algo-know-yeah Team Note

ID	이름
singun11	김신건
antifly55	서형빈
Neogulee	최성훈

Table of contents

7. Lis
8. Knapsack
9. Coin Change

0. Base

```
#include <bits/stdc++.h>

#define for1(s,n) for(int i = s; i < n; i++)
#define for1j(s,n) for(int j = s; j < n; j++)
#define foreach(k) for(auto i : k)
#define foreachj(k) for(auto j : k)
#define pb(a) push_back(a)
#define sz(a) a.size()

using namespace std;
typedef unsigned long long ull;
typedef long long ll;
typedef vector <int> iv1;
typedef vector <vector<int>> iv2;
typedef vector <ll> llv1;
typedef vector <llv1> llv2;
typedef unsigned int uint;
typedef vector <ull> ullv1;
typedef vector <vector <ull>> ullv2;
typedef pair<int, int> pii;
typedef pair<ll, ll> ll;

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

}
```

1. Graph

1.1. dijkstra

```
#define MAX 100000
#define INF (ll)1e18

struct edge {
    ll node;
    ll cost;
    bool operator<(const edge &to) const {
        return cost > to.cost;
    }
};
```

```
struct WGraph {
    ll n;
    vector<vector<edge>> adj;
    llv1 prev;
    WGraph(ll n) : n{n}, adj(n+1) {}
    void addEdge(ll s, ll e, ll cost) {
        adj[s].push_back({e, cost});
    }

    void input(ll m) { // 단방향
        ll a, b, c;
        while(m--) {
            cin >> a >> b >> c;
            addEdge(a,b,c);
        }
    }

    void inputD(ll m) { // 양방향
        ll a, b, c;
        while(m--){
            cin >> a >> b >> c;
            addEdge(a,b,c);
            addEdge(b,a,c);
        }
    }

    llv1 dijkstra(ll s) {
        llv1 dist(n+1, INF);
        prev.resize(n+1, -1);
        priority_queue<edge> pq;
        pq.push({ s, 0ll });
        dist[s] = 0;
        while (!pq.empty()) {
            edge cur = pq.top();
            pq.pop();
            if (cur.cost > dist[cur.node]) continue;
            for (auto &nxt : adj[cur.node])
                if (dist[cur.node] + nxt.cost < dist[nxt.node]) {
                    prev[nxt.node] = cur.node;
                    dist[nxt.node] = dist[cur.node] + nxt.cost;
                    pq.push({ nxt.node, dist[nxt.node] });
                }
        }
        return dist;
    }

    llv1 getPath(ll s, ll e) {
        llv1 ret;
        ll current = e;
        while(current != -1) {
            ret.push_back(current);
            current = prev[current];
        }
```

```
        reverse(ret.begin(), ret.end());
        return ret;
    }
};
```

1.2. bellman-ford

```
#define MAX 100010
#define INF (ll)1e18

struct edge {
    int to, cost;
};

int n;
vector<edge> v[MAX];
ll D[MAX];
bool bellman(ll start_point){
    fill(D,D+n+1, INF);
    D[start_point] = 0;

    bool isCycle = false;
    for1(1, n+1) {
        for1j(1, n+1) {
        for(int k=0; k<sz(v[j]); k++) {
            edge p = v[j][k];
            int end = p.to;
            ll dist = D[j] + p.cost;
                if (D[j] != INF && D[end] > dist) {
                    D[end] = dist;
                    if (i == n) isCycle = true;
                }
            }
        }
    }
    return isCycle;
}
```

1.3. kruskal

```
int root[MAXN];
int level[MAXN];

class Edge{
public:
    int node[2];
    int distance;
    Edge(int a, int b, int distance){
        this->node[0] = a;
```

```
        this->node[1] = b;
        this->distance = distance;
    }

    bool operator<(Edge &edge){
        return this->distance < edge.distance;
    }
};

void init(int n) {
    for1(0, n){
        root[i] = i;
        level[i] = 1;
    }
}

int find(int x) {
    return root[x] == x ? x : root[x] = find(root[x]);
}

// merge와 동시에 cycle 여부 확인
bool merge(int x, int y) {
    x = find(x);
    y = find(y);
    if (x == y) return true;
    if (level[x] < level[y]) root[x] = y;
    else root[y] = x;
    if (level[x] == level[y]) level[x]++;
    return false;
}
```

1.4. prim

```
struct edge {
    ll crt;
    ll node, cost;
};

struct WGraph {
    ll V;
    vector<edge> adj[MAX];
    vector<ll> prev;
    WGraph(ll V) : V{V} {}
    void addEdge(ll s, ll e, ll cost) {
        adj[s].push_back({s, e, cost});
        adj[e].push_back({e, s, cost});
    }

    ll prim(vector<edge> &selected) { // selected에 선택된 간선정보 vector 담김
        selected.clear();
```

```
        vector<bool> added(V, false);
        llv1 minWeight(V, INF), parent(V, -1);

        ll ret = 0;
        minWeight[0] = parent[0] = 0;
        for (int iter = 0; iter < V; iter++) {
            int u = -1;
            for (int v = 0; v < V; v++) {
                if (!added[v] && (u == -1 || minWeight[u]>minWeight[v]))
                    u = v;
            }

            if (parent[u] != u)
                selected.push_back({parent[u], u, minWeight[u]});

            ret += minWeight[u];
            added[u] = true;

            for1(0, sz(adj[u])) {
                int v = adj[u][i].node, weight = adj[u][i].cost;
                if (!added[v] && minWeight[v]>weight) {
                    parent[v] = u;
                    minWeight[v] = weight;
                }
            }
        }
        return ret;
    }
};
```

1.5. topological sort

```
int n;
int link[MAXN];
iv1 graph[MAXN];

void topologySort() {
    iv1 result;
    queue<int> q;

    for1(1, n+1) {
        if(link[i] == 0) q.push(i);
    }

    while(!q.empty()) {
        int x = q.front();
        q.pop();
        result.pb(x);

        for1(0, sz(graph[x])) {
            int y = graph[x][i];
```

```
            if(--link[y]==0) q.push(y);
        }
    }

    for1(0, n) {
        cout << result[i] << " ";
    }
}
```

1.6. union-find

```
int root[MAXN];
int level[MAXN];

void init(int n) {
    for1(0, n){
        root[i] = i;
        level[i] = 1;
    }
}

int find(int x) {
    return root[x] == x ? x : root[x] = find(root[x]);
}

void merge(int x, int y) {
    x = find(x);
    y = find(y);
    if (x == y) return;
    if (level[x] < level[y]) root[x] = y;
    else root[y] = x;
    if (level[x] == level[y]) level[x]++;
}
```

1.7. SCC

```
int ans, cnt;
int visit[MAX], sn[MAX];
bool finished[MAX];
vector<int> adj[MAX];
stack<int> st;
vector<vector<int> > scc;

int dfs(int curr){
    visit[curr] = ++cnt;
    st.push(curr);

    int result = visit[curr];
    for(int i = 0; i < adj[curr].size(); i++){
```

```
            int next = adj[curr][i];
            if(visit[next] == 0) result = min(result, dfs(next));
            else if(!finished[next]) result = min(result, visit[next]);
        }

        if(result == visit[curr]){
            vector<int> currSCC;
            while(1){
                int t = st.top();
                st.pop();
                currSCC.push_back(t);
                finished[t] = true;
                sn[t] = ans;
                if(t == curr) break;
            }
            sort(currSCC.begin(), currSCC.end());
            scc.push_back(currSCC);
            ans++;
        }
        return result;
    }

    void makeSCC(int v){
        for(int i=1; i<=v; i++)
            if(!visit[i]) dfs(i);
        sort(scc.begin(), scc.end());
    }
```

1.8. Maximum flow(dinic)

```
#define MAX_V 101
#define SRC 1
#define SINK MAX_V-1
#define INF (ll)1e18

struct Edge {
    ll v, capacity, rev;
    Edge(ll v, ll capacity, ll rev): v(v), capacity(capacity), rev(rev) {}
};

vector<Edge> vt[MAX_V];
ll level[MAX_V];
ll work[MAX_V];


void addEdge(ll start, ll end, ll capacity) {
    vt[start].emplace_back(end, capacity, (ll)vt[end].size());
    vt[end].emplace_back(start, capacity, (ll)vt[start].size()-1);
}

// 레벨 그래프 만드는 BFS
```

```
bool bfs() {
    memset(level, -1, sizeof(level));        //레벨 그래프 초기화
    queue <ll> q;
    level[SRC] = 0;
    q.push(SRC);

    while(!q.empty()){
        int here = q.front(); q.pop();
        for (auto i : vt[here]) {
            ll there = i.v;
            if(level[there] == -1 && i.capacity > 0) {
                level[there] = level[here] + 1;
                q.push(there);
            }
        }
    }
    return level[SINK] != -1;
}


ll dfs(ll here, ll crt_capacity) {
    if(here == SINK) return crt_capacity;

    for(ll &i = work[here]; i < vt[here].size(); i++) {
        ll there = vt[here][i].v;
        ll capacity = vt[here][i].capacity;

        if(level[here] + 1 == level[there] && capacity > 0) {
            ll next_capacity = dfs(there, min(crt_capacity, capacity));

            if(next_capacity > 0) {
                vt[here][i].capacity -= next_capacity;
                vt[there][vt[here][i].rev].capacity += next_capacity;
                return next_capacity;
            }
        }
    }
    return 0;
}

ll dinic() {
    ll ret = 0;
    while(bfs()) {
        memset(work, 0, sizeof(work));

        while(1) {
            ll flow = dfs(SRC, INF);
            if(!flow) break;
            ret += flow;
        }
    }
    return ret;
}
```

1.9. Maximum flow minimum cost

```
#define MX_N 100
#define MX_NODE 2*(MX_N+2)
#define SRC MX_NODE-2
#define SINK MX_NODE-1
#define INF 1000000000

ll N, M;
ll cost[MX_NODE][MX_NODE]; // 각 간선의 Cost
ll capacity[MX_NODE][MX_NODE]; // 각 간선의 용량
ll flow[MX_NODE][MX_NODE]; // 각 간선에 흐르고 있는 유량
llv1 edge[MX_NODE]; // 각 정점의 인접리스트

ll MCMF() {
    ll ret = 0;
    while(1) {
        ll prev[MX_NODE], dist[MX_NODE];
        bool isInQ[MX_NODE];
        queue<ll> Q;
        fill(prev, prev+MX_NODE, -1);
        fill(dist, dist+MX_NODE, INF);
        fill(isInQ, isInQ+MX_NODE, false);

        dist[SRC] = 0;
        Q.push(SRC);
        isInQ[SRC] = true;

        while(!Q.empty()) {
            ll current = Q.front();
            Q.pop();

            isInQ[current] = false;

            for(ll next: edge[current])
                if(capacity[current][next] - flow[current][next] > 0 && dist[next]
> dist[current] + cost[current][next]) {
                    dist[next] = dist[current] + cost[current][next];
                    prev[next] = current;

                    if(!isInQ[next]) {
                        Q.push(next);
                        isInQ[next] = true;
                    }
                }
        }

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

        ll current_flow = INF;
```

```
        for(ll i = SINK; i != SRC; i = prev[i])
            current_flow = min(current_flow, capacity[prev[i]][i] - flow[prev[i]]
[i]);

        for(ll i = SINK; i != SRC; i = prev[i]) {
            ret += current_flow * cost[prev[i]][i];
            flow[prev[i]][i] += current_flow;
            flow[i][prev[i]] -= current_flow;
        }
    }
    return ret;
}
```

2. Tree

2.1. segment tree

```
ll a[MAX], tree[MAX * 4];

void init(int node, int x, int y) {
    if (x == y) {
        tree[node] = a[x];
        return;
    }
    int mid = (x + y)/2;
    init(node*2, x, mid);
    init(node*2 + 1, mid + 1, y);
    tree[node] = tree[node*2] + tree[node*2 + 1];
}

void update(int pos, ll val, int node, int x, int y) {
    if (pos < x || pos > y) return;
    if (x==y) {
        tree[node] = val;
        return;
    }
    int mid = (x + y)/2;
    update(pos, val, node*2, x, mid);
    update(pos, val, node*2 + 1, mid + 1, y);
    tree[node] = tree[node*2] + tree[node*2 + 1];
}

ll query(int lo, int hi, int node, int x, int y) {
    if (lo > y || hi < x) return 0;
    if (lo <= x && y <= hi) return tree[node];
    int mid = (x + y)/2;
    return query(lo, hi, node*2, x, mid) + query(lo, hi, node*2 + 1, mid + 1, y);
}
```

2.2. segment tree with lazy propagation

```
ll seg[4 * MAX], lazy[4 * MAX];

void update_lazy(ll node, ll x, ll y) {
    if (!lazy[node])
        return;
    seg[node] += (y - x + 1)*lazy[node];
    if (x != y) {
        lazy[node * 2] += lazy[node];
        lazy[node * 2 + 1] += lazy[node];
    }
    lazy[node] = 0;
}

ll update(ll lo, ll hi, ll val, ll node, ll x, ll y) {
    update_lazy(node, x, y);
    if (y < lo || hi < x)
        return seg[node];
    if (lo <= x && y <= hi) {
        lazy[node] += val;
        update_lazy(node, x, y);
        return seg[node];
    }
    ll mid = (x + y)/2;
    return seg[node] = update(lo, hi, val, node * 2, x, mid) + update(lo, hi, val,
node * 2 + 1, mid + 1, y);
}

ll query(ll lo, ll hi, ll node, ll x, ll y) {
    update_lazy(node, x, y);
    if (y < lo || hi < x)
        return 0;
    if (lo <= x && y <= hi)
        return seg[node];
    ll mid = (x + y)/2;
    return query(lo, hi, node * 2, x, mid) + query(lo, hi, node * 2 + 1, mid + 1,
y);
}
```

2.3. merge sort tree

```
llv1 a;
llv1 mTree[Mx];
void makeTree(ll idx, ll ss, ll se) {
    if(ss == se) {
        mTree[idx].push_back(a[ss]);
        return;
    }

    ll mid = (ss+se)/2;
```

```
    makeTree(2*idx+1, ss, mid);
    makeTree(2*idx+2, mid+1, se);

    merge(mTree[2*idx+1].begin(), mTree[2*idx+1].end(), mTree[2*idx+2].begin(),
mTree[2*idx+2].end(), back_inserter(mTree[idx]));
}

ll query(ll node, ll start, ll end, ll q_s, ll q_e, ll k) {
    //i j k: Ai, Ai+1, ..., Aj로 이루어진 부분 수열 중에서 k보다 큰 원소의 개수를 출력
한다.
    if (q_s > end || start > q_e) return 0;

    if (q_s <= start && q_e >= end) {
        return mTree[node].size() - (upper_bound(mTree[node].begin(),
mTree[node].end(), k) - mTree[node].begin());
    }

    ll mid = (start+end)/2;
    ll p1 = query(2*node+1, start, mid, q_s, q_e, k);
    ll p2 = query(2*node+2, mid+1, end, q_s, q_e, k);
    return p1 + p2;
}
```

2.4. LCA

```
struct LCA {
    vector<int> serials, no2se, se2no, loc; // length, loc의 index는 no
    vector<ll> length;
    SegmentTree *seg; // 최솟값 segment tree
    LCA(vector<vector<pair<int, ll>>> &edges) {
        int N = edges.size();
        no2se = vector<int>(N, -1);
        se2no = vector<int>(N, -1);
        loc = vector<int>(N, -1);
        length = vector<ll>(N, -1);
        length[0] = 0;
        vector<bool> visited(N, false);

        init_serials(0, visited, edges);
        seg = new SegmentTree(serials);
        seg->init(1, 1, serials.size());
    }

    void init_serials(int current, vector<bool>& visited, vector<vector<pair<int,
ll>>>&edges) {
        static int cnt = 0;
        visited[current] = true;
        if (no2se[current] == -1) {
            no2se[current] = cnt++;
            se2no[no2se[current]] = current;
            loc[current] = serials.size();
```

```
        }
        serials.push_back(no2se[current]);
        for1(0, edges[current].size()) {
            int next = edges[current][i].first;
            int cost = edges[current][i].second;
            if (visited[next])
                continue;
            length[next] = length[current] + cost;
            init_serials(next, visited, edges);
            serials.push_back(no2se[current]);
        }
        visited[current] = false;
    }

    ll query(int u, int v) { // 두 정점 사이의 거리
        if (loc[u] > loc[v])
            swap(u, v);
        ll lca = seg->query(loc[u] + 1, loc[v] + 1, 1, 1, serials.size());
        return length[u] + length[v] - 2ll * length[se2no[lca]];
    }
};
```

3. String

3.1. KMP

```
string content;
string obj;
int fail[MX];

vector <int> kmp (string s, string o) {
    fill(fail,fail+MX,0);
    vector<int> result;
    int N = s.length();
    int M = o.length();
    for(int i=1, j=0; i<M; i++){
        while(j > 0 && o[i] != o[j]) j = fail[j-1];
        if(o[i] == o[j]) fail[i] = ++j;
    }
    for(int i = 0, j = 0; i < N; i++) {
        while(j > 0 && s[i] != o[j]) j = fail[j-1];
        if(s[i] == o[j]) {
            if(j == M-1) { // matching OK;
                result.push_back(i - M + 2);
                j = fail[j];
            }
            else j++;
        }
    }
    return result;
}
```

3.2. Trie

```
const int ALPHABETS = 26;

int chToIdx(char ch) { return ch - 'a'; }
struct Trie {
    bool check = false;
    Trie* chil[ALPHABETS];
    Trie() {
        for (int i = 0; i < ALPHABETS; i++)
            chil[i] = NULL;
    }
    ~Trie() {
        for (int i = 0; i < ALPHABETS; i++)
            if (chil[i])
                delete chil[i];
    }
    void insert(string& s, int idx = 0) {
        if (idx == s.size()) {
            check = true;
            return;
        }
        int next = chToIdx(s[idx]);
        if (chil[next] == NULL)
            chil[next] = new Trie();
        chil[next]->insert(s, idx + 1);
    }
    bool find(string& s, int idx = 0) {
        if (idx == s.size())
            return check;
        int next = chToIdx(s[idx]);
        if (chil[next] == NULL)
            return false;
        return chil[next]->find(s, idx + 1);
    }
};
```

4. Geometry

4.1. convexHull

```

```

5. Extra

5.1. Treap

5.2. MCC

5.3. ExtendEuclid

```
int gcd(int a, int b){
    if(b==0) return a;
    return gcd(b, a%b);
}

// ax+by=gcd(a,b)
pii ext_gcd(int a, int b){
    if(b==0) return pii(1, 0);
    pii tmp = ext_gcd(b, a%b);
    return pii(tmp.second, tmp.first - (a/b) * tmp.second);
}

// ax = 1 (mod b)
ll mod_inv(int a, int b){
    return (ext_gcd(a, b).first + b) % b;
}
```

5.4. Fermat

```
ll pow(ll a, ll b){
    if(b == 0) return 1;
    ll n = pow(a, b/2)%p;
    ll temp = (n * n)%p;

    if(b%2==0) return temp;
    return (a * temp)%p;
}

ll fermat(ll a, ll b){
    return a%p*pow(b, p-2)%p;
}
```

5.5. FFT

```
const double PI = acos(-1);
typedef complex<double> cpx;

void FFT(vector<cpx> &f, cpx w){
    int n = f.size();
    if(n == 1) return;

    vector<cpx> even(n/2), odd(n/2);
    for(int i = 0; i < n; ++i)
        (i%2 ? odd : even)[i/2] = f[i];

    FFT(even, w*w);
    FFT(odd, w*w);

    cpx wp(1, 0);
    for(int i = 0; i < n/2; ++i){
        f[i] = even[i] + wp*odd[i];
        f[i + n/2] = even[i] - wp*odd[i];
        wp *= w;
    }
}

vector<cpx> multiply(vector<cpx> a, vector<cpx> b){
    int n = 1;
    while(n < a.size()+1 || n < b.size()+1) n *= 2;
    n *= 2;
    a.resize(n);
    b.resize(n);
    vector<cpx> c(n);

    cpx w(cos(2*PI/n), sin(2*PI/n));

    FFT(a, w);
    FFT(b, w);

    for(int i = 0; i < n; ++i)
        c[i] = a[i]*b[i];

    FFT(c, cpx(1, 0)/w);
    for(int i = 0; i < n; ++i){
        c[i] /= cpx(n, 0);
        c[i] = cpx(round(c[i].real()), round(c[i].imag()));
    }
    return c;
}
```

5.6. ConvexHullTrick

```
struct linear{
    ll a, b;
```

```
    double s;
};

ll dp[MAX], top=0;
linear f[MAX];

double cross(linear &f, linear &g){
    return (g.b-f.b)/(f.a-g.a);
}

void addLine(ll a, ll b){ // y = ax + b
    linear g({a, b, 0});
    while(top > 0){
        g.s = cross(f[top-1], g);
        if(f[top-1].s < g.s) break;
        top--;
    }
    f[top++] = g;
}

ll searchLine(ll x){
    ll pos = top-1;
    if(x < f[top-1].s){
        ll lo = 0, hi = top-1;
        while(lo+1 < hi){
            ll mid = (lo+hi)/2;
            (x < f[mid].s ? hi:lo) = mid;
        }
        pos = lo;
    }
    return pos;
}
```

5.7. LIS

```

```

5.8. Knapsack

```
ll N, maxWeight,ans;
ll D[2][11000];
ll weight[110], cost[110];

void knapsack() {
    for(int x=1; x<=N; x++) {
        for(int y=0; y<=maxWeight; y++) {
            if(y>=weight[x]) {
                D[x%2][y] = max(D[(x+1)%2][y],D[(x+1)%2][y-weight[x]]+cost[x]);
            }
```

```
            else {
                D[x%2][y] = D[(x+1)%2][y];
            }
            ans = max(ans, D[x%2][y]);
        }
    }
}

void input() {
    cin >> N >> maxWeight;
    for(int x=1; x<=N; x++) {
        cin >> weight[x] >> cost[x];
    }
}
```

5.9. Coin Change

```
ll CC(llv1& coin, ll money, ll MX) {
    ll D[MX];
    fill(D, D+MX, 0);
    D[0] = 1;
    for(int i = coin.size()-1; i >=0; i--) {
        for(int j = coin[i]; j <= money; j++) {
            D[j] += D[j - coin[i]];
            D[j] %= MOD;
        }
    }
    return D[money] % MOD;
}
```