Team Note.

Complied 2019-10-04

TheQuickBrownFox

Chung-Ang University

1 Typedef

```
typedef long long ll;
typedef vector<ll> Vll; // vector
typedef vector< Vll > Matrix; // vector
typedef double D;
typedef queue<ll> Q; //queue
typedef deque<ll> DQ; // deque
typedef priority_queue<ll, vector<ll>, greater<ll> > min_heap;
typedef priority_queue<ll> max_heap; //(Heap : functional, vector)
```

2 Numerical Algorithm

2.1 GCD / LCM / MODULAR

```
ll gcd(ll a, ll b){ while(b) { int t = a%b; a=b; b=t;} return a; }

ll lcm(ll a, ll b){ return a / gcd(a,b)*b; }

ll mod(ll num, ll div){ return ((num%div) + div) % div; }

ll combination(ll n, ll k){n==k:1,(k<0¦¦k>n):0,k<1:1}, c(n,k)=c(n-1,k-1)+c(n-1,k)
//이거 수도코드임
```

2.2 Matrix

2.2.1 Matrix Multiplication with Operator Overloading

```
//Matrix a(n by m), Matrix b(m by l), result Matrix c(n by l)
typedef vector< Vll > matrix;
const ll mod =  31991;
matrix operator* (const matrix &a, const matrix &b) {
    int n = a.size(), m = a[0].size(), l = b[0].size();
    matrix c(n, vector<long long>(l));
    for (int i = 0; i<n; i++) {
        for (int j = 0; j<l; j++) {
            for (int k = 0; k<m; k++) {
                c[i][j] += a[i][k] * b[k][j];
            }
            c[i][j] %= mod; //If not need = erase;
        }
    }
    return c;
}
```

2.2.1 Fastpow of Matrix (M^n in log n)

```
//a=source matrix, t=몇승, ans=result matrix(Unit matrix, I)
while (t > 0) {
    if (t % 2 == 1) {
        ans = ans * a;
    }
    a = a * a;
    t /= 2;
}
```

2.3 Fast Exponential (a^n is calculated in $O(\lg n)$)

```
const ll MOD = 1000000;
ll fast_power(ll base, ll power) {
    ll result = 1;
    while(power > 0) {

        if(power & 1) {
            result = (result*base) % MOD;
        }
        base = (base * base) % MOD;
        power >>=1 ;
    }
    return result;
}
```

2.4 C++ Next Permutation (Get all permutation of vector<int>)

```
int main () {
  int myints[] = {1,2,3};
  sort (myints,myints+3);


  do {
    std::cout << myints[0] << ' ' << myints[1] << ' ' << myints[2];
  } while ( std::next_permutation(myints,myints+3) );


  return 0;
}
```

2.5 Count Prime Number in range

```
// credit : https://github.com/stjepang/snippets/blob/master/count_primes.cpp
// Primes up to 10^12 can be counted in ~1 second.
const int MAXN = 1000005; // MAXN is the maximum value of sqrt(N) + 2
bool prime[MAXN];
int prec[MAXN];
vector<int> P;
void init() {
    prime[2] = true;
    for (int i = 3; i < MAXN; i += 2) prime[i] = true;
    for (int i = 3; i*i < MAXN; i += 2){
        if (prime[i]){
            for (int j = i*i; j < MAXN; j += i+i) prime[j] = false;
        }
    }
    for(int i=1; i<MAXN; i++){
    if (prime[i]) P.push_back(i);
        prec[i] = prec[i-1] + prime[i];
    }
}

ll rec(ll N, int K) {
    if (N <= 1 || K < 0) return 0;
    if (N <= P[K]) return N-1;
    if (N < MAXN && 1ll * P[K]*P[K] > N) return N-1 - prec[N] + prec[P[K]];
    const int LIM = 250;
    static int memo[LIM*LIM][LIM];
    bool ok = N < LIM*LIM;
    if (ok && memo[N][K]) return memo[N][K];
    ll ret = N/P[K] - rec(N/P[K], K-1) + rec(N, K-1);
    if (ok) memo[N][K] = ret;
    return ret;
}

ll count_primes(ll N) { //less than or equal to
    if (N < MAXN) return prec[N];
    int K = prec[(int)sqrt(N) + 1];
    return N-1 - rec(N, K) + prec[P[K]];
}
```

3 Graph Theory

3.1 DFS / BFS with Graph Class (Adjacency List)

```
// credit : https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
using namespace std;
// Have to include <list>, <vector>, <algorithm>
// This class represents a directed graph using
// adjacency list representation
class Graph
{
    int V;    // No. of vertices

    // Pointer to an array containing adjacency
    // lists
    list<int> *adj;
    void DFSUtil(int v, bool visited[]);
public:
    Graph(int V);  // Constructor

    // function to add an edge to graph
    void addEdge(int v, int w);

    // prints BFS traversal from a given source s
    void BFS(int s);
    void DFS(int v);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}
void Graph::DFSUtil(int v, bool visited[])
{
    // Mark the current node as visited and
    // print it
    visited[v] = true;
    cout << v << " ";
```

```
    // Recur for all the vertices adjacent
    // to this vertex
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFSUtil(*i, visited);
}

// DFS traversal of the vertices reachable from v.
// It uses recursive DFSUtil()
void Graph::DFS(int v)
{
    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    // Call the recursive helper function
    // to print DFS traversal
    DFSUtil(v, visited);
}
void Graph::BFS(int s)
{
    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for(int i = 0; i < V; i++)
        visited[i] = false;

    // Create a queue for BFS
    list<int> queue;

    // Mark the current node as visited and enqueue it
    visited[s] = true;
    queue.push_back(s);

    // 'i' will be used to get all adjacent
    // vertices of a vertex
    list<int>::iterator i;

    while(!queue.empty())
    {
        // Dequeue a vertex from queue and print it
        s = queue.front();
        cout << s << " ";
        queue.pop_front();

        // Get all adjacent vertices of the dequeued
        // vertex s. If a adjacent has not been visited,
        // then mark it visited and enqueue it
        for (i = adj[s].begin(); i != adj[s].end(); ++i)
        {
            if (!visited[*i])
            {
                visited[*i] = true;
                queue.push_back(*i);
            }
        }
    }
}
```

3.2 Find Existence of Cycle in DAG by DFS

```
// This function is a variation of DFSUytil() in https://www.geeksforgeeks.org/archives/18212
// Have to of declaration of Function in public: and private:
bool Graph::isCyclicUtil(int v, bool visited[], bool *recStack){
    if(visited[v] == false)
    {
        // Mark the current node as visited and part of recursion stack
        visited[v] = true;
        recStack[v] = true;

        // Recur for all the vertices adjacent to this vertex
        list<int>::iterator i;
        for(i = adj[v].begin(); i != adj[v].end(); ++i)
        {
            if ( !visited[*i] && isCyclicUtil(*i, visited, recStack) )
                return true;
            else if (recStack[*i])
                return true;
        }

    }
```

```
    recStack[v] = false;  // remove the vertex from recursion stack
    return false;
}

// Returns true if the graph contains a cycle, else false.
// This function is a variation of DFS() in https://www.geeksforgeeks.org/archives/18212
bool Graph::isCyclic()
{
    // Mark all the vertices as not visited and not part of recursion
    // stack
    bool *visited = new bool[V];
    bool *recStack = new bool[V];
    for(int i = 0; i < V; i++)
    {
        visited[i] = false;
        recStack[i] = false;
    }

    // Call the recursive helper function to detect cycle in different
    // DFS trees
    for(int i = 0; i < V; i++)
        if (isCyclicUtil(i, visited, recStack))
            return true;

    return false;
}
```

3.3 Strong Connected Components (Tarjan) $O(V+E)$ with graph class

```
// A recursive function to print DFS starting from v
// Have to include <stack> <list>
// Have to add public:printSCCs,getTranspose private:fillOrder,DFSUtil
void Graph::DFSUtil(int v, bool visited[])
{
    // Mark the current node as visited and print it
    visited[v] = true;
    cout << v << " ";

    // Recur for all the vertices adjacent to this vertex
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFSUtil(*i, visited);
}

Graph Graph::getTranspose()
{
    Graph g(V);
    for (int v = 0; v < V; v++)
    {
        // Recur for all the vertices adjacent to this vertex
        list<int>::iterator i;
        for(i = adj[v].begin(); i != adj[v].end(); ++i)
        {
            g.adj[*i].push_back(v);
        }
    }
    return g;
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}

void Graph::fillOrder(int v, bool visited[], stack<int> &Stack)
{
    // Mark the current node as visited and print it
    visited[v] = true;

    // Recur for all the vertices adjacent to this vertex
    list<int>::iterator i;
    for(i = adj[v].begin(); i != adj[v].end(); ++i)
        if(!visited[*i])
            fillOrder(*i, visited, Stack);

    // All vertices reachable from v are processed by now, push v
    Stack.push(v);
}

// The main function that finds and prints all strongly connected
// components
void Graph::printSCCs()
{
    stack<int> Stack;
```

```
    // Mark all the vertices as not visited (For first DFS)
    bool *visited = new bool[V];
    for(int i = 0; i < V; i++)
        visited[i] = false;

    // Fill vertices in stack according to their finishing times
    for(int i = 0; i < V; i++)
        if(visited[i] == false)
            fillOrder(i, visited, Stack);

    // Create a reversed graph
    Graph gr = getTranspose();

    // Mark all the vertices as not visited (For second DFS)
    for(int i = 0; i < V; i++)
        visited[i] = false;

    // Now process all vertices in order defined by Stack
    while (Stack.empty() == false)
    {
        // Pop a vertex from stack
        int v = Stack.top();
        Stack.pop();

        // Print Strongly connected component of the popped vertex
        if (visited[v] == false)
        {
            gr.DFSUtil(v, visited);
            cout << endl;
        }
    }
}
// Driver program to test above functions
int main()
{
    // Create a graph given in the above diagram
    Graph g(5);
    g.addEdge(1, 0);
    g.addEdge(0, 2);
    g.addEdge(2, 1);
    g.addEdge(0, 3);
    g.addEdge(3, 4);

    cout << "Following are strongly connected components in "
            "given graph \n";
    g.printSCCs(); //

    return 0;
}
```

3.4 Floyd $O(n^3)$

```
int map[  ][  ] = {0,};
for (int k = 0; k < N; k++)
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            if (map[i][k] && map[k][j]) map[i][j] = 1;
```

3.5 Fast Djikstra (NOT IN GRAPH CLASS)

```
//정점 index는 0에서 n-1까지!
//vector< vector<edge> >& graph는 idx -> to로 향하는 모든 edge
//distance will overflow an int. If you have time, change it ll
//INT_MAX는 <climits>헤더에
struct edge { int to, length; };

int dijkstra(const vector< vector<edge> > &graph, int source, int target) {
    vector<int> min_distance( graph.size(), INT_MAX );
    min_distance[ source ] = 0;
    set< pair<int,int> > active_vertices;
    active_vertices.insert( {0,source} );

    while (!active_vertices.empty()) {
        int where = active_vertices.begin()->second;
        if (where == target) return min_distance[where];
        active_vertices.erase( active_vertices.begin() );
        for (auto ed : graph[where])
            if (min_distance[ed.to] > min_distance[where] + ed.length) {
                active_vertices.erase( { min_distance[ed.to], ed.to } );
                min_distance[ed.to] = min_distance[where] + ed.length;
                active_vertices.insert( { min_distance[ed.to], ed.to } );
            }
    }
    return INT_MAX;
}
```

4 Tree

4.1 LCA in tree (Adjacency Matrix)

```
const int MAXN = 100000;  //Can be changed
const int MAXLN = 21;   //Can be changed
vector<int> tree[MAXN];
int depth[MAXN];
int par[MAXLN][MAXN];
void dfs(int nod, int parent) {
        for (int next : tree[nod]) {
            if (next == parent) continue;
            depth[next] = depth[nod] + 1;
            par[0][next] = nod;
            dfs(next, nod);
        }
}
void prepare_lca() {
    const int root = 0;
    dfs(root, -1);
    par[0][root] = root;
    for (int i = 1; i < MAXLN; ++i)
        for (int j = 0; j < n; ++j)
            par[i][j] = par[i - 1][par[i - 1][j]];
}
// find lowest common ancestor in tree between u & v
// assumption : must call 'prepare_lca' once before call this
// O(logV)
int lca(int u, int v) {
    if (depth[u] < depth[v]) swap(u, v);
    if (depth[u] > depth[v]) {
        for (int i = MAXLN - 1; i >= 0; --i)

            if (depth[u] - (1 << i) >= depth[v])
                u = par[i][u];
    }
    if (u == v) return u;
    for (int i = MAXLN - 1; i >= 0; --i) {
        if (par[i][u] != par[i][v]) {
            u = par[i][u];
            v = par[i][v];
        }
    }
    return par[0][u];
}
```

4.1.1 Rooted Tree distance between u to v

```
int d[40001],p[40001],l[40001],c[40001];
vector<pair<int,int>> a[40001];

void bfs()
{
    d[1]=p[1]=l[1]=0,c[1]=true;
    queue<int> q;
    q.push(1);
    while(!q.empty())
    {
        int now=q.front(); q.pop();
        for(auto i : a[now])
        {
            int next=i.first,cost=i.second;
            if(c[next])continue;
            c[next]=true;
            d[next]=d[now]+cost;
            p[next]=now;
            l[next]=l[now]+1;
            q.push(next);
        }
    }
}
int lca(int u,int v)
{
    if(l[u]<l[v]) swap(u,v);
    while(l[u]!=l[v]) u=p[u];
    while(u!=v) u=p[u],v=p[v];
    return u;
}

int main()
{
    int n,m,u,v,c;
    scanf("%d",&n);
    while(--n)
    {
        // from u - > v : cost = c
        scanf("%d %d %d",&u,&v,&c);
        a[u].push_back({v,c});
        a[v].push_back({u,c});
    }
    bfs();
    scanf("%d",&m);
    while(m--){
        scanf("%d %d",&u,&v);
        printf("%d\n",d[u]+d[v]-2*d[lca(u,v)]);
    }
    return 0;
}
```

4.2 Union & Find

```
// ll is long long
// Vll is vector<ll>

typedef vector<ll> Vll;

ll find(Vll& C, ll x){ return (C[x] == x) ? x : C[x] = find(C,C[x]); }
void merge(Vll& C, ll x, ll y){ C[find(C, x)] = find(C,y); }  //union
```

4.3 Kruskal (Making MST)

```
#include <iostream>
#include <cstdio>
#include <vector>
#include <utility>
#include <algorithm>

using namespace std;

typedef long long lld;
typedef long double Ld;
typedef pair<lld, lld> pll;

#define sci(n) scanf("%lld", &(n))

typedef pair<lld, pll> edge;

void link(vector<int> &, int, int);
int find(vector<int> &, int);

int main(void)
{
    lld n, e;
    sci(n), sci(e);
    vector<edge> edges;
    for (int i = 0; i < e; ++i)
    {
        lld v, w, c;
        sci(v), sci(w), sci(c);
        edges.push_back(edge(c, pll(v, w)));
    }
    vector<int> uf(n + 1);
    for (int i = 0; i < n + 1; ++i)    uf[i] = i;
    sort(edges.begin(), edges.end());
    lld cost = 0;
    for (int i = 0; i < e; ++i)
    {
        lld v = edges[i].second.first, w = edges[i].second.second;
        if (find(uf, v) != find(uf, w))
        {
            link(uf, v, w);
            cost += edges[i].first;
        }
    }

    cout << cost << endl;

    return 0;
}

void link(vector<int> &uf, int x, int y)
{
    uf[find(uf, x)] = find(uf, y);
}

int find(vector<int> &uf, int x)
{
    if (uf[x] == x)    return x;
    return uf[x] = find(uf, uf[x]);
}
```

5 Flow, Matching

5.1 Hopcroft-Karp Bipartite Matching

```
constint MAXN =50005, MAXM =50005;
vector<int> gph[MAXN];
int dis[MAXN], l[MAXN], r[MAXM], vis[MAXN];
void clear(){for(int i =0; i<MAXN; i++) gph[i].clear();}
void add_edge(int l, int r){ gph[l].push_back(r);}
bool bfs(int n){
    queue<int> que;
```

```
    bool ok =0;
    memset(dis, 0, sizeof(dis));
    for(int i =0; i<n; i++){
        if(l[i]==-1&&!dis[i]){
            que.push(i);
            dis[i]=1;
        }
    }
    while(!que.empty()){
        int x = que.front();
        que.pop();
        for(auto&i : gph[x]){
            if(r[i]==-1) ok =1;
            elseif(!dis[r[i]]){
                dis[r[i]]= dis[x]+1;
                que.push(r[i]);
            }
        }
    }
    return ok;
}
bool dfs(int x){
    if(vis[x])return0;
    vis[x]=1;
    for(auto&i : gph[x]){
        if(r[i]==-1¦¦(!vis[r[i]]&& dis[r[i]]== dis[x]+1&& dfs(r[i]))){
            l[x]= i; r[i]= x;
            return1;
        }
    }
    return 0;
}
//match(n) Get Number of Maximum Matching (Bipertite) match(0):from 0th
int match(int n){
    memset(l, -1, sizeof(l));
    memset(r, -1, sizeof(r));
    int ret =0;
    while(bfs(n)){
        memset(vis, 0, sizeof(vis));
        for(int i =0; i<n; i++)if(l[i]==-1&& dfs(i)) ret++;
    }
    return ret;
}
bool chk[MAXN + MAXM];
void rdfs(int x, int n){
    if(chk[x])return;
    chk[x]=1;
    for(auto&i : gph[x]){
        chk[i + n]=1;
        rdfs(r[i], n);
    }
}
vector<int> getcover(int n, int m){// solve min. vertex cover
    match(n);
    memset(chk, 0, sizeof(chk));
    for(int i =0; i<n; i++)if(l[i]==-1) rdfs(i, n);
    vector<int> v;
    for(int i =0; i<n; i++)if(!chk[i]) v.push_back(i);
    for(int i = n; i<n + m; i++)if(chk[i]) v.push_back(i);
    return v;
}
```

5.2 Dinic Algorithm(Get Max Flow in $O(V^2E)$)

```
constint MAXN =505;
struct edg {int pos, cap, rev;};
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){
    gph[s].push_back({ e, x, (int)gph[e].size()});
    gph[e].push_back({ s, 0, (int)gph[s].size()-1});
}
int dis[MAXN], pnt[MAXN];
bool bfs(int src, int sink){
    memset(dis, 0, sizeof(dis));
    memset(pnt, 0, sizeof(pnt));
    queue<int> que;
    que.push(src);
    dis[src]=1;
    while(!que.empty()){
        int x = que.front();
```

```
        que.pop();
        for(auto&e : gph[x]){
            if(e.cap>0&&!dis[e.pos]){
                dis[e.pos]= dis[x]+1;
                que.push(e.pos);
            }
        }
    }
    return dis[sink]>0;
}
int dfs(int x, int sink, int f){
    if(x == sink)return f;
    for(; pnt[x]< gph[x].size(); pnt[x]++){
        edg e = gph[x][pnt[x]];
        if(e.cap>0&& dis[e.pos]== dis[x]+1){
            int 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;
}
//Get Network Flow (match)
int match (int src, int sink){
    int ret =0;
    while(bfs(src, sink)){
        int r;
        while((r = dfs(src, sink, 2e9))) ret += r;
    }
    return ret;
}
```

6 String

6.1 KMP

```
typedef vector<int> seq_t;
void calculate_pi(vector<int>& pi, const seq_t& str) {
    pi[0] = -1;
    for (int i = 1, j = -1; i < str.size(); i++) {
        while (j >= 0 && str[i] != str[j + 1]) j = pi[j];
        if (str[i] == str[j + 1])
            pi[i] = ++j;
        else
            pi[i] = -1;
    }
}

// returns all positions matched
// O(|text|+|pattern|)
vector<int> kmp(const seq_t& text, const seq_t& pattern) {
    vector<int> pi(pattern.size()), ans;
    if (pattern.size() == 0) return ans;
    calculate_pi(pi, pattern);
    for (int i = 0, j = -1; i < text.size(); i++) {
        while (j >= 0 && text[i] != pattern[j + 1]) j = pi[j];
        if (text[i] == pattern[j + 1]) {
            j++;
            if (j + 1 == pattern.size()) {
                ans.push_back(i - j);
                j = pi[j];
            }
        }
    }
    return ans;
}
```

6.1.1 How the KMP Algorithm is working (with Failure Func.)

KMP Algorithm은 패턴 P와 문자열 S가 있을 때, 문자열 S에 패턴 P가 부분문자열로 있는지, 혹은 몇 개 있는지 선형시간 O(|S|+|P|)에 확인하는 방법이다.

기본적인 아이디어는 **실패 함수** f(i)에서 온다. 아래는 패턴 PP가 "abracadabra"일 때 실패 함수 값을 표로 나타낸 것이다.

i	1	2	3	4	5	6	7	8	9	10	11
P	a	b	r	a	c	a	d	a	b	r	a

f	0	0	0	1	0	1	0	1	2	3	4

P[i,j]를 패턴 P의 i번째 문자부터 j번째 문자까지로 구성된 부분문자열이라 하자. 실패 함수 f(i)는 i보다 작은 값이며, P[i-f(i)+1,i]와 P[1,f(i)]가 같도록 되는 최대값으로 정의된다.

패턴 P의 실패함수는 아래처럼 O(¦P¦)에 계산할 수 있다.

```
int k = 0;
F[1] = 0;
for (int i=2;i<=¦P¦;i++){
    while (k && P[k+1] != P[i]) k = F[k];
    if (P[k+1] == P[i]) k++;
    F[i] = k;
}
```

위 방법의 시간복잡도가 O(¦P¦)라는 것을 쉽게 보이기 위해서는 4번째 줄의 while문에 집중할 필요가 있다. while문을 한 번 돌 수록 변수 k는 적어도 1 감소하게된다. 변수 k는 0에서부터 시작해서 최대 ¦P¦만큼 증가하므로 감소 또한 최대 ¦P¦번 일어날 수 있다. 따라서 전체 시간복잡도가 O(¦P¦)가 된다.

실패 함수를 계산하는 것과 문자열 S에 패턴 P가 존재하는지 확인하는 것은 완전히 똑같다.

```
int k = 0;
for (int i=1;i<=¦S¦;i++){
    while (k && P[k+1] != S[i]) k = F[k];
    if (P[k+1] == S[i]) k++;
    if (k == ¦P¦){
        // MATCHED WITH S[i-¦P¦+1, i]!!
        k = F[k];
    }
}
```

실패 함수를 구할 때와 마찬가지의 이유로 시간복잡도가 O(¦S¦)이다.

6.2 Aho-Corasick Algorithm (and its usage)

```
//https://gist.github.com/koosaga/96e5de4ccb99616f9bc3a760ec964cbe
const int MAXN = 100005, MAXC = 26;
struct aho_corasick{
    int trie[MAXN][MAXC], piv; // trie
    int fail[MAXN]; // failure link
    int term[MAXN]; // output check
    void init(vector<string> &v){
        memset(trie, 0, sizeof(trie));
        memset(fail, 0, sizeof(fail));
        memset(term, 0, sizeof(term));
        piv = 0;
        for(auto &i : v){
            int p = 0;
            for(auto &j : i){
                if(!trie[p][j]) trie[p][j] = ++piv;
                p = trie[p][j];
            }
            term[p] = 1;
        }
        queue<int> que;
        for(int i=0; i<MAXC; i++){
            if(trie[0][i]) que.push(trie[0][i]);
        }
        while(!que.empty()){
            int x = que.front();
            que.pop();
            for(int i=0; i<MAXC; i++){
                if(trie[x][i]){
                    int p = fail[x];
                    while(p && !trie[p][i]) p = fail[p];
                    p = trie[p][i];
                    fail[trie[x][i]] = p;
                    if(term[p]) term[trie[x][i]] = 1;
                    que.push(trie[x][i]);
                }
            }
        }
    }
    bool query(string &s){
        int p = 0;
        for(auto &i : s){
            while(p && !trie[p][i]) p = fail[p];
            p = trie[p][i];
            if(term[p]) return 1;
```

```
        }
        return 0;
    }
}aho_corasick;
```

6.3 Usage of std::string

```
//문자열내부에 있는 특정문자 모두 치환
string ReplaceAll(string &str, const string& from, const string& to){
    size_t start_pos = 0; //string처음부터 검사
    while((start_pos = str.find(from, start_pos)) != string::npos)
    //from을 찾을 수 없을 때까지
    {
        str.replace(start_pos, from.length(), to);
        start_pos += to.length();
    // 중복검사를 피하고 from.length() > to.length()인 경우를 위해서
    }
    return str;
}
//공백제거

str.erase(std::remove(str.begin(), str.end(), ' '), str.end());
//이렇게 하면 str안에 있는 문자열이 지워진다.

#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
int main(){
string str=”Hello World”;
str.erase(remove(str.begin(),str.end(), ‘ ‘), str.end());
cout<<str<<endl; //HelloWorld
string output = ReplaceAll(str, "Hello", "Hell");
     cout<<output<<endl; //HellWorld

return 0;
}
```

7 Special Data Structure

7.1 Segment Tree

```
#include <iostream>
#include <cstdio>
#include <vector>
using namespace std;

typedef long long ll;

vector<ll> seg;
vector<ll> arr;
vector<ll> lazy;

void init(int, int, int);
void propagate(int, int, int);
void update(int, int, int, ll, int, int);
ll sum(int, int, int, int, int);

int main(void)
{
    int n, m, k;
    cin >> n >> m >> k;
    arr.resize(n + 1);
    lazy.resize(n * 4, 0);
    seg.resize(n * 4);
    for (int i = 0; i < n; ++i)
        cin >> arr[i + 1];

    init(1, 1, n);
    for (int i = 0; i < m + k; ++i)
    {
        int a, b, c;
        cin >> a >> b >> c;
        if (a == 1)
        {
            update(1, 1, n, c - arr[b], b, b);
            arr[b] = c;
        }
        else
```

```
            cout << sum(1, 1, n, b, c) << endl;
    }

    return 0;
}

void init(int nn, int l, int r)
{
    if (l == r)
    {
        seg[nn] = arr[l];
        return;
    }
    int mid = (l + r) / 2;
    init(nn * 2, l, mid);
    init(nn * 2 + 1, mid + 1, r);
    seg[nn] = seg[nn * 2] + seg[nn * 2 + 1];
}
//Lazy Propagation 쓰는 Segtree임
void propagate(int nn, int nl, int nr)
{
    if (lazy[nn] != 0)
    {
        if (nl != nr)
        {
            lazy[nn * 2] += lazy[nn];
            lazy[nn * 2 + 1] += lazy[nn];
        }
        seg[nn] += lazy[nn] * (nr - nl + 1);
        lazy[nn] = 0;
    }
}
void update(int nn, int nl, int nr, ll k, int l, int r)
{
    propagate(nn, nl, nr);

    if (nr < l || r < nl)
        return;
    if (l <= nl && nr <= r)
    {
        lazy[nn] += k;
        propagate(nn, nl, nr);
        return;
    }
    int mid = (nl + nr) / 2;
    update(nn * 2, nl, mid, k, l, r);
    update(nn * 2 + 1, mid + 1, nr, k, l, r);
    seg[nn] = seg[nn * 2] + seg[nn * 2 + 1];
}

ll sum(int nn, int nl, int nr, int l, int r)
{
    propagate(nn, nl, nr);

    if (nr < l || r < nl)
        return 0;
    if (l <= nl && nr <= r)
        return seg[nn];
    int mid = (nl + nr) / 2;
    return sum(nn * 2, nl, mid, l, r) + sum(nn * 2 + 1, mid + 1, nr, l, r);
}
```

7.2 Convex Hull Trick(For Dynamic Prgramming)

일부 dp문제에서 시간복잡도를 획기적으로 줄여주는 걸로 유명한 테크닉입니다.
이번에 koi 2014 전국본선 3번으로 나왔으니 인지도가 더 올라갈 거 같네요.

개략적으로 설명하자면

$$dp[i] = \min_{j<i}(dp[j] + a[i]b[j]) \text{ (단. } b[i\text{-}1] >= b[i])$$

문제를 풀다가 이런 형태의 점화식이 나올 때는 보통 n^2 말고는 희망이 없는데

$$f(i) = \min_{j<i}(a[i]b[j] + dp[j])$$

$$f(x = a[i]) = \min_{j<i}(b[j] * x + dp[j])$$

이걸 이런 식으로 해석하면 기울기와 절편이 j에 따라 결정되는 형태의 일차함수

들로 해석할수 있게 됩니다.
이러면 저러한 dp식을 구할때

(dp[0] = 0)
1. 0번 선분을 넣는다 (기울기 = b[0]. 절편 = dp[0])
2. 현재 들어간 선분 중 최솟값을 찾는다 (dp[1])
3. 1번 선분을 넣는다 (기울기 = b[1]. 절편 = dp[1])
4. 현재 들어간 선분 중 최솟값을 찾는다 (dp[2])
.....

즉,

* dp[i]를 구하고
* 선분을 넣어주는

연산을 O(1)이나 O(lgn)에 할 수 있는 자료구조가 있으면 시간복잡도 향상을 꾀할 수 있습니다.

이 때 convex hull trick은 저 작업들을 모조리 O(lgn)만에 할 수 있습니다.
뿐만 아니라 a[i] <a[i+1]일 경우에는 O(1)만에 해버릴 수도 있습니다. (사실 스위핑이라서 정확히 O(1)은 아닙니다. 그냥 O(n/n)..)

```
#include <cstdio>
typedef long long lint;

lint a[100005],b[100005],d[100005];
lint la[100005], lb[100005];

int sz,p,n;
double cross(int x, int y){return (double)(lb[y] - lb[x]) / (la[x] - la[y]);
}

void insert(lint p, lint q){
    la[sz] = p;
    lb[sz] = q;
    // 일단 넣고
    while(sz>1 && cross(sz-1,sz-2) > cross(sz-1,sz)){
        // 자신 - (sz-1) 교점이 (sz-1) - (sz-2) 교점보다 앞에 있을때
        // sz-1 원소는 필요가 없다
        la[sz-1] = la[sz];
        lb[sz-1] = lb[sz];
        sz--;
    }
    sz++;
}

lint query(lint x){
    while (p+1 < sz && cross(p,p+1) <= x) p++;
    // 교점이 x 뒤에 있는 원소들은 모두 pop_front
    return lb[p] + la[p] * x;
}

int main(){
    scanf("%d",&n);
    for (int i=0; i<n; i++) {
        scanf("%I64d",&a[i]);
    }
    for (int i=0; i<n; i++) {
        scanf("%I64d",&b[i]);
    }
    insert(b[0],0);
    for (int i=1; i<n; i++) {
        d[i] = query(a[i]);
        insert(b[i],d[i]);
    }
    printf("%I64d",d[n-1]);
}
```

7.3 Custom Heap with C++ STL, lambda

```
#include <queue>
struct cmp{
    operator bool()(<T> a, <T> b){
        return <compare_value>;
    }
};

priority_queue< ll, vector<ll>, cmp)> min_heap;
```

7.4 FFT(Fast Fourier Transform)-Multiplication of polynomial

```
//출처: http://blog.myungwoo.kr/54
#define _USE_MATH_DEFINES
#include <math.h>
#include <complex>
#include <vector>
#include <algorithm>
using namespace std;

#define sz(v) ((int)(v).size())
#define all(v) (v).begin(),(v).end()
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;
    }
}
```

```
//다항식 a와 b의 convolution vector을 구함(합성곱) O(nlogn)
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;
    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
));
}
```

7.4.1 FFT Problem Example

문제) 크기가 N인 정수 배열 A와 크기가 M인 정수 배열 B가 있다. ($M \le N \le 500,000$). 크기가 M인 A의 (연속한) 부분 배열 C가 있을 때, 함수 f의 정의는 다음과 같다. $f(C) = \sum_{i=0}^{M} B[i] \times C[i]$이 때, f(C)가 최대가 되는 C를 찾아 f(C) 값을 구하시오.

8 Geometry

8.1 Convex Hull 2D(Monotone Chain)

```
// Convex Hull, Monotone chain by O(nlogn)
// 윗껍질, 아랫껍질에 각 각 끝 점이 중복되니 주의.#include <iostream>
#include <utility>
#include <vector>
#include <algorithm>
#include <cstdio>
#define x first
#define y second
usingnamespace std;
typedef longlong ll;
typedef pair<int,int> pii;
pii operator-(pii a, pii b){return{a.x-b.x, a.y-b.y};}
ll cross(pii a, pii b){return b.y*1ll*a.x- b.x*1ll*a.y;}
bool ccw(pii a, pii b, pii c){return cross(b-a, c-a)>=0;}
pair<vector<pii>, vector<pii>> ConvexHull(vector<pii> pt){
```

```
    sort(pt.begin(),pt.end());
    vector<pii> uhl,dhl;
    int un=0,dn=0;
    for(int i=0;i<pt.size();i++){
        while(un>=2&& ccw(uhl[un-2],uhl[un-1],pt[i])){
            uhl.pop_back(), un--;
        }
        uhl.push_back(pt[i]);
        un++;
    }
    reverse(pt.begin(),pt.end());
    for(int i =0;i<pt.size();i++){
        while(dn>=2&& ccw(dhl[dn-2],dhl[dn-1],pt[i])){
            dhl.pop_back(), dn--;
        }
        dhl.push_back(pt[i]); dn++;
    }
    return{uhl,dhl};
}

int main(){
    int n;scanf("%d",&n);
    vector< pii > v;
    for(int i =0;i<n;i++){
       pii p;
       scanf("%d%d",&p.first, &p.second); v.push_back(p);
}
    pair<vector<pii>, vector<pii>> ans = ConvexHull(v);
    printf("%d",(int)ans.first.size()+(int)ans.second.size()-2);
    return0;
}
```

8.2 Closest Pair(2D) by Line Sweeping

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cmath>
#include <vector>
#include <set>
using namespace std;

typedef long long ll;

typedef struct _dot {
    _dot() {};
    _dot(int x, int y) {
        this->_x = x;
        this->_y = y;
    }
    int _x;
    int _y;

    bool operator < (const _dot &v) const {
        if (_y == v._y) {
            return _x < v._x;
        }
        else {
            return _y < v._y;
        }
    }

} dot ;

auto cmp_x = [](dot& a, dot& b) {
    return (a._x < b._x);
};

auto cmp_y = [](dot& a, dot& b) {
    return (a._y < b._y);
};

ll distance(dot& a, dot& b) {
    ll x = a._x, xx = b._x, y = a._y, yy = b._y;
    return (((x - xx)*(x - xx)) + ((y - yy)*(y - yy)));
}

int main() {
    vector<dot> v;
    set<dot> s;
    int n; scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        dot temp; int _x, _y;
        scanf("%d%d", &_x, &_y);
        v.push_back(dot(_x,_y));
```

```
    }
    sort(v.begin(), v.end(), cmp_x);
    s = { v[0],v[1] };
    ll max_dist = distance(v[0], v[1]);

    //일단 set을 사용하고, idx 2~n까지 전체탐색 하면서
    //처음 잰 거리보다 작은 dot들을 다 cutting 하는거임 그리고 일단
    //다 Bianry tree에 넣어
    ll start = 0;
    for (int idx = 2; idx < v.size(); idx++) {
        dot cur = v[idx];
        while(start<idx) {
            dot p = v[start];
            ll x = cur._x - p._x;
            if (x*x > max_dist) {
                s.erase(p);
                start++;
            }
            else {
                break;
            }
        }
        ll dist = max_dist + 1;
        auto lo = dot(-10e5, cur._y-dist);
        auto hi = dot(10e5, cur._y +dist);
        auto lb = s.lower_bound(lo);
        auto hb = s.upper_bound(hi);
        for (auto iter = lb; iter != hb; iter++) {

            dot cmp_dot;
            cmp_dot._x = (*iter)._x;
            cmp_dot._y = (*iter)._y;
            ll d = distance(cur, cmp_dot);
            if (d < max_dist) {
                max_dist = d;
            }
        }
        s.insert(cur);
    }
    cout << max_dist;
    return 0;
}
```

9 Pre Coding

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstdio>
#include <vector>
#include <utility>
#include <algorithm>
#include <string>
#include <queue>
#include <stack>
#include <tuple>
#include <list>
#include <set>
#include <map>
#include <functional>
#include <cmath>
#include <cstring>
#include <cstdlib>
usingnamespace std;
typedeflonglong ll;
typedef pair<ll, ll> pll;

#define xx first
#define yy second
#define $1 first
#define $2 second

void solveCase(){
    ios::sync_with_stdio(false);
    cin.tie(NUL); cout.tie(NULL);

}
int main(){

#ifdef _DEBUG
    freopen("input.txt", "r", stdin);
#endif
    solveCase();
}
```

```
// LIS
int arr[1001], L[1001], P[1001], len, N;        //arr[n]에 input 들어 있음
void backtrace(int idx, int num) {
    if (idx ==0) return;
    if (P[idx] == num) {
        backtrace(idx -1, num -1);
        cout<< arr[idx] <<" ";
    } else
        backtrace(idx -1, num);
}
int main() {
    for (int i =1; i <= N; ++i) {
        auto pos = lower_bound(L +1, L + len +1, arr[i]);
        *pos = arr[i];
        P[i] = distance(L, pos);
        if (pos == L + len +1) len++;
    }
    cout<< len <<"\n";
    backtrace(N, len);
    return 0;
}
```

```
#SEG TREE 슈도 코드(sum 기준)
const int SIZE = (1 << 21)        //약 200만 이상
const int MAX_INTER = (1 << 20) - 1
초기화
for(int a = MAX_INTER; a >= 1; a--) tree[a] = tree[a * 2] + tree[a * 2 + 1]
function sub_sum(left, right)
    left == MAX_INTER;     right += MAX_INTER;
    loop(left <= right)
        //왼쪽 노드의 parent 값을 활용하려면 left child여야 한다.
        if(left % 2 == 1) sum += tree[left++];
        //오른쪽 노드의 parent 값을 활용하려면 right child여야 한다.
        if(right % 2 == 0) sum += tree[right--];
        left /= 2;
        right /= 2;

function update(idx, val)
    idx += MAX_INTER;
    tree[idx] = val;
    idx /= 2;
    while(idx >= 1)
        tree[idx] = tree[idx * 2] + tree[idx * 2 + 1];
        idx /= 2;

# 에라토스 테네스의 체
    bool check[1001] = { 0, };
    for (int i =2; i <= n; i++) {
        if (!check[i]) //i번째를 아직 지우지 않았다면 소수이다.
            for (int j =1; j <= (n / i); j++)
                if (!check[i * j]) check[i * j] =true;
    }
```

```
//Trie
#defineNULL0
constint MAX_ITER =26;
struct Node {
    bool finish;
    Node *next[MAX_ITER];
    void insert(char*key) {
```

```
        if (*key =='\0')
            finish =true;
        else {
            int cur =*key -'0';
            if (next[cur] ==NULL)
                next[cur] =new Node();
            next[cur]->insert(key +1);
        }
    }
    bool find(char*key) {
        if (*key =='\0') returnfalse;
        if (finish) returntrue;
        int cur =*key -'0';
        return next[cur]->find(key +1);
    }
};

// 다익스트라. 1..N ==> 0... N-1.
출발1 도착N (1:1)포함. (출발 N, 도착 1인 경우 모든 edge들을 뒤집어서 반대로 풀자)
음수 cost 처리 불가능. O(NlogN).
음수 cost가 없는 N:N인 경우 다익스트라를 N번 돌리는게 플로이드 워셜보다 빠르다.
const int MAX_V =20000;
const int INF =123456789; // 절대 나올 수 없는 경로값
typedefpair<int, int> P;
int main(){
    int V, E, K;
    vector<P> adj[MAX_V]; // (이어진 정점 번호, 거리)
    cin>> V >> E >> K;
    K--;
    for (int i =0; i < E; i++) {
        int u, v, w;
        cin>> u >> v >> w;
        adj[u -1].push_back(P(v -1, w));
    }
    int dist[MAX_V], prev[MAX_V];
    fill(dist, dist+MAX_V, INF);
    bool visited[MAX_V] = {0};
    priority_queue<P, vector<P>, greater<P>> PQ;
    // 다익스트라 알고리즘
    dist[K] =0; // 시작점으로의 거리는 0
    PQ.push(P(0, K)); // 시작점만 PQ에 넣고 시작
    while(!PQ.empty()){ // PQ가 비면 종료
        int curr;
        do{
            curr = PQ.top().second;
            PQ.pop();
        }while(!PQ.empty() && visited[curr]); // curr가 방문한 정점이면 무시
        // 더 이상 방문할 수 있는 정점이 없으면 종료
        if(visited[curr]) break;
        visited[curr] =true; // 방문
        for(auto& p: adj[curr]){
            int next = p.first, d = p.second;
            // 거리가 갱신될 경우 PQ에 새로 넣음
            if(dist[next] > dist[curr] + d){
                dist[next] = dist[curr] + d;
                prev[next] = curr;
                PQ.push(P(dist[next], next));
            }
        }
    }
```

```
}

// 벨만 포드. 음수 cost 처리 가능 O(V^2). 음수 cycle이 있는지 판별 가능.
음수 cycle 찾는 법은 알고리즘 수행 후, 대각 선을 봐서 0이 아닌 것이 1개라도 있으면 음수 cycle이 존재하는 것이다.
#include<vector>
#include<utility>
#include<algorithm>
#include<iostream>
usingnamespacestd;
typedefpair<int, int> P;
constint INF =123456789; // 절대 나올 수 없는 경로값
int main() {
    int N, M, dist[500];
    cin>> N >> M;
    vector<P> adj[500];
    for (int i =0; i < M; i++) {
        int A, B, C;
        cin>> A >> B >> C;
        adj[A -1].push_back(P(B -1, C));
    }
    bool minusCycle =false;
    fill(dist, dist + N, INF);
    dist[0] =0;
    for (int i =0; i < N; i++) { // (N-1) + 1번의 루프. 마지막은 음의 싸이클 존재 여부 확인용
        for (int j =0; j < N; j++) {
            // N-1번의 루프에 걸쳐 각 정점이 i+1개 정점을 거쳐오는 최단경로 갱신
            for (auto&p : adj[j]) {
                int next = p.first, d = p.second;
                if (dist[j] != INF && dist[next] > dist[j] + d) {
                    dist[next] = dist[j] + d;
                    // N번째 루프에 값이 갱신되면 음의 싸이클이 존재한다.
                    if (i == N -1) minusCycle =true;
                }
            }
        }
    }
    if (minusCycle) puts("-1");
    else {
        for (int i =1; i < N; i++)
            printf("%d\n", dist[i] != INF ? dist[i] : -1);
    }
}

# 플로이드 워셜. 음수 cost 처리 가능. O(N^3)
#include<algorithm>
#include<iostream>
usingnamespacestd;
constint INF =123456789; // 절대 나올 수 없는 경로값
int main() {
    int N, M, dist[100][100];
    cin>> N >> M;
    // dist 배열 초기화
    for (int i =0; i < N; i++)
        for (int j =0; j < N; j++)
            dist[i][j] = i == j ? 0 : INF;
    // 간선 정보 입력받음
    for (int i =0; i < M; i++) {
```

```
        int a, b, c;
        cin>> a >> b >> c;
        dist[a -1][b -1] = min(dist[a -1][b -1], c);
        // 플로이드 와샬 알고리즘 적용
        for (int k =0; k < N; k++)
            for (int i =0; i < N; i++)
                for (int j =0; j < N; j++)
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
        // 이제 dist 배열은 실제 최단 경로를 담고 있음
    }
}
```

외적의 결과는 벡터 값. 외적의 크기는 평행사변형의 넓이와 같다.
외적 값이 0이면, 두 벡터 사이의 각이 0도 이거나 180도(즉 a와 b가 평행)
교차 판별. ccw(a, b, c) * ccw(a, b, d) < 0 && ccw(c, d ,a) * ccw(c, d, b) < 0 이어야 교차. 접하는 경우 판단하려면 ccw 값이 둘 다 0인 경우를 주의해야 한다.(교점이 무수히 많거나 없는 경우). 둘 중 하나만 0이면 수직일 수도 있고, ccw가 0인 경우에 대해 신경을 많이 써야 한다.

LCA

```
#include<iostream>
#include<cstdio>
#include<vector>
#include<algorithm>
#include<cmath>
#include<queue>
usingnamespacestd;
#define NUM 20
int n;
vector<int> level(100001, 0);
vector<vector<int>> parent(100001, vector<int>(NUM, 0));
vector<vector<int>> graph(100001);
queue<int> q;
int lca(int a, int b);
int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int a, b;
    //그래프 정보 입력 받는 부분
    cin>> n;
    for (int i =1; i <= n -1; i++) {
        cin>> a >> b;
        graph[a].push_back(b);
        graph[b].push_back(a);
    }
    //초기화, root의 level을 1로 설정
    level[1] =1;
    //그래프를 바탕으로 트리로 변환하는 과정.
    //level과 parent를 갱신하면서 진행. level이 0이면 아직 미방문
    //시작 노드를 1로 하고 BFS 탐색.
    q.push(1);
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        for (int i =0; i < graph[node].size(); i++) {
            int dst = graph[node][i];
            //아직 미방문했다는 뜻. 즉, dst가 node의 child
            if (level[dst] ==0) {
                level[dst] = level[node] +1;
                //dst의 2^0번째 부모가 node라는 것을 업데이트.
```

```
                parent[dst][0] = node;
                q.push(dst);
            }
        }
    }
    //여기까지가 트리를 생성하는 사전 작업.
    //parent table 갱신
    for (int i =1; i < NUM; i++)
        for (int j =1; j <= n; j++)
            parent[j][i] = parent[parent[j][i -1]][i -1];
    //쿼리 입력 받고 처리하는 부분
    int m;
    cin>> m;
    int ans;
    for (int i =0; i < m; i++) {
        cin>> a >> b;
        ans = lca(a, b);
        printf("%d\n", ans);
    }
    return0;
}
int lca(int a, int b) {
    int lca_node =0;
    while (lca_node ==0) {
        //높이가 같을 때
        if (level[a] == level[b]) {
            //a와 b가 동일한지 확인해서 같으면 return a
            if (a == b)
                return a;
            //다르면 lca를 찾아 위로 올라간다.
            //parent table 값을 이용해서 올라가는데, 테이블의 가장 큰 쪽(2^19)부터 시작해서 해당 위치의 조상 노드가 같으면 안 올라가고
            //다를 경우 올라간다. 그러다 보면 최소 공통 조상(lca) 노드 바로 아래쪽 까지 올라가진다.
            for (int i = NUM -1; i >=0; i--) {
                if (parent[a][i] != parent[b][i]) {
                    a = parent[a][i];
                    b = parent[b][i];
                }
            }
            //이제, 현재 보고 있는 노드의 부모 노드가 최소공통조상(lca) 노드가 된다.
            return lca_node = parent[a][0];
        }
        //높이가 다를 경우, 더 깊이 있는 것을 올려서 높이를 같게 만든다.
        else {
            int gap, two =0;
            //두 노드의 깊이 차이 계산
            gap = abs(level[a] - level[b]);
            if (level[a] < level[b]) {
                while (gap >0) {
                    //b를 위로 올리는 작업
                    if (gap % 2==1)
                        b = parent[b][two];
                    gap /=2;
                    two++;
                }
            }
            else {
                while (gap >0) {
```

```
                    //a를 위로 올리는 작업
                    if (gap % 2==1)
                        a = parent[a][two];
                    gap /=2;
                    two++;
                }
            }
        }
    }
}
```

min heap

priority_queue<T, vector<T>, greater<T> > pq;

lower bound는 찾고자 하는 값 이상이 처음으로 나타나는 위치

upper bound는 찾고자 하는 값보다 큰 값이 처음으로 나타나는 위치.