
ll countlessthan(Node* root,ll data)
{
  if(root == NULL) return 0;
  if(root->data >= data) return countlessthan(root->left,data);
  else
  {
    if(root->left != NULL) return root->left->size + 1 +
countlessthan(root->right,data);
    else return 1 + countlessthan(root->right,data);
  }
}
```

2.4. LCA

```
/*************************************************************
O(logN)
트리의 두 정점 사이의 최소공통조상 구하기
sparse table 버전, level < 2^19
p[i][j] : i 노드의 2^j 번째 조상의 index
d[i], v[i] : dfs tree 에서 i 노드의 depth, i 노드 visit check
*************************************************************/
ll p[N][20];
ll d[N];
bool v[N];
vector<ll> adj[N];
void make_tree(ll x, ll depth)
{
  v[x] = 1;
  d[x] = depth;
  for(ll i=0; i<adj[x].size(); i++)
    if(!v[adj[x][i]])
    {
      p[adj[x][i]][0] = x;
      make_tree(adj[x][i], depth + 1);
    }
}
void init()
{
    // root idx: 1, -1 은 부모 없음
  for(ll i=0; i<20; i++) p[1][i] = -1;
  make_tree(1, 0);
  for(ll k=1; k<20; k++)
    for(ll i=1; i<=n; i++)
      p[i][k] = p[p[i][k-1]][k-1];
}
ll lca(ll x, ll y)
{
  if(d[x] > d[y]) swap(x, y);
  for(ll i=19; i>=0; i--)
    if(d[y] - d[x] >= (1 << i)) y = p[y][i];
  if(x == y) return x;
  for(ll i=19; i>=0; i--)
    if(p[x][i] != p[y][i])
    {
      x = p[x][i];
      y = p[y][i];
    }
  return p[x][0];
}
```

3. Network Flow

3.1. Dinic

```
/*****************************
 * O(V^2 E)
 * 모든 에지의 capacity 가 0 혹은 1 일 때, min(V^(2/3) * E, E^3/2)
 *****************************/
const ll MAXN = 1010;
struct edg{ll pos, cap, rev;};
vector<edg> gph[MAXN];
void clear()
{
    for(int i=0; i<MAXN; i++) gph[i].clear();
}
void add_edge(ll s, ll e, ll v){
    gph[s].push_back({e, v, (ll)gph[e].size()});
    gph[e].push_back({s, 0, (ll)gph[s].size()-1});
}
ll level[MAXN], pnt[MAXN];
bool bfs(ll src, ll sink)
{
    memset(level, 0, sizeof(level));
    memset(pnt, 0, sizeof(pnt));
```

```
    queue<ll> que;
    que.push(src);
    level[src] = 1;
    while(!que.empty())
    {
        ll x = que.front();
        que.pop();
        for(auto &e : gph[x])
        {
            if(e.cap > 0 && !level[e.pos])
            {
                level[e.pos] = level[x] + 1;
                que.push(e.pos);
            }
        }
    }
    return level[sink] > 0;
}
ll dfs(ll x, ll sink, ll f)
{
    if(x == sink) return f;
    for(; pnt[x] < gph[x].size(); pnt[x]++)
    {
        edg e = gph[x][pnt[x]];
        if(e.cap > 0 && level[e.pos] == level[x] +1)
        {
            ll w = dfs(e.pos, sink, min(f, e.cap));
            if(w)
            {
                gph[x][pnt[x]].cap -= w;
                gph[e.pos][e.rev].cap += w;
                return w;
            }
        }
    }
    return 0;
}
ll match(ll src, ll sink)
{
    ll ret = 0, r;
    while(bfs(src, sink))
        while((r = dfs(src, sink, INF))) ret += r;
    return ret;
}
```

3.2. MCMF

```
/*********************
 * O((V+E)*f) ~ O(V*E*f)
 * match(src, sink) := min cost
 * maxflow := 흐를 수 있는 max flow
 *********************/
const int MAXN = 100;
int maxflow = 0;
struct edg{ int pos, cap, rev, cost; };
vector<edg> gph[MAXN];
void clear()
{
    for(int i=0; i<MAXN; i++) gph[i].clear();
}
void add_edge(int s, int e, int x, int c)
{   // start, end, capacity, cost 순
    gph[s].push_back({e, x, (int)gph[e].size(), c});
    gph[e].push_back({s,0, (int)gph[s].size()-1, -c});
}
int dist[MAXN], pa[MAXN], pe[MAXN];
bool inque[MAXN];
bool spfa(int src, int sink)
{
    fill(dist, dist+MAXN, INF);
    fill(inque, inque+MAXN, 0);
    queue<int> que;
    dist[src] = 0;
    inque[src] = 1;
    que.push(src);
    bool ok = 0;
    while(!que.empty())
    {
        int x = que.front();
        que.pop();
        if(x == sink) ok = 1;
        inque[x] = 0;
        for(int i=0; i<gph[x].size(); i++)
        {
            edg e = gph[x][i];
            if(e.cap > 0 && dist[e.pos] > dist[x] + e.cost)
            {
                dist[e.pos] = dist[x] + e.cost;
                pa[e.pos] = x;
```

```
                pe[e.pos] = i;
                if(!inque[e.pos])
                {
                    inque[e.pos] = 1;
                    que.push(e.pos);
                }
            }
        }
    }
    return ok;
}
int match(int src, int sink)
{
    int ret = 0;
    while(spfa(src, sink))
    {
        int cap = 1e9;
        for(int pos = sink; pos != src; pos = pa[pos])
            cap = min(cap, gph[pa[pos]][pe[pos]].cap);
        ret += dist[sink] * cap;
        for(int pos = sink; pos != src; pos = pa[pos])
        {
            int rev = gph[pa[pos]][pe[pos]].rev;
            gph[pa[pos]][pe[pos]].cap -= cap;
            gph[pos][rev].cap += cap;
        }
        maxflow += cap;
    }
    return ret;
}
```

3.3. Bipartite Matching

```
/******************************
 * O(E√V)
 * sol() : return 매칭 수
 ******************************/
// A[i], B[i]: 그룹의 i번 정점과 매칭된 상대편 그룹 정점 번호
int N, A[MAX], B[MAX], dist[MAX]; // dist[i]: (A그룹의) i번 정점의 레벨
bool used[MAX]; // used: (A그룹의) 이 정점이 매칭에 속해 있는가?
vector<int> adj[MAX];

void bfs()
{
    queue<int> q;
    for(int i=0; i<N; i++)
    {   // 매칭에 안 속한 A그룹의 정점만 레벨 0인 채로 시작
        if(!used[i])
        {
            dist[i] = 0;
            q.push(i);
        }
        else dist[i] = INF;
    }
    while(!q.empty())
    {   // BFS를 통해 A그룹 정점에 0, 1, 2, 3, ... 의 레벨을 매김
        int i = q.front();
        q.pop();
        for(int x : adj[i])
        {
            if(B[x] != -1 && dist[B[x]] == INF)
            {
                dist[B[x]] = dist[i] + 1;
                q.push(B[x]);
            }
        }
    }
}
bool dfs(int a)
{
    for(int b: adj[a])
    {   // 이분 매칭 코드와 상당히 유사하나,
        dist 배열에 대한 조건이 추가로 붙음
        if(B[b] == -1 || ((dist[B[b]] == dist[a]+1) && dfs(B[b])))
        {   // used 배열 값도 true가 됨
            used[a] = true;
            A[a] = b;
            B[b] = a;
            return true;
        }
    }
    return false;
}
ll sol()
{
    ll match = 0;
    fill(A, A+MAX, -1);
```

```
    fill(B, B+MAX, -1);
    while(1)
    {
        bfs();
        ll flow = 0;
        for(int i=0; i<N; i++)
            if(!used[i] && dfs(i)) flow++;
        if(flow == 0) break;
        match += flow;
    }
    return match;
}
```

4. Query Tree

4.1. Segment Tree + Lazy propagation

```
/*************************************************************
O(logN) range update, range query
현재 구간합 버전
1. update(1, 0, n-1, l, r, v); // [l, r]에 v만큼 update
2. query(1, 0, n-1, l, r); // [l, r] 구간 쿼리
*************************************************************/
ll a[4*N + 1] = { 0 };
ll lazy[4*N + 1] = { 0 };
void update(ll n, ll nl, ll nr, ll l, ll r, ll v)
{
    if(lazy[n])
    {
        a[n] += (nr-nl+1) * lazy[n];
        if(nl!=nr) lazy[2*n] += lazy[n];
        if(nl!=nr) lazy[2*n+1] += lazy[n];
        lazy[n] = 0;
    }
    if(r < nl || nr < l) return;
    else if(l <= nl && nr <= r)
    {
        a[n] += (nr-nl+1) * v;
        if(nl!=nr) lazy[2*n] += v;
        if(nl!=nr) lazy[2*n+1] += v;
    }
    else
    {
        ll mid = (nl + nr) / 2;
        update(2*n, nl, mid, l, r, v);
        update(2*n+1, mid + 1, nr, l, r, v);
        a[n] = a[2*n] + a[2*n+1];
    }
}
ll query(ll n, ll nl, ll nr, ll l, ll r)
{
    if(lazy[n])
    {
        a[n] += (nr-nl+1) * lazy[n];
        if(nl!=nr) lazy[2*n] += lazy[n];
        if(nl!=nr) lazy[2*n+1] += lazy[n];
        lazy[n] = 0;
    }
    if(r < nl || nr < l) return 0;
    else if(l <= nl && nr <= r) return a[n];
    else
    {
        ll mid = (nl + nr) / 2;
        return query(2*n, nl, mid, l, r) + query(2*n+1, mid + 1, nr, l,
r);
    }
}
```

4.2. Fenwick Tree [Range update]

```
/*************************************************************
O(logN) range update, range query
1. update(a, b, v); // (a, b)에 v만큼 더하기
2. query(ll a, ll b) // (a, b) 구간 합
*************************************************************/
void update(ll tree*, ll p, ll v)
{
  for (; p <= N; p += p&(-p))
      tree[p] += v;
}
void update(ll a, ll b, ll v)
{
  // Add v to A[a...b]
  update(B1, a, v);
  update(B1, b + 1, -v);
```

```
  update(B2, a, v * (a-1));
  update(B2, b + 1, -v * b);
}
ll query(ll tree*, ll b)
{
  ll sum = 0;
  for(; b > 0; b -= b&(-b))
    sum += tree[b];
  return sum;
}
ll query(ll b)
{
  // Return sum A[1...b]
  return query(B1, b) * b - query(B2, b);
}
ll query(ll a, ll b)
{
  // Return sum A[a...b]
  return query(b) - query(a-1);
}
```

4.3. 2D Fenwick

```
/*************************************************************
O(logN) point update, range query
현재 2D 구간합 버전
1. update(x, y, v); // (x, y)에 v만큼 더하기
2. sol(ll x, ll y, ll xx, ll yy)
// (x, y) ~ (xx, yy) 직사각형 구간 합
*************************************************************/
ll a[N][N] = { 0 };
ll tree[N][N] = { 0 };
void update(ll x, ll y, ll v)
{
    for(ll xx = x; xx <= n; xx += (xx & -xx))
        for(ll yy = y; yy <= n; yy += (yy & -yy))
            tree[xx][yy] += v;
}
ll query(ll x, ll y)
{
    ll ret = 0;
    for(ll xx = x; xx; xx -= (xx & -xx))
        for(ll yy = y; yy; yy -= (yy & -yy))
            ret += tree[xx][yy];
    return ret;
}
ll sol(ll x, ll y, ll xx, ll yy)
{
    x--; y--;
    return query(xx, yy) - query(x, yy) - query(xx, y) + query(x, y);
}
```

5. Geometric

5.1. vector2 class

```
/*************************************************************
vector2 클래스는 1개의 점을 나타낸다.
long long 버전
*************************************************************/
struct vector2
{
    ll x, y;
    explicit vector2(ll x_ = 0, ll y_ = 0) : x(x_), y(y_) {}
    bool operator==(const vector2& rhs) const
    {
        return x == rhs.x && y == rhs.y;
    }
    bool operator<(const vector2& rhs) const
    {
        return x != rhs.x ? x < rhs.x : y < rhs.y;
    }
    vector2 operator+(const vector2& rhs) const
    {
        return vector2(x + rhs.x, y + rhs.y);
    }
    vector2 operator-(const vector2& rhs) const
    {
        return vector2(x - rhs.x, y - rhs.y);
    }
    vector2 operator*(ll rhs) const
    {
        return vector2(x * rhs, y * rhs);
    }
    ll dot(const vector2& rhs) const
```

```
    {
        return x * rhs.x + y * rhs.y;
    }
    ll cross(const vector2& rhs) const
    {
        return x * rhs.y - rhs.x * y;
    }
    double norm() const
    {
        return hypot(x, y);
    }
};
ll ccw(vector2 a, vector2 b)
{
    return a.cross(b);
}
ll ccw(vector2 p, vector2 a, vector2 b)
{
    return ccw(a-p, b-p);
}
bool isParallel(vector2 a, vector2 b, vector2 c, vector2 d)
{
    return (a-b).cross(c-d) == 0;
}
```

5.2. 선분의 교점개수

```
/***********************************************************
두 선분 A, B 에 대하여, (A: a -> b, B : c -> d)
1. 교점이 있으면(한 점 또는 무한개의 점), true
2. 교점이 없으면, false 를 반환한다.
ab, cd 는 두 선분 중 하나의 선분을 기준으로, 다른 선분의 두 점들에 대한 ccw
값을 곱한 값이다.
따라서 ab, cd 각 값이 의미하는 것은
해당 값이 음수이면 기준 선분에 대해 다른 선분의 두 점이 서로 반대방향에 있는
것이고 (즉, 기준 선분이 가운데에 있음)
해당 값이 양수이면 서로 같은방향에 있는 것이다.
해당 값이 0 이라면, 한 선분이 다른 선분의 일직선 상 위에 존재하거나, 다른
선분의 한점이 기준 선분 위에 존재하는 것이다. (ㅗ 모양)
if 조건문에 걸리는 케이스는, (ab, cd 모두 0)
1. 한 선분이 다른 선분의 일직선 상 위에 존재하는 경우 (만나지 않거나, 한
점에서 만나거나, 무한개의 점에서 만나거나)
2. 두 선분의 끝점에서 서로  만나는 경우 (V 모양, 단순히 한 점에서 만나는 것이
아님! 이유는 서로가 ㅗ, ㅗ이려면 V 여야 한다.)
따라서 위 경우에서 교점이 없는 경우는 1 번에서 만나지 않는 경우이다.
이것은 a, b 와 c, d 가 pair 정렬기준으로 정렬되어 있을 때, (x 기준으로 우선
정렬, x 값이 같다면 y 기준으로 정렬)
b < c || d < a 인 경우이다.
if 조건문에서 걸리지 않는 케이스에서 교점이 생기려면,
두 선분이 서로 크로스되거나 (X 모양), 한 선분의 끝 점이 다른 선분 위에 있어야
한다. (ㅗ 모양)
1. 크로스 되는 경우에는, ab < 0 && cd < 0 이고,
2. 한 선분의 끝 점이 다른 선분 위에 있는 경우에는, ab == 0 && cd < 0  OR  ab
< 0 && cd == 0 이다.
이 밖의 경우에는 교점이 존재하지 않는다.
***********************************************************/
bool segmentIntersect(vector2 a, vector2 b, vector2 c, vector2 d)
{
    ll ab = ccw(a, b, c) * ccw(a, b, d);
    ll cd = ccw(c, d, a) * ccw(c, d, b);
    if(ab == 0 && cd == 0)
    {
        if(b < a) swap(a, b);
        if(d < c) swap(c, d);
        return !(b < c || d < a);
    }
    return ab <= 0 && cd <= 0;
}
```

5.3. 다각형의 넓이

```
ll area(const vector<vector2>& p)
{
    ll ret = 0;
    for(ll i = 0; i < p.size(); i++)
    {
        ll j = (i+1) % p.size();
        ret += p[i].x * p[j].y - p[j].x * p[i].y;
    }
    return abs(ret) / 2;
}
```

5.4. 다각형의 내외부 판별

```
/**********************************************************
vector2 클래스 double 버전 필요
**********************************************************/
bool isInside(vector2 q, const vector<vector2>& p)
{
    ll crosses = 0;
    for(ll i = 0; i < p.size(); i++)
    {
        ll j = (i+1) % p.size();
        // 1.선분을 반직선이 가로지르거나 (-/- or -\-)
        // 2.선분의 한 점은 반직선 위에, 나머지 한 점은 반직선보다 위에 있는
경우 (_/_ or _\_)
        if((p[i].y > q.y) != (p[j].y > q.y))
        {
            // 선분과 반직선의 교차점 atX
            double atX = (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y -
p[i].y) + p[i].x;
            // 오른쪽에 있어야 카운트
            if(q.x < atX) crosses++;
        }
    }
    if(crosses % 2 == 1) return true;
    else return false;
}
```

5.5. Convex Hull

```
/**********************************************************
정렬 O(NlogN) + 점선택 O(N) = O(NlogN)
P1 ~ Pn 까지의 점들이 있을 때 볼록 다각형을 이루는 점들을 찾는 알고리즘
1. P1 을 기준으로 P2 ~ Pn 을 (P1 -> Pi)의 기울기의 오름차순으로 정렬한다.
(CCW 를 비교연산으로 이용)
2. P1 과 기울기가 제일 낮은 점 P_first 를 선택하여 스택(벡터)에 넣는다.
3. 그 다음 점 Px 부터 이전의 2 개의 점 (P_top-1 -> P_top)과 (P_top -> Px)가
CCW 일 경우에 그냥 push 하고,
4. 아닐 경우(CW)에는 CCW 가 될 때까지 P_top 을 pop 하고, Px 를 push 한다.
5. 마지막 점까지 3-4 를 반복한다.
6. 스택(벡터)에 남아있는 점들이 볼록 다각형을 이루는 점들이다.
**********************************************************/
vector<vector2> p, ret;
vector2 f;
bool cmp(vector2& x, vector2& y)
{
    double flag = ccw(f, x, y);
    if(flag > 0) return true;
    else if(flag == 0)
    {
        // 일직선 상에 있을 때, 더 먼 점이 더 큰 점으로 결정
        if((x-f).norm() < (y-f).norm()) return true;
        else return false;
    }
    else return false;
}
void graham(vector<vector2>& p, vector<vector2>& ret)
{
    sort(p.begin(), p.end());
    f = p[0]; ret.push_back(f);
    sort(p.begin() + 1, p.end(), cmp);
    for(ll i=1; i<p.size(); i++)
    {
        // ccw 가 0 일 경우에는 일직선 상위의 점이고 뒤의 점이 더 멀리 있으므로
무조건 pop 한다.
        while(ret.size() >= 2 && ccw(ret[ret.size()-1] - ret[ret.size()-
2], p[i] - ret[ret.size()-1]) <= 0) ret.pop_back();
        ret.push_back(p[i]);
    }
}
```

5.6. Rotating Calipers

```
/**********************************************************
볼록 껍질 O(NlogN) + 로테이팅 캘리퍼스 O(N) = O(NlogN)
a: x 기준 가장 왼쪽 점
b: x 기준 가장 오른쪽 점
1. a, b 를 지나는 서로 반대 방향의 두 평행선을 회전시키며 a, b 사이의 거리를
잰다.
2. 회전각이 더 작은쪽으로 회전시킨다. 회전에 따라서 a 또는 b 가 가리키는 점이
바뀐다.
3. 반 바퀴를 돌아서 a 와 b 가 가리키는 점이 처음과 반대이면 끝!
**********************************************************/
double diameter(const vector<vector2>& p)
```

```
{
    ll n = p.size();
    // 우선 가장 왼쪽에 있는 점과 오른쪽에 있는 점을 찾는다.
    ll left = min_element(p.begin(), p.end()) - p.begin();
    ll right = max_element(p.begin(), p.end()) - p.begin();
    // p[left]와 p[right]에 각각 수직선을 붙인다.
       두 수직선은 서로 정반대 방향을 가리키므로,
    // a 의 방향만을 표현하면 된다.
    vector2 calipersA(0, 1);
    double ret = (p[right] - p[left]).norm();
    // toNext[i]: p[i]에서 다음 점까지의 방향을 나타내는 단위 벡터
    vector<vector2> toNext(n);
    for(ll i = 0; i < n; i++)
       toNext[i] = (p[(i+1) % n] - p[i]).normalize();
    // a 와 b 는 각각 두 선분이 어디에 붙은 채로 회전하고 있는지를 나타낸다.
    ll a = left, b = right;
    // 반 바퀴 돌아서 두 선분이 서로 위치를 바꿀 때까지 계속한다.
    while(a != right || b != left)
    {
        // a 에서 다음 점까지의 각도와 b 에서 다음 점까지의 각도 중
           어느 쪽이 작은지 확인한다.
        double cosThetaA = calipersA.dot(toNext[a]);
        double cosThetaB = -calipersA.dot(toNext[b]);
        if(cosThetaA > cosThetaB) calipersA = toNext[a], a = (a+1) % n;
        else calipersA = toNext[b] * -1, b = (b+1) % n;
        ret = max(ret, (p[a] - p[b]).norm());
    }
    return ret;
}
```

6. Math

6.1. 에라스토테네스의 체

```
/*********************************************************
O(NloglogN)
fill(isprime, isprime+N, 1);
*********************************************************/
bool isprime[N];
vector<ll> prime;
void make_prime(ll n) // O(NloglogN)
{
    isprime[0] = isprime[1] = 0;
    for(ll i = 2; i <= n; i++)
    {
        if(isprime[i])
        {
            for(ll j = i*i; j < n; j += i) isprime[j] = 0;
            prime.push_back(i);
        }
    }
}
```

6.2. 소인수분해

```
/*********************************************************
O(NloglogN)
isprime[i]; i 의 소수여부
fprime[i]: i 를 합성수라고 판별한 첫번째 소수
*********************************************************/
void prime_proc(int n) {
    fill(fsprime, isprime+N, 0);
    fill(isprime, isprime+N, 1);
    isprime[0] = isprime[1] = 0;
    for(ll i = 2; i <= n; i++)
    {
        if(isprime[i])
        {
            fprime[i] = i; // i 는 소수
            for(ll j = i*i; j < n; j += i)
            {
                isprime[j] = 0;
                if(fprime[j] == 0) fprime[j] = i;
            }
        }
    }
}
while(n>1) { // n 을 소인수분해
    cout << fprime[n] << '\n'
    n/=fprime[n];
}
```

6.3. GCD

```
/*********************************************************
O(log(x+y))
*********************************************************/
ll gcd(ll x, ll y)
{
    while(y != 0) {x = x % y; swap(x, y);}
    return x;
}
```

6.4. XGCD + Modular Inverse

```
/*********************************************************
O(log(a+b))
a 의 mod_inv b,즉 a * b == 1 (mod n) 을 구하려면
xgcd(n, a) 에서 t0(음수 주의)
ax + by == gcd(a, b) -> s0 == x, t0 == y
*********************************************************/
tuple<ll, ll, ll> xgcd(ll a, ll b)
{
    ll q, s0, t0, s1, t1;
    r0 = a, r1 = b;
    s0 = 1, s1 = 0;
    t0 = 0, t1 = 1;
    while(r1 != 0)
    {
        q = r0 / r1;
        r0 = r0 % r1;
        swap(r0, r1);
        s0 = s0 - q * s1;
        swap(s0, s1);
        t0 = t0 - q * t1;
        swap(t0, t1);
    }
    return {r0, s0, t0};
    // r0: a, b 의 최대공약수
    // s0, t0: 베주 계수 (서로소)
}
ll mod_inv(ll a, ll n)
{
  ll b = get<2>(xgcd(n, a));
  if(b < 0) b += n;
  return b;
}
```

6.5. 중국인의 나머지 정리

```
/*********************************************************
x == a1 (mod m1)
x == a2 (mod m2) 인, x 를 구하여라.
식이 여러 개인 경우 2 개의 식을
x == crt(a1, m1, a2, m2) (mod m1*m2) 로 reduce
*********************************************************/
ll crt(ll a1, ll m1, ll a2, ll m2)
{
  return (a1 * m2 * mul_inv(m2, m1) + a2 * m1 * mul_inv(m1, m2)) %
(m1*m2);
}
```

6.6. 오일러 피함수

```
/*********************************************************
O(NloglogN)
자연수 n 보다 작고 n 과 서로소인 자연수의 개수
prime_proc( ... ) 으로 fprime 을 얻는다.
*********************************************************/
int phi(int n, vector<int> const& fprime)
{
  if(n == 1) return 1;
  int ret = 1, r = 1, lastp;
  while(n > 1)
  {
    lastp = fprime[n];
    n /= lastp;
    if(fprime[n] != lastp)
    {
      ret *= (r * lastp - r);
      r = 1;
    }
    else r *= lastp;
  }
  return ret;
}
```

6.7. FFT

```
/************************************************************
O(NlogN) 다항식 v와 w의 곱 ret을 O(NlogN)에 계산한다.
HOW TO USE: vector<ll> ret = fft::multiply(v, w);
v[i], w[i] : x^i 번째 항의 계수
항의 계수가 0일 경우에도 0을 넣어줘야 한다.
************************************************************/
#include <complex>
#include <cmath>
namespace fft {
  typedef complex<double> base;
  void fft(vector<base> &a, bool inv)
  {
    int n = a.size(), j = 0;
    vector<base> roots(n/2);
    for(int i=1; i<n; i++)
    {
      int bit = (n >> 1);
      while(j >= bit)
      {
        j -= bit;
        bit >>= 1;
      }
      j += bit;
      if(i < j) swap(a[i], a[j]);
    }
    double ang = 2 * acos(-1) / n * (inv ? -1 : 1);
    for(int i=0; i<n/2; i++)
    {
      roots[i] = base(cos(ang * i), sin(ang * i));
    }
    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++)
        {
          base u = a[j+k], v = a[j+k+i/2] * roots[step * k];
          a[j+k] = u+v;
          a[j+k+i/2] = u-v;
        }
      }
    }
    if(inv) for(int i=0; i<n; i++) a[i] /= n;
  }
  vector<ll> multiply(vector<ll> &v, vector<ll> &w)
  {
    vector<base> fv(v.begin(), v.end()), fw(w.begin(), w.end());
    int n = 2; while(n < v.size() + w.size()) n <<= 1;
    fv.resize(n); fw.resize(n);
    fft(fv, 0); fft(fw, 0);
    for(int i=0; i<n; i++) fv[i] *= fw[i];
    fft(fv, 1);
      vector<ll> ret(n);
    for(int i=0; i<n; i++) ret[i] = (ll)round(fv[i].real());
      return ret;
  }
}
```

7. ETC

7.1. Union-Find

```
/************************************************************
O(4)
memset(p, -1, sizeof(p));
p[x]가 음수일 경우, 집합의 부모이며 절대값 == 집합의 크기
************************************************************/
int p[N];
int find(int x)
{
  if(p[x] < 0) return x;
  p[x] = find(p[x]);
  return p[x];
}
void merge(int x, int y)
{
  x = find(x); y = find(y);
  if(x == y) return;
  p[x] += p[y];
  p[y] = x;
}
```

7.2. Parallel Binary Search

```
/*********************************************************
O(QlogN) 보다 빠르게 할 수 있다.
*********************************************************/
int n; // 쿼리의 개수
int lo[300000], hi[300000];
vector<int> qa[300000];
int main()
{
  /* 테스트케이스 입력부 */
  // scanf("%d", &n);
  // ...
  for(i=0; i<n; ++i) {
    lo[i] = BSEARCH_MIN, hi[i] = BSEARCH_MAX;
  }
  while(true) {
    bool flag = true;
    for(i=0; i<n; ++i) {
      if(lo[i] == hi[i]) continue;
      qa[(lo[i]+hi[i])>>1].push_back(i); // <-- 이 `i`를 주목 (*)
      flag = false;
    }
    if(flag) break;
    for(i = BSEARCH_MIN; i <= BSEARCH_MAX; ++i) {
      // `i`는 mid 값
      for(int idx: qa[i]) {
        // 여기서 `idx`가 (*)의 `i`임.
        // query id `idx` 에 대해 연산 후 lo[idx] or hi[idx] 결정
        if(/* condition */) lo[idx] = i+1;
        else hi[idx] = i;
      }
      // 새로운 qa 배열을 만드는 최적화된 코드
      vector<int> empty;
      swap(empty, qa[i]);
    }
  }
  // 출력
  return 0;
```

7.3. Fast Cin/Cout

```
/*********************************************************
주의!! printf, scanf 와 혼용해서 쓰면 안된다.
*********************************************************/
ios_base::sync_with_stdio(false);
```