Team Note.

Complied 2018-11-12

Kim SeongMin

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)
```

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

```cpp
// 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)

```cpp
// 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()
```

```cpp
{
    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 $O(n^3)$

```cpp
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)

```cpp
//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 = 100;  //Can be changed
const int MAXLN = 9;   //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];
}
```

4.2 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 Ford-Fulkerson

```
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
#include <cstring>

#define MAX_V 100
#define INF 2100000000

typedef int network[MAX_V][MAX_V];

//capacity[u][v]: capacity of u -> v
//flow[u][v]: flow of u -> v
network capacity, flow;
int V, e;

int networkFlow(int, int);

int main(void)
{

    return 0;
}

int networkFlow(int source, int sink)
{
    memset(flow, 0, sizeof(flow));
    int totalFlow = 0;
    while (1)
    {
            //search out augmenting path by using BFS
            std::vector<int> parent(MAX_V, -1);
            std::queue<int> q;
            parent[source] = source;
            q.push(source);
            while (!q.empty() && parent[sink] == -1)
            {
                    int h = q.front();  q.pop();
                    for (int t = 0; t < V; ++t)
                    {
                            //search out edge such that residual capacity > 0
                            if (capacity[h][t] - flow[h][t] > 0 && parent[t] == -1)
                            {
                                    parent[t] = h;
                                    q.push(t);
                            }
                    }
            }
            //Termination condition
            if (parent[sink] == -1)
                    break;
            //determine how much send flow through augmenting path
            int amount = INF;
            for (int p = sink; p != source; p = parent[p])
                    amount = std::min(capacity[parent[p]][p] - flow[parent[p]][p], amount);
            //send flow through augmenting path
            for (int p = sink; p != source; p = parent[p])
            {
                    flow[parent[p]][p] += amount;
                    flow[p][parent[p]] -= amount;
            }

            totalFlow += amount;
    }

    return totalFlow;
}
```

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

```
struct Node
{
    vector<int> adj;
};
Node graf[MAX_N];

struct Edge
{
    int u, v, cap;
    int flow;
};
vector<Edge> E;

int v, e;
int s, t;
int dist[MAX_N];
int upTo[MAX_N];

int idd = 0;

//Dinicov algoritam za nalazenje maksimalnog protoka izmedju dva cvora
 u grafu
//Slozenost: O(V^2 * E)
```

```
inline bool BFS()
{
    for (int i=1;i<=v;i++) dist[i] = -1;
    queue<int> bfs_queue;
    bfs_queue.push(s);
    dist[s] = 0;
    while (!bfs_queue.empty())
    {
        int xt = bfs_queue.front();
        bfs_queue.pop();
        for (int i=0;i<graf[xt].adj.size();i++)
        {
            int currID = graf[xt].adj[i];
            int xt1 = E[currID].v;
            if (dist[xt1] == -1 && E[currID].flow < E[currID].cap)
            {
                bfs_queue.push(xt1);
                dist[xt1] = dist[xt] + 1;
            }
        }
    }
    return (dist[t] != -1);
}

inline int DFS(int xt, int minCap)
{
    if (minCap == 0) return 0;
    if (xt == t) return minCap;
    while (upTo[xt] < graf[xt].adj.size())
    {
        int currID = graf[xt].adj[upTo[xt]];
        int xt1 = E[currID].v;
        if (dist[xt1] != dist[xt] + 1)
        {
            upTo[xt]++;
            continue;
        }
        int aug = DFS(xt1, min(minCap, E[currID].cap - E[currID].flow)
);
        if (aug > 0)
        {
            E[currID].flow += aug;
            if (currID&1) currID--; else currID++;
            E[currID].flow -= aug;
            return aug;
        }
        upTo[xt]++;
    }
    return 0;
}

inline int Dinic()
{
    int flow = 0;
    while (true)
    {
        if (!BFS()) break;
        for (int i=1;i<=v;i++) upTo[i] = 0;
        while (true)
        {
            int currFlow = DFS(s, INF);
            if (currFlow == 0) break;
            flow += currFlow;
        }
    }
    return flow;
}

inline void addEdge(int u, int v, int cap)
{
    Edge E1, E2;

    E1.u = u, E1.v = v, E1.cap = cap, E1.flow = 0;
    E2.u = v, E2.v = u, E2.cap = 0, E2.flow = 0;

    graf[u].adj.push_back(idd++);
    E.push_back(E1);
    graf[v].adj.push_back(idd++);
    E.push_back(E2);
}
```

```
int main()
{
    v = 4, e = 5;
    s = 1, t = 4;

    addEdge(1, 2, 40);
    addEdge(1, 4, 20);
    addEdge(2, 4, 20);
    addEdge(2, 3, 30);
    addEdge(3, 4, 10);

    printf("%d\n",Dinic());

    return 0;
}
```

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

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

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 일차함수들로 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)..)

달리 할말이 없네요... 나머지는 코드포스 어셉먹인 소스코드로 대체하겠습니다.

```
#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;
    }
}

void multiply(const vector<int> &a,const vector<int> &b,vector<int> &r
es)
{
    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≤N≤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(그라함 스캔)

```
// A C++ program to find convex hull of a set of points. Refer
// https://www.geeksforgeeks.org/orientation-3-ordered-points/
// for explanation of orientation()
#include <iostream>
#include <stack>
#include <stdlib.h>
using namespace std;

struct Point
{
    int x, y;
};

// A globle point needed for  sorting points with reference
// to  the first point Used in compare function of qsort()
Point p0;

// A utility function to find next to top in a stack
Point nextToTop(stack<Point> &S)
{
    Point p = S.top();
    S.pop();
    Point res = S.top();
    S.push(p);
    return res;
}

// A utility function to swap two points
int swap(Point &p1, Point &p2)
{
    Point temp = p1;
    p1 = p2;
    p2 = temp;
}

// A utility function to return square of distance
// between p1 and p2
int distSq(Point p1, Point p2)
{
    return (p1.x - p2.x)*(p1.x - p2.x) +
        (p1.y - p2.y)*(p1.y - p2.y);
}

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
        (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0;  // colinear
    return (val > 0) ? 1 : 2; // clock or counterclock wise
}

// A function used by library function qsort() to sort an array of
// points with respect to the first point
int compare(const void *vp1, const void *vp2)
```

```
{
    Point *p1 = (Point *)vp1;
    Point *p2 = (Point *)vp2;

    // Find orientation
    int o = orientation(p0, *p1, *p2);
    if (o == 0)
        return (distSq(p0, *p2) >= distSq(p0, *p1)) ? -1 : 1;

    return (o == 2) ? -1 : 1;
}

// Prints convex hull of a set of n points.
void convexHull(Point points[], int n)
{
    // Find the bottommost point
    int ymin = points[0].y, min = 0;
    for (int i = 1; i < n; i++)
    {
        int y = points[i].y;

        // Pick the bottom-most or chose the left
        // most point in case of tie
        if ((y < ymin) || (ymin == y &&
            points[i].x < points[min].x))
            ymin = points[i].y, min = i;
    }

    // Place the bottom-most point at first position
    swap(points[0], points[min]);

    // Sort n-1 points with respect to the first point.
    // A point p1 comes before p2 in sorted ouput if p2
    // has larger polar angle (in counterclockwise
    // direction) than p1
    p0 = points[0];
    qsort(&points[1], n - 1, sizeof(Point), compare);

    // If two or more points make same angle with p0,
    // Remove all but the one that is farthest from p0
    // Remember that, in above sorting, our criteria was
    // to keep the farthest point at the end when more than
    // one points have same angle.
    int m = 1; // Initialize size of modified array
    for (int i = 1; i<n; i++)
    {
        // Keep removing i while angle of i and i+1 is same
        // with respect to p0
        while (i < n - 1 && orientation(p0, points[i],
            points[i + 1]) == 0)
            i++;


        points[m] = points[i];
        m++;  // Update size of modified array
    }

    // If modified array of points has less than 3 points,
    // convex hull is not possible
    if (m < 3) return;

    // Create an empty stack and push first three points
    // to it.
    stack<Point> S;
    S.push(points[0]);
    S.push(points[1]);
    S.push(points[2]);

    // Process remaining n-3 points
    for (int i = 3; i < m; i++)
    {
        // Keep removing top while the angle formed by
        // points next-to-top, top, and points[i] makes
        // a non-left turn
        while (orientation(nextToTop(S), S.top(), points[i]) != 2)
            S.pop();
        S.push(points[i]);
    }

    // Now stack has the output points, print contents of stack
    while (!S.empty())
    {
        Point p = S.top();
        //여기서 vector에 넣든, print를 하면 됨. 컨벡스헐 나오는곳
        S.pop();
    }
}
```

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 <set>
#include <map>
#include <functional>
#include <cmath>
#include <cstring>

using namespace std;

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

typedef long long LL;
typedef long double LD;
typedef pair<LL,LL> pll;

int main(void)
{
    ios::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);

    return 0;
}
```