목차

그래프 관련

(1) scc

```
struct SCC {
private:
	int n, m, componentNum;
	std::vector< std::vector<int> > edge, rev_edge;
	std::vector<int> number, order;
	std::vector<bool> visit;

	void dfs(int x) {
		visit[x] = true;
		for (auto it : edge[x]) {
			if (visit[it]) continue;
			dfs(it);
		}
		order.push_back(x);
	}
	void rev_dfs(int x) {
		visit[x] = true;
		number[x] = componentNum;
		for (auto it : rev_edge[x]) {
			if (visit[it]) continue;
			rev_dfs(it);
		}
	}
public:
	SCC() {};
	SCC(int _n) {
		n = _n;
		number.clear(); edge.clear(); rev_edge.clear();
		edge.resize(n + 1);
		rev_edge.resize(n + 1);
		number.resize(n + 1);
	}
	// usage
	void push_edge(int x, int y) {
		edge[x].push_back(y);
		rev_edge[y].push_back(x);
	}
	void scc() {
		visit.clear(); visit.resize(n + 1);
		order.clear();
		FOR(i, 1, n) {
			if (visit[i]) continue;
			dfs(i);
		}
		visit.clear(); visit.resize(n + 1);
		componentNum = 0;
		for (int i = n - 1; i >= 0; i--) {
			if (visit[order[i]]) continue;
			componentNum++;
			rev_dfs(order[i]);
		}
	}
	int getComponentNumber() {
		return componentNum;
	}
	int getSpecificNumber(int x) {
		return number[x];
	}
	std::vector<int> getTotalNumber() {
		return number;
	}
};
```

(2) bcc

```
struct bcc_tree{
    int N;
    std::vector<vector<int>> v;
    std::vector<bool> chk;
    std::vector<int> bcc,d,low,parent;
    int ti;

    bcc_tree(){}
    bcc_tree(int size){
        N = size;
        bcc.clear(); v.clear();
        chk.resize(N+2); d.resize(N+2); low.resize(N+2); parent.resize(N+2);
        ti = 0;

        vector<int> temp;
        for (int i=0;i<=N;i++){
            v.push_back(temp);
            chk[i]=false; d[i]=low[i]=-1; parent[i] = i;
        }
    }
    void insert(int a,int b){
        v[a].push_back(b); v[b].push_back(a);
```

```
    }
    void tarjan(int a){
        bool isart = false;
        int child = 0;
        chk[a] = true;
        d[a]=low[a]=ti++;

        for (int i=0;i<v[a].size();i++){
            int b = v[a][i];
            if (!chk[b]){
                parent[b]=a;
                tarjan(b);
                child++;
                if (low[b] >= d[a] && parent[a]!=a) isart=true;
                low[a]=min(low[a],low[b]);
            }
            else if (parent[a]!=b) low[a]=min(low[a],d[b]);
        }
        if ((parent[a]!=a && isart) || (parent[a]==a && child > 1)){
            bcc.push_back(a);
        }
    }
    std::vector<int> get_bcc(){
        for (int i=1;i<=N;i++) if (!chk[i]) tarjan(i);
        return bcc;
    }
};
```

(3) max_flow (dinic)

```
#include <stdio.h>
#include <vector>
#include <memory.h>
#include <queue>
#define minf(a,b) ((a)<(b)?(a):(b))
#define INF 0x7fffffff/2
using namespace std;

struct Edge {
    Edge(int s, int t, int cap,int rev) {
        this->s = s; this->t = t;
        this->cap = cap; this->rev = rev;
    }
    int s,t,cap,rev;
};

struct Dinic {
private:
    static const int SIZE = 2555;

    int source, sink;
    int level[SIZE], iter[SIZE];
    vector<int> graph[SIZE];
    vector<Edge> edges;

    void bfs(int s) {
        memset (level, -1, sizeof(level));
        queue<int> Q;
        level[s] = 0;
        Q.push(s);
        while (!Q.empty()) {
            int v = Q.front(); Q.pop();
            for (auto it : graph[v]) {
                Edge &e = edges[it];
                if (e.cap > 0 && level[e.t] < 0) {
                    level[e.t] = level[v] + 1;
                    Q.push(e.t);
                }
            }
        }
    }
    int dfs(int v, int t, int f) {
        if (v == t) return f;
        for (int &i = iter[v]; i < graph[v].size(); i++) {
            Edge &e = edges[graph[v][i]];
            if (e.cap > 0 && level[v] < level[e.t]) {
                int d = dfs(e.t, t, minf(f, e.cap));
                if (d > 0) {
                    e.cap -= d;
                    edges[e.rev].cap += d;
                    return d;
                }
            }
        }
        return 0;
    }

public:
```

```
    Dinic(int source,int sink) {
        this->source = source;
        this->sink = sink;
    }
    void addEdge(int s,int t,int cap) {
        int z = (int) edges.size();
        edges.push_back(Edge(s,t,cap,z+1)); graph[s].push_back(z);
        edges.push_back(Edge(t,s,0,z)); graph[t].push_back(z+1);
    }
    int getFlow() {
        int flow = 0;
        while(1) {
            bfs(source);
            if (level[sink] < 0) return flow;
            memset(iter, 0 ,sizeof iter);
            int f;
            while ((f = dfs(source, sink, INF)) > 0) flow += f;
        }
    }
};
```

(4) mcmf

```
nclude <vector>
#include <stdio.h>
#include <stdlib.h>
#include <queue>
#define INF 0x7fffffff/2
#define FOR(i,n,m) for (int i=(n);i<=(m);i++)
using namespace std;

struct MCMF{
    int SIZE;
    int source, sink;
    vector<vector<int>> flow,cost,v;
    vector<int> dist;
    vector<int> in_que,parent;

    MCMF(int source,int sink,int SIZE) { // 노드의 마지막이 sink가 아닌 경우
        this->source = source; this->sink = sink; this->SIZE = SIZE;
        v.resize(SIZE+2);
        dist.resize(SIZE+2); in_que.resize(SIZE+2); parent.resize(SIZE+2);
        FOR(i,0,SIZE+2){
            flow.push_back(dist);
            cost.push_back(dist);
        }
    }
    void add_edge(int a,int b,int c){
        if (!flow[a][b] && !flow[b][a]) v[a].push_back(b), v[b].push_back(a);
        flow[a][b]=1; cost[a][b]=c; cost[b][a]=-c;
    }
    void add_edge(int a,int b,int c,int f){
        if (!flow[a][b] && !flow[b][a]) v[a].push_back(b), v[b].push_back(a);
        flow[a][b]=f; cost[a][b]=c; cost[b][a]=-c;
    }
    bool SPFA(){
        FOR(i,0,SIZE) dist[i]=INF, in_que[i]=parent[i]=0;
        queue<int> q;
        dist[source]=0; q.push(source); in_que[source]=1;
        while(!q.empty()){
            int a = q.front(); q.pop();
            in_que[a] = 0;
            for (auto b : v[a]){
                if (!flow[a][b]) continue;
                if (dist[b] > dist[a]+cost[a][b]){
                    dist[b] = dist[a]+cost[a][b];
                    parent[b]=a;
                    if (!in_que[b]) in_que[b]=1,q.push(b);
                }
            }
        }
        return dist[sink] != INF;
    }
    pair<int,int> get_MCMF(){
        int ans=0,cnt=0;
        while (SPFA()){
            cnt++;
            ans += dist[sink];
            for (int i=sink;i!=source;i=parent[i]) flow[parent[i]][i]--, flow[i][parent[i]]++;
        }
        return make_pair(ans, cnt);
    }
};
```

(5) 이분 매칭(HK)

```
#include <stdio.h>
#include <memory.h>
```

```
#include <vector>
#include <queue>
using namespace std;

/*
 시간복잡도 O(M sqrt N)

 Usage
 Hopcroft_Karp hk(n,m);
 // 간선 연결
 // V : 1~n,  U : 1~m
 // V의 a와 U의 b와 연결
 hk.graph[a].push_back(b);

 // 매칭수
 int matching = hk.solve();

 // 매칭결과
 hk.mx[x] // 정점 x와 매칭된 U의 정점번호
 hk.my[y] // 정점 y와 매칭된 V의 정점번호
*/

struct Hopcroft_Karp {
    static const int SIZE = 1010;

    int n,m;
    int mx[SIZE], my[SIZE];
    int total;

    int dist[SIZE];
    int inf_dist;

    vector<int> graph[SIZE];

    Hopcroft_Karp(int n,int m) {
        this->n = n; this->m = m;
        memset(mx, -1, sizeof(mx));
        memset(my, -1, sizeof(my));
        memset(dist, 0, sizeof(dist));
        total = 0;
    }

    void bfs() {
        queue<int> q;
        for(int i=1; i<=n; i++){
            if (mx[i] == -1) { dist[i] = 0; q.push(i); }
            else dist[i] = -1;
        }
        while (!q.empty()) {
            int x = q.front(); q.pop();
            for (int i=0; i<graph[x].size(); i++) {
                int y = graph[x][i];
                if (my[y] != -1 && dist[my[y]] == -1) {
                    dist[my[y]] = dist[x] + 1;
                    q.push(my[y]);
                }
            }
        }
    }

    bool dfs(int x) {
        for (int i=0; i<graph[x].size(); i++) {
            int y = graph[x][i];
            if (my[y] == -1 || (dist[my[y]]==dist[x]+1 && dfs(my[y]))) {
                mx[x] = y;
                my[y] = x;
                return true;
            }
        }
        dist[x] = -1;
        return false;
    }

    int solve() {
        int flow;
        do {
            bfs();
            flow = 0;
            for (int i=1;i<=n;i++) if (mx[i] == -1 && dfs(i)) flow ++;
            total += flow;
        } while(flow);
        return total;
    }
};
```

(6) Bellman-ford

```
// https://www.acmicpc.net/problem/11657
```

```
#include <stdio.h>
#include <vector>
#define FOR(i,n,m) for (int i=(n);i<=(m);i++)
#define si(n) fscanf(in,"%d",&n)
//FILE *in = fopen("input.txt", "r"), *out = fopen("output.txt", "w");
FILE *in = stdin, *out = stdout;
using namespace std;

typedef int ll;
struct EDGE {
	int v; ll cnt;
	EDGE() {};
	EDGE(int _v, ll _cnt) :v(_v), cnt(_cnt) {};
};
struct BellmanFord {
private:
	int n, m;
	std::vector< std::vector<EDGE> > edge; // TODO : Change size as maximum N
	std::vector<ll> dist;
	bool bellmanford(int start) {
		FOR(i, 1, n) dist[i] = 0x7fffffff;
		dist[start] = 0;
		FOR(k, 1, n - 1) {
			FOR(i, 1, n) {
				if (dist[i] == 0x7fffffff) continue;
				for (int j = 0; j < edge[i].size(); j++) {
					ll y = edge[i][j].v;
					ll cnt = edge[i][j].cnt;
					dist[y] = dist[y] < dist[i] + cnt ? dist[y] :
dist[i] + cnt;
				}
			}
		}

		bool neg_cycle = false;
		FOR(i, 1, n) {
			if (dist[i] == 0x7fffffff) continue;
			for (int j = 0; j < edge[i].size(); j++) {
				ll y = edge[i][j].v;
				ll cnt = edge[i][j].cnt;
				if (dist[y] > dist[i] + cnt) neg_cycle = true;
			}
		}
		return neg_cycle;
	}

public:
	BellmanFord() {};
	BellmanFord(int _n) {
		n = _n;
		dist.clear(); edge.clear();
		dist.resize(n + 1);
		edge.resize(n + 1);
	}
	void push_edge(int x, int y, ll cnt) {
		edge[x].push_back(EDGE(y, cnt));
		//edge[y].push_back(EDGE(x, cnt)); // bi-directed case
	}
	bool calc(int start) {
		return bellmanford(start);
	}
	std::vector<ll> getDist() {
		return dist;
	}
};
int n, m;
int main() {
	BellmanFord bf;

	si(n), si(m);
	bf = BellmanFord(n);
	FOR(i, 1, m) {
		int x, y, cnt;
		si(x), si(y), si(cnt); // x->y, dist : cnt
		bf.push_edge(x, y, cnt); // push edge to graph
	}

	if (bf.calc(1)) {
		fprintf(out, "-1"); // negative cycle
		return 0;
	}
	std::vector<ll> ans = bf.getDist(); // get shortest distance
	FOR(i, 2, n) {
		fprintf(out, "%d₩n", ans[i] == 0x7fffffff ? -1 : ans[i]);
	}
	return 0;
}
```

트리

(1) Segment Tree

```
#include <vector>
typedef long long int ll;
struct segment_tree { // Lazy Propagation
private:
    int B, N;
    std::vector<ll> tree;
    std::vector<ll> accum_tree;
    ll update(int x, int L, int R, int range_L, int range_R, ll cnt, ll f(ll, ll)) {
        // L, R : current node's rane
        // range_L, range_R : target range
        if (L > range_R || R < range_L) return DEFAULT;
        if (L > R || range_L > range_R) return DEFAULT;
        if (range_L <= L && R <= range_R) {
            tree[x] = f(tree[x], cnt);
            accum_tree[x] = f(accum_tree[x], cnt * (R - L + 1));
            return cnt*(R-L+1);
        }
        int mid = (L + R) / 2;
        ll left_v = update(x * 2, L, mid, range_L, range_R, cnt, f);
        ll right_v = update(x * 2 + 1, mid + 1, R, range_L, range_R, cnt, f);

        // For accumulate
            accum_tree[x] = f(accum_tree[x], left_v);
            accum_tree[x] = f(accum_tree[x], right_v);
        return f(left_v, right_v);
    }
    ll find_single(int x, int L, int R, int X, ll f(ll, ll)) {
        // L, R : current node's rane
        // range_L, range_R : target range
        if (L > X || R < X) return DEFAULT;
        if (L > R) return DEFAULT;
        if (L == R) return tree[x];
        int mid = (L + R) / 2;
        return f(tree[x], f(find_single(x * 2, L, mid, X, f), find_single(x * 2 + 1, mid + 1, R, X, f)));
    }
    ll find_range(int x, int L, int R, int range_L, int range_R, ll f(ll, ll), ll accum) {
        if (L > range_R || R < range_L) return DEFAULT;
        if (L > R || range_L > range_R) return DEFAULT;
        if (range_L <= L && R <= range_R) {
            return accum_tree[x] + accum * (R-L+1);
        }
        int mid = (L + R) / 2;
        ll left_v = find_range(x * 2, L, mid, range_L, range_R, f, accum + tree[x]);
        ll right_v = find_range(x * 2+1, mid+1,R, range_L, range_R, f, accum + tree[x]);
        return f(left_v, right_v);
    }

public:
    ll DEFAULT;
    segment_tree(int size, ll DEF) {
        B = N = size;
        tree.resize(N * 4);
        accum_tree.resize(N * 4);
        FOR(i, 1, N * 4 - 1) tree[i] = accum_tree[i] = DEFAULT;
    }
    void update(int L, int R, ll cnt, ll f(ll, ll)) { update(1, 1, N, L, R, cnt, f); }
    ll find_single(int X, ll f(ll, ll)) { return find_single(1, 1, N, X, f); }
    ll find_range(int L, int R, ll f(ll, ll)) { return find_range(1, 1, N, L, R, f, DEFAULT); }
};
```

(2) indexed tree

```
typedef long long int ll;
struct indexed_tree {
private:
        int N;
        std::vector<ll> tree;
        void update_tree(int x, ll cnt, ll f(ll, ll)) {
                if (x == 0) return;
                tree[x] = f(tree[x], cnt);
                update_tree(x >> 1, cnt, f);
        }
        ll find_tree(int left, int right, ll f(ll, ll)) {
                int L = left, R = right;
                ll res = DEFAULT;
                while (L <= R) {
                        if (L & 1) res = f(res, tree[L++]);
                        if (!(R & 1)) res = f(res, tree[R--]);
                        L >>= 1, R >>= 1;
                }
                return res;
        }

public:
        ll DEFAULT;
```

```
        indexed_tree() {};
        indexed_tree(ll size, ll _DEFAULT) {
                N = size;
                tree.resize(size * 2 + 2);
                FOR(i, 1, size*2 + 1) tree[i] = DEFAULT;
        }

        void update(int x, ll cnt, ll f(ll, ll)) { update_tree(x + N, cnt, f); }
        ll find(int left, int right, ll f(ll, ll)) { return find_tree(left + N, right + N, f); }
};
```

(3) persist segment tree

```
#include <stdio.h>

struct NODE{
    NODE(): v(0), left(0), right(0) {}
    int v;
    NODE *left, *right;
};

struct SegmentTree {
private:
    static const int MAXN = 100010;
    int ysize; NODE *tree[MAXN];
    NODE *make_tree(NODE *now, int s, int e, int y, int v)
    {
        if (y < s || e < y) return now;
        NODE *ret = new NODE();
        if (s == e){
            if (now) ret->v = now->v + v;
            else ret->v = v;
            return ret;
        }
        int m = (s+e) >> 1;
        ret->left = make_tree(now ? now->left : 0, s, m, y, v);
        ret->right = make_tree(now ? now->right : 0, m+1, e, y, v);
        ret->v = 0;
        if (ret->left) ret->v += ret->left->v;
        if (ret->right) ret->v += ret->right->v;
        return ret;
    }
    int get(NODE *node, int s, int e, int target_s, int target_e) {
        if (node == NULL) return 0;
        if (target_e < s || target_s > e) return 0;
        if (target_s <= s && e <= target_e) {
            return node->v;
        }
        int m = (s + e) >> 1;
        return get(node->left, s, m, target_s, target_e) + get(node->right, m+1, e, target_s,
target_e);
    }
public:
    // x좌표가 증가하는 순으로 순차적으로 불러야함, 1 ~ N
    SegmentTree(int ysize) {
        this->ysize = ysize;
        this->tree[0] = new NODE();
    }
    void update(int x,int y,int v) {
        tree[x] = make_tree(tree[x-1], 1, ysize, y, v);
    }
    int get(int x1,int x2,int y1,int y2) {
        return get(tree[x2],1,ysize,y1,y2) - get(tree[x1-1],1,ysize,y1,y2);
    }
};
```

(4) Trie

```
struct trie {
    char c; int sz, leaf;
    trie *next[33];
}*root;

trie* NEW(char c) {
    trie *re = (trie*) malloc(sizeof(trie));
    re->c = c; re->sz = 1; re->leaf = 0;
    for(int i='a'-'a'+1;i<='z'-'a'+1;i++) re->next[i] = NULL;
    return re;
}

void make_trie(trie *node,int it, int len) {
    char c = word[it];

    if(it == len+1) {
        node->leaf = 1;
        return ;
    }
```

```
    if ('A' <= c && c <= 'Z') c = c - 'A' + 'a';
    if(node->next[c-'a'+1] == NULL) node->next[c-'a'+1] = NEW(c);


    make_trie(node->next[c-'a'+1],it+1,len);
}

void dfs(trie *node, int it, int tar) {
    if(it < 0) return;
    char c = sen[it];

    if (node->leaf && dp[it]) {
        dp[tar] = 1;
        via[tar] = it;
        return ;
    }

    if (node->next[c-'a'+1] == NULL) return ;
    dfs(node->next[c-'a'+1], it-1, tar);
}
```

(5) Segment tree by 성원

```
struct segment_tree { // with Lazy Propagation
private:
    int N, Z;
    vector<ll> tree;
    vector<ll> lazy;
    void update(int x, int L, int R, int range_L, int range_R, ll val, ll f(ll, ll)) {
        // x : node's index
        // L, R : current node's range
        // range_L, range_R : target range
        if (range_R < L || R < range_L) return ;
        if (L > R || range_L > range_R) return ;
        if (range_L <= L && R <= range_R) {
            lazy[x] += val;
            return ;
        }

        if (lazy[x] != 0) {
            lazy[x*2] += lazy[x];
            lazy[x*2+1] += lazy[x];
            lazy[x] = 0;
        }

        int M = (L + R) / 2;
        update(x*2, L, M, range_L, range_R, val, f);
        update(x*2+1, M+1, R, range_L, range_R, val, f);
        tree[x] = f(tree[x*2]+lazy[x*2], tree[x*2+1]+lazy[x*2+1]);
    }
    ll find_range(int x, int L, int R, int range_L, int range_R, ll f(ll, ll)) {
        // x : node's index
        // L, R : current node's range
        // range_L, range_R : target range

        if (range_R < L || R < range_L) return DEFAULT;
        if (L > R || range_L > range_R) return DEFAULT;
        if (range_L <= L && R <= range_R) return tree[x] + lazy[x];

        int M = (L + R) / 2;
        if (lazy[x] != 0) {
            lazy[x*2] += lazy[x];
            lazy[x*2+1] += lazy[x];
            lazy[x] = 0;
        }

        ll val_L = find_range(x*2, L, M, range_L, range_R, f);
        ll val_R = find_range(x*2+1, M+1, R, range_L, range_R, f);
        tree[x] = f(tree[x*2]+lazy[x*2], tree[x*2+1]+lazy[x*2+1]);
        return f(val_L, val_R);
    }

public:
    ll DEFAULT;
    segment_tree(int size, ll DEF) {
        N = size; DEFAULT = DEF;
        for (Z=1;Z<N;Z*=2);
        tree.resize(2*Z + 1);
        lazy.resize(2*Z + 1);
        for (int i=1;i<=2*Z;i++) tree[i] = DEFAULT, lazy[i] = 0;
    }
    void update(int L, int R, ll val, ll f(ll, ll)) { update(1, 1, N, L, R, val, f); }
    ll find_single(int X, ll f(ll, ll)) { return find_range(1, 1, N, X, X, f); }
    ll find_range(int L, int R, ll f(ll, ll)) { return find_range(1, 1, N, L, R, f); }
    ll get(int x, int range_R,int L,int R, ll f(ll, ll)) {
        int M = (L + R) / 2;
        if (R - L + 1 == 1) {
```

```
            if (find_single(L, f) < 2) return L + 1;
            return L;
        }
        if (range_R <= M) return get(2*x, range_R, L, M, f);
        // M < range_R
        if (find_range(M+1, range_R, f) >= 2) {
            return get(x*2, M, L, M, f);
        }
        return get(x*2+1, range_R, M+1, R, f);
    }
    void init(int x,int L, int R, int X, ll val, ll f(ll, ll)) {
        // x : node's index
        // L, R : current node's range
        // range_L, range_R : target range
        if (X < L || R < X) return ;
        if (L > R) return ;
        if (X <= L && R <= X) {
            tree[x] = val;
            return ;
        }
        int M = (L + R) / 2;
        init(x*2, L, M, X, val, f);
        init(x*2+1, M+1, R, X, val, f);
        tree[x] = f(tree[x*2], tree[x*2+1]);
    }
};

ll MIN(ll a,ll b) {
    return minf(a,b);
}

// min(s ~ x-1) 이 2보다 큰 가장 작은 s찾기
 int y = (int) st.get(1, x-1, 1, N, MIN);
```

기하

(1) convex hull

```
#include <math.h>
#include <algorithm>
#include <vector>
using namespace std;
typedef int T;
#define vp vector<POINT>
#define EPS 1e-6

struct POINT {
        T x, y, dis;
        double angle;
        POINT() {};
        POINT(T _x, T _y) :x(_x), y(_y) {};
        bool operator()(POINT A, POINT B){
                return A.y < B.y;
        }
        bool operator<(const POINT &rhs) {
                if (-EPS < angle - rhs.angle && angle - rhs.angle < EPS) return dis < rhs.dis;
                return angle < rhs.angle;
        }
};
struct ConvexHull {
private:
        vp a, output;
        int n;

        int ccw(POINT a, POINT b, POINT c) {
                T t = (b.x - a.x)*(c.y - a.y) - (b.y - a.y)*(c.x - a.x);
                if (t > EPS) return 1;
                if (t < EPS) return -1;
                return 0;
        }

        void calc() {
                sort(a.begin(), a.end(), POINT());
                a[0].angle = -10000;
                for (int i = 1; i < n; i++) {
                        a[i].angle = atan2(a[i].y - a[0].y, a[i].x - a[0].x);
                        a[i].dis = (a[i].y - a[0].y)*(a[i].y - a[0].y) + (a[i].x -
a[0].x)*(a[i].x - a[0].x);
                }
                sort(a.begin(), a.end());

                vp stack;
                stack.push_back(a[0]);
                stack.push_back(a[1]);
                for (int i = 2; i < n; i++) {
                        while (stack.size() > 1) {
                                int m = stack.size();
                                int t = ccw(stack[m - 2], stack[m - 1], a[i]);
                                if (t <= 0) stack.pop_back();
                                else break;
```

```
                }
                stack.push_back(a[i]);
            }
            output = stack;
        }

public:
        ConvexHull() {};
        ConvexHull(vp _input) {
            a.clear(); output.clear();
            a = _input;
            n = a.size();
            calc();
        }
        vp result() {
            return output;
        }
};
```

정수론

(1) 확장 유클리드

```
/*
 *  find (c, d) which satisfy
 *  a * c + b * d = gcd(a, b)
 */

#define make_pair mp
typedef long long ll;
pair<ll, ll> extended_gcd(ll a,ll b) {
  if (b == 0) return mp(1,0);
  pair<ll, ll> t = extended_gcd(b, a%b);
  return mp(t.second, t.first - t.second * (a/b));
}
```

dp 최적화

(1) convexhull trick with container

```
#include <stdio.h>
#define maxf(a,b) ((a)>(b)?(a):(b))
#define minf(a,b) ((a)<(b)?(a):(b))
typedef long long ll;

struct CHT {
    static const int SIZE = 200020;
    ll aa[SIZE], bb[SIZE];
    int sz = 0;

    double getX(int a,int b) {
        return (1.0) * (bb[b] - bb[a]) / (aa[a] - aa[b]);
    }
    void add(ll a,ll b) {
        aa[sz] = a; bb[sz] = b;

        if (sz > 0 && aa[sz-1] == aa[sz]) {
            aa[sz-1] = aa[sz];
            bb[sz-1] = minf(bb[sz-1],bb[sz]);
            sz--;
        }

        while (sz>=2 && (getX(sz-2, sz-1) > getX(sz-1, sz))) { // get(sz-2,sz-1) < getX(sz-1, sz) 유지
            aa[sz - 1] = aa[sz];
            bb[sz - 1] = bb[sz];
            sz --;
        }
        sz ++;
    }
    ll query(ll pos) {
        int l = 0, r = sz - 2;
        ll ans = -9e18;
        while (l <= r) {
            int m = (l + r) / 2;
            if (getX(m, m+1) <= pos) {
                ans = maxf(ans, aa[m+1] * pos + bb[m+1]);
                l = m + 1;
            } else {
                ans = maxf(ans, aa[m] * pos + bb[m]);
                r = m - 1;
            }
        }
        return ans;
    }
};

/*
            int p = 0;
            for (int i=sub.sz-1;i>=0;i--) {
```

```
            if (sub.aa[i] > a) {
                sub.aa[i+1] = sub.aa[i];
                sub.bb[i+1] = sub.bb[i];
            } else {
                p = i + 1;
                break;
            }
        }
        sub.aa[p] = a; sub.bb[p] = b;
        sub.sz ++;

        if (sub.sz >= 500) {
            tmp.sz = 0;
            int s1 = 0, s2 = 0;
            while (s1 < cht.sz && s2 < sub.sz) {
                if (cht.aa[s1] < sub.aa[s2]) tmp.add(cht.aa[s1], cht.bb[s1]), s1 ++;
                else tmp.add(sub.aa[s2], sub.bb[s2]), s2 ++;
            }
            while (s1 < cht.sz) tmp.add(cht.aa[s1], cht.bb[s1]), s1 ++;
            while (s2 < sub.sz) tmp.add(sub.aa[s2], sub.bb[s2]), s2 ++;
            cht.sz = tmp.sz;
            for (int i=0;i<cht.sz;i++) cht.aa[i] = tmp.aa[i], cht.bb[i] = tmp.bb[i];
            sub.sz = 0;
*/
```

(2) divide&conquer optimazation

```
/*
 조건 1) DP 점화식 꼴
 D[t][i]=mink<i(D[t-1][k]+C[k][i])
 조건 2) A[t][i]는 D[t][i]를 만족시키는 최소 k라 할 때 아래 부등식을 만족
 A[t][i]≤A[t][i+1]

 조건 1의 점화식 꼴이고, 비용 C가 사각부등식 C[a][c]+C[b][d]≤C[a][d]+C[b][c]를 만족하는 경우
조건 2도 만족한다.
 */

#include <stdio.h>
#define maxf(a,b) ((a)>(b)?(a):(b))
#define minf(a,b) ((a)<(b)?(a):(b))
typedef long long ll;
ll dp[100010];
ll T[100010], V[100010];
int D, N;

ll cost(ll a,ll b) {
    return (b-a) * T[b] + V[a];
}

void divide (int l,int r,int p,int q) {
    if (l > r) return;
    int m = (l + r) >> 1;

    int pos = -1;
    for (int i=maxf(p, m-D);i<=minf(q, m);i++) {
        if (dp[m] < cost(i,m)) {
            dp[m] = cost(i,m);
            pos = i;
        }
    }
    divide(l,m-1,p,pos);
    divide(m+1,r,pos,q);
}

int main() {
    scanf ("%d%d",&N,&D);
    for (int i=1;i<=N;i++) scanf ("%lld",&T[i]);
    for (int i=1;i<=N;i++) scanf ("%lld",&V[i]);
    divide(1, N, 1, N);
    ll ans = 0;
    for (int i=1;i<=N;i++) ans = maxf(ans, dp[i]);
    printf ("%lld",ans);
    return 0;
}
```

Suffix Array & LCP - KCM1700

```
#include <cstdio>
#include <numeric>
#include <cstring>
#include <algorithm>
#include <vector>

using namespace std;
template<typename Ty>
struct SuffixArray
{
        vector<Ty> in;
        vector<int> out;
```

```
	template<typename Pt>
		SuffixArray(Pt begin, Pt end) : in(begin, end) {}

	vector<int> build() {
		int n = (int)in.size(), c = 0;
		vector<int> temp(n), pos2bckt(n), bckt(n), bpos(n);
		out.resize(n);
		for (int i = 0; i < n; i++) out[i] = i;
		sort(out.begin(), out.end(), [&](int a,int b) { return in[a] < in[b];});
		for(int i = 0; i < n; i++) { bckt[i] = c; if (i+1 == n || in[out[i]] != in[out[i+1]]) c++; }
		for(int h = 1; h < n && c < n; h <<= 1) {
			for (int i = 0; i < n; i++) pos2bckt[out[i]] = bckt[i];
			for (int i = n-1; i>=0; i--) bpos[bckt[i]] = i;
			for (int i = 0; i < n; i++) if (out[i] >= n-h) temp[bpos[bckt[i]]++] = out[i];
			for (int i = 0; i < n; i++) if (out[i] >= h) temp[bpos[pos2bckt[out[i]-h]]++] = out[i]-h;
			c = 0;
			for (int i = 0; i + 1 < n; i++) {
				int a = (bckt[i] != bckt[i+1]) || (temp[i] >= n-h) || pos2bckt[temp[i+1]+h] != pos2bckt[temp[i]+h];
				bckt[i] = c;
				c += a;
			}
			bckt[n-1] = c++;
			temp.swap(out);
		}
		return out;
	}

	vector<int> lcparray() {
		if (in.size() != out.size()) build();
		int n = (int)in.size();
		if (n == 0)
			return vector<int>();
		vector<int> rank(n); // temporary
		vector<int> height(n-1); // output lcp array

		for(int i = 0; i < n; i ++) rank[out[i]] = i;
		int h = 0;
		for(int i = 0; i < n; i ++) {
			if(rank[i] == 0) continue;
			int j = out[rank[i]-1];
			while(i+h < n && j+h < n && in[i+h] == in[j+h]) h++;
			height[rank[i]-1] = h;
			if(h > 0) h--;
		}
		return height;
	}
};



#include <cstdlib>
#include <ctime>
#include <string>

int main(){
	string s;
	for (int i = 0; i < 1048576; i++) s.push_back(rand()%64+48);
	SuffixArray<char> sa(s.begin(), s.end());
	sa.lcparray();
	printf("%f₩n", (double) (clock()*1000.0/CLOCKS_PER_SEC));
	return 0;
}
```

Manacher

```
int R=0,p=0;
FOR(i, 1, n) {
	if (i <= R) dy[i] = min(dy[2 * p - i], R - i);
	else dy[i] = 0;
	while (i - dy[i] - 1 >= 1 && i + dy[i] + 1 <= 2 * n - 1 && a[i - dy[i] - 1] == a[i + dy[i] + 1]) dy[i]++;
	if (i + dy[i] > R) R = i + dy[i], p = i;
    }
```

Aho-Corasick

```
#include <queue>
#define TOTLEN 2000005
#define ALPHABET 26
using namespace std;
int ahoN;
struct Node{
	int next[ALPHABET];
	int fail;
	int depth;
```

```
}node[TOTLEN];
struct Aho_Corasick{
	void push(char a[],int len){
		int x=0;
		for (int i=1;i<=len;i++){
			if (node[x].next[a[i]-'a']==0) node[x].next[a[i]-'a']=++ahoN;
			x=node[x].next[a[i]-'a'];
		}
		node[x].depth=len;
	}
	void calc(){ // make failure link
		queue<int>Q;
		Q.push(0);
		while (!Q.empty()){
			int x=Q.front(); Q.pop();

node[x].depth=node[x].depth<node[node[x].failure].depth?node[node[x].failure].depth:node[x].dept
h;
			for (int i=0;i<ALPHABET;i++){
				int y=node[x].next[i];
				if (y==0) continue;
				if (x==0) node[y].fail=0;
				else{
					int X=x;
					for(;;){
						int z=node[X].fail;
						if (node[z].next[i]!=0){

node[y].failure=node[z].next[i];
							break;
						}
						if (z==0) break;
						X=z;
					}
				}
				Q.push(y);
			}
		}
	}
};
```

KMP

```
#include <stdio.h>
#include <vector>
#include <iostream>
#include <cstdio>
#define M 2000009
using namespace std;

char S[M],T[M];
int fail[M];
vector<int> ans;
int main(){
	int i;

	cin.getline(T,M);
	cin.getline(S,M);
	int now=-1;
	fail[0]=now;
	for (i=1;S[i];i++){
		while(now>=0 && S[now+1]!=S[i]) now=fail[now];
		if (S[now+1]==S[i]) now++;
		fail[i]=now;
	}
	int len = i;
	now = -1;
	for (i=0;T[i];i++){
		while(now>=0 && S[now+1]!=T[i]) now=fail[now];
		if (S[now+1]==T[i]) now++;
		if (!S[now+1]){
			ans.push_back(i-len+2);
			now=fail[now];
		}
	}
	printf("%d₩n",(int)ans.size());
	for (i=0;i<ans.size();i++) printf("%d ",ans[i]);
	return 0;
}
```

그 외

(1) FFT

```
#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;
    }
}

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));
}
```