TEAM Hosu-Galbi

- Shortest Path

1. Dijkstra Algorithm

2. Bellman-Ford Algorithm

- String

1. KMP Algorithm

2. Trie

3. Aho-Corasik

- Data Structure

1. Segment Tree

2. Segment Tree Lazy

3. Disjoint set

- Flow

1. Bipartite Matching

2. Network Flow

- Geometry

1. Convex Hull Algorithm

2. FFT Algorithm

- C++ Reference

- Shortest Path

1. Dijkstra Algorithm

```
typedef pair<int, int> p;
int v, e, k;
//start, end, cost
vector<p> adj[MAX];

vector<int> dijkstra(int src) {
    priority_queue<p> pq;
    pq.push({ 0, src });

    vector<int> dist(v + 1, INF);
    dist[src] = 0;

    while (!pq.empty()) {
        int cost = -pq.top().first;
        int here = pq.top().second;
        pq.pop();
        if (cost > dist[here]) continue;
        for (auto& n : adj[here]) {
            int there = n.first;
            int nextDist = cost + n.second;
            if (dist[there] > nextDist) {
                dist[there] = nextDist;
                pq.push({ -nextDist, there });
            }
        }
    }
    return dist;
}
```

2. Bellman-Ford Algorithm – 간선 음수

```cpp
typedef pair<ll, ll> p;
int N, M;
vector<p> graph[MAX_V];
ll res[MAX_V];
int main() {
    cin >> N >> M;
    int a, b, c;
    for (int i = 0; i < M; ++i) {
        cin >> a >> b >> c;
        graph[a].push_back({ b, c });
    }
    for (int i = 1; i <= N; ++i)
        res[i] = INF;
    res[1] = 0;
    bool cycle = false;
    for (int i = 1; i <= N; ++i) {
        for (int j = 1; j <= N; ++j) {
            for (int k = 0; k < graph[j].size(); ++k) {
                ll nextV = graph[j][k].first;
                ll nextCost = graph[j][k].second;
                if ((res[j] != INF) && res[nextV] > res[j] + nextCost) {
                    res[nextV] = res[j] + nextCost;
                    if (i == N) { cycle = true; break; }
                }
            }
        }
    }
    if (cycle) cout << -1 << endl;
    else {
        for (int i = 2; i <= N; ++i) {
            if (res[i] == INF) cout << -1 << endl;
            else cout << res[i] << endl;}
    }
    return 0;
}
```

- String

KMP – 텍스트에서 검색어 하나를 찾아낼 때 사용 (1:1)

```cpp
string T, P;
vector<int> getPi(const string& N) {
    int M = N.size();
    vector<int> pi(M, 0);
    int begin = 1, matched = 0;
    while (begin + matched < M) {
        if (N[begin + matched] == N[matched]) {
            matched++;
            pi[begin + matched - 1] = matched;
        }
        else
            if (matched == 0)
                begin++;
            else {
                begin += matched - pi[matched - 1];
                matched = pi[matched - 1];
            }
    }
    return pi;
}
vector<int> kmpSearch(const string& H, const string& N) {
    int n = H.size(), m = N.size();

    vector<int> result;
    vector<int> pi = getPi(N);

    int matched = 0;
    for (int i = 0; i < n; ++i) {
        while (matched > 0 && H[i] != N[matched])
            matched = pi[matched - 1];
        if (H[i] == N[matched]) {
            matched++;
            if (matched == m) {
                result.push_back(i - m + 2);
                matched = pi[matched - 1];
            }
        }
    }
    return result;
}
```

Trie – 접두사 저장 트리

```cpp
struct Trie {
    Trie* go[26];
    bool output;
    int branch; // 가지, 즉 자식 노드의 개수
    int words; // 현재 노드 서브트리에 있는 단어의 개수
    Trie() : output(false), branch(0), words(0)
    { fill(go, go + 26, nullptr); }
    ~Trie() {
        for (int i = 0; i < 26; i++)
            if (go[i]) delete go[i];}
    // 트라이에 단어를 삽입하는 함수
    void insert(char* str) {// char W[81]; root->insert(W);
        if (*str == 'W0') {
            branch++;
            output = true;
        }
        else {
            if (!go[*str - 'a']) {
                branch++;
                go[*str - 'a'] = new Trie;
            }
            words++;
            go[*str - 'a']->insert(str + 1);
        }}
    // 현재 노드에서 더 필요한 총 타이핑 횟수를 세는 재귀 함수
    long long cntKeystrokes(bool isRoot = false) {
        long long result = 0;
        // 맨 처음이거나, 현재로부터 도달가능한
        //단어가 2개 이상이면 속한 단어 개수만큼 타이핑++
        // 바꿔 말하면, 위 경우가 아니면 타이핑이 필요없다
        if (isRoot || branch > 1) result = words;
        // 각 자식들의 결과를 모두 더해서 반환
        for (int i = 0; i < 26; i++)
            if (go[i]) result += go[i]->cntKeystrokes();
        return result;
    }};
```

Aho-Corasik 텍스트에서 검색어 여러 개를 찾아낼 때 사용 (1:N)

```cpp
// 트라이 구조체
struct Trie {
    // 현재 노드에서 해당 문자를 받으면 가는 노드
    Trie* go[26];
    // 현재 노드에서 해당 문자의 go 목적지가 없을 때 가는 노드
    Trie* fail;
    // 현재 노드에 도달하면 찾는 문자열 집합: 이 문제에서는 존재성만 따지면 됨
    bool output;

    Trie() {
        fill(go, go + 26, nullptr);
        output = false;
    }
    ~Trie() {
        for (int i = 0; i < 26; i++)
            if (go[i]) delete go[i];
    }
    void insert(const char* key) {
        if (*key == '₩0') {
            output = true;
            return;
        }
        int next = *key - 'a';
        if (!go[next]) {
            go[next] = new Trie;
        }
        go[next]->insert(key + 1);
    }
};

int main() {
    int N, M;
    char str[10001];
    // 트라이에 S의 원소들을 모두 집어넣는다.
    Trie* root = new Trie;
    scanf("%d", &N);
    for (int i = 0; i < N; i++) {
        scanf("%s", str);
        root->insert(str);
    }

    // BFS를 통해 트라이 노드를 방문하며 fail 함수를 만든다.
    queue<Trie*> Q;
    root->fail = root;
    Q.push(root);
    while (!Q.empty()) {
        Trie* current = Q.front();
        Q.pop();

        // 26개의 input 각각에 대해 처리한다.
        for (int i = 0; i < 26; i++) {
            Trie* next = current->go[i];
            if (!next) continue;

            // 루트의 fail은 루트다.
            if (current == root) next->fail = root;
            else {
                Trie* dest = current->fail;
                // fail을 참조할 가장 가까운 조상을 찾아간다.
                while (dest != root && !dest->go[i])
                    dest = dest->fail;
                // fail(px) = go(fail(p), x)
                if (dest->go[i]) dest = dest->go[i];
                next->fail = dest;
            }
            // fail(x) = y일 때, output(y) ⊂ output(x)
            if (next->fail->output) next->output = true;

            // 큐에 다음 노드 push
            Q.push(next);
        }
    }

    // 각 문자열을 받아 문제를 푼다.
    scanf("%d", &M);
    for (int i = 0; i < M; i++) {
        scanf("%s", str);
        // 루트부터 시작
        Trie* current = root;
        bool result = false;
        for (int c = 0; str[c]; c++) {
            int next = str[c] - 'a';
            // 현재 노드에서 갈 수 없으면 fail을 계속 따라감
            while (current != root && !current->go[next])
                current = current->fail;
            // go 함수가 존재하면 이동. 루트면 이게 false일 수도 있다
            if (current->go[next])
                current = current->go[next];
            // 현재 노드에 output이 있으면 찾은 것이다.
            if (current->output) {
                result = true;
                break;
            }
        }
        // 결과 출력
        puts(result ? "YES" : "NO");
    }
    // 내 힘은 소중하기에 꼭 동적할당을 해제한다.
    delete root;
}
```

- Data Structure

Segment Tree – 구간 쿼리

```cpp
struct SegTree {
    int n;
    vector<int> rangeMin;
    SegTree(const vector<int>& array) {
        n = array.size();
        rangeMin.resize(n * 4);
        init(array, 0, n - 1, 1);
    }
    int init(const vector<int>& array, int left, int right, int node) {
        if (left == right)
            return rangeMin[node] = array[left];
        int mid = (left + right) / 2;
        int leftMin = init(array, left, mid, node * 2);
        int rightMin = init(array, mid + 1, right, node * 2 + 1);
        return rangeMin[node] = leftMin + rightMin;
    }
    int query(int left, int right, int node, int nodeLeft, int nodeRight) {
        if (right < nodeLeft || nodeRight < left) return 0;
        if (left <= nodeLeft && nodeRight <= right)
            return rangeMin[node];

        int mid = (nodeLeft + nodeRight) / 2;
        return query(left, right, node * 2, nodeLeft, mid) +
            query(left, right, node * 2 + 1, mid + 1, nodeRight);
    }
    int query(int left, int right) {
        return query(left, right, 1, 0, n - 1);}
    int update(int index, int newValue, int node, int nodeLeft, int nodeRight) {
        if (index < nodeLeft || nodeRight < index)
            return rangeMin[node];

        if (nodeLeft == nodeRight) return rangeMin[node] = newValue;
        int mid = (nodeLeft + nodeRight) / 2;
        return rangeMin[node] =
            update(index, newValue, node * 2, nodeLeft, mid) +
            update(index, newValue, node * 2 + 1, mid + 1, nodeRight);
    }
    int update(int index, int newValue) {
        return update(index, newValue, 1, 0, n - 1);}
};
```

Segment Tree Lazy – 특정 구간에 모든 값 연산

```cpp
const int ST_MAX = 1<<21;
struct SegTree{ //SegTree st;
    int start;
    long long arr[ST_MAX], lazy[ST_MAX];
    // 생성자
    SegTree(){
        start = ST_MAX/2;
        fill(arr, arr+ST_MAX, 0);
        fill(lazy, lazy+ST_MAX, 0);
    }
    // 리프 노드들의 값을 먼저 입력한 후 전체 세그먼트 트리 구축
    void construct(){
        for(int i=start-1; i>0; i--)
            arr[i] = arr[i*2] + arr[i*2+1];
    }
    // 구간 [ns, ne)인 node의 lazy 값을 propagate
    void propagate(int node, int ns, int ne){
        // lazy 값이 존재하면 실행
        if(lazy[node] != 0){
            // 리프 노드가 아니면 자식들에게 lazy 미룸
            if(node < start){
                lazy[node*2] += lazy[node];
                lazy[node*2+1] += lazy[node];
            }
            // 자신에 해당하는 만큼의 값을 더함
            arr[node] += lazy[node] * (ne - ns);
            lazy[node] = 0;
        }
    }
    // 구간 [s, e)에 k를 더하라
    void update(int s, int e, int k){ update(s, e, k, 1, 0, start); }
    void update(int s, int e, int k, int node, int ns, int ne){
        // 일단 propagate
        propagate(node, ns, ne);
        if(e <= ns || ne <= s) return;
        if(s <= ns && ne <= e){
            // 이 노드가 구간에 완전히 포함되면 lazy 부여 후 propagate
            lazy[node] += k;
            propagate(node, ns, ne);
            return;
        }
```

```
    void update(int s, int e, int k, int node, int ns, int ne){
        // 일단 propagate
        propagate(node, ns, ne);
        if(e <= ns || ne <= s) return;
        if(s <= ns && ne <= e){
            // 이 노드가 구간에 완전히 포함되면 lazy 부여 후 propagate
            lazy[node] += k;
            propagate(node, ns, ne);
            return;
        }
        int mid = (ns+ne)/2;
        update(s, e, k, node*2, ns, mid);
        update(s, e, k, node*2+1, mid, ne);
        // 마지막에 자식들의 값을 사용해 다시 자신의 값 갱신
        arr[node] = arr[node*2] + arr[node*2+1];
    }
    // 구간 [s, e)의 합을 구하라
    long long query(int s, int e){ return query(s, e, 1, 0, start); }
    long long query(int s, int e, int node, int ns, int ne){
        propagate(node, ns, ne);
        if(e <= ns || ne <= s) return 0;
        if(s <= ns && ne <= e) return arr[node];
        int mid = (ns+ne)/2;
        return query(s, e, node*2, ns, mid) + query(s, e, node*2+1, mid, ne);
    }
};
```

Disjoint set – Union Find

```
struct DisjointSet {
    vector<int> parent, rank;
    DisjointSet(int n) :parent(n), rank(n + 1) {
        for (int i = 0; i < n; ++i)
            parent[i] = i;
    }
    int find(int u) {
        if (u == parent[u]) return u;
        return parent[u] = find(parent[u]);
    }
    void merge(int u, int v) {
        u = find(u); v = find(v);
        if (u == v) return;
        if (rank[u] > rank[v]) swap(u, v);
        parent[u] = v;
        if (rank[u] == rank[v]) ++rank[v];
    }
    bool isSame(int b, int c) {
        if (find(b) == find(c))
            return true;
        else return false;
    }
};
```

- Flow

Bipartite Matching – 2 Groups Network

```
//A, B 그룹의 크기
int n, m;
//연결 여부
vector<int> adj[MAX];
//rooms[b] = a
// B그룹의 b와 A그룹의 a 매칭
int rooms[MAX];
bool visited[MAX];
bool dfs(int a) {
    if (visited[a]) return false;
    visited[a] = true;
    for (auto node : adj[a]) {
        if (rooms[node] == 0 || dfs(rooms[node])) {
            rooms[node] = a;
            return true;
        }
    }
    return false;
}

int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; ++i) {
        int val; cin >> val;
        while (val--) {
            int temp;  cin >> temp;
            adj[i].push_back(temp);
        }
    }
    int ans = 0;
    for (int i = 1; i <= n; ++i) {
        memset(visited, false, sizeof(visited));
        if (dfs(i)) ++ans;
    }//최대 매칭
    cout << ans << endl;
    return 0;
}
```

Network Flow – Only 용량

```
int n;
vector<int> adj[MAX];       //간선
int capacity[MAX][MAX];     //i -> j 가는 용량
int flow[MAX][MAX];         //i -> j 가는 유량
int parent[MAX];            //지나온 경로 기억

int networkFlow(int source, int sink) {
    memset(flow, 0, sizeof(flow));
    int totalFlow = 0;
    while (true) {
        memset(parent, -1, sizeof(parent));
        queue<int> q;
        parent[source] = source;
        q.push(source);
        while (!q.empty() && parent[sink] == -1) {
            int curr = q.front();
            q.pop();
            for (int next : adj[curr]) {
                if (capacity[curr][next] - flow[curr][next] > 0 && parent[next] == -1) {
                    q.push(next);
                    parent[next] = curr;
                    if (next == sink) break;
                }
            }
        }
        if (parent[sink] == -1) break;
        //경로중에 최소 유량 찾기
        int amount = INF;
        //parent[i]의 경로를 기억하므로
        for (int p = sink; p != source; p = parent[p])
            amount = min(capacity[parent[p]][p] - flow[parent[p]][p], amount);
        for (int p = sink; p != source; p = parent[p]) {
            flow[parent[p]][p] += amount;
            flow[p][parent[p]] -= amount;
        }
        totalFlow += amount;
    }
    return totalFlow;
}
```

- Geometry

Convex-Hull

```
struct Point {
    int x, y; // 실제 위치
    int p, q; // 기준점으로부터의 상대 위치
    Point() : Point(0, 0, 1, 0) {}
    Point(int x1, int y1) : Point(x1, y1, 1, 0) {}
    Point(int x1, int y1, int p1, int q1) : x(x1), y(y1), p(p1), q(q1) {}
    // p, q 값을 기준으로 정렬하기 위한 관계연산자
    bool operator <(const Point& O) {
        if (1LL * q * O.p != 1LL * p * O.q) return 1LL * q * O.p < 1LL * p * O.q;
        if (y != O.y) return y < O.y;
        return x < O.x;
    }
};
// 벡터 AB와 벡터 AC의 CW/CCW
long long ccw(const Point& A, const Point& B, const Point& C) {
    return 1LL * (B.x - A.x) * (C.y - A.y) - 1LL * (B.y - A.y) * (C.x - A.x);
}

int main() {
    int N;
    scanf("%d", &N);
    Point p[MAX];
    for (int i = 0; i < N; i++) {
        int x, y;
        scanf("%d %d", &x, &y);
        p[i] = Point(x, y);
    }
    // 점들을 y좌표 -> x좌표 순으로 정렬: 0번 점이 제일 아래 제일 왼쪽
    sort(p, p + N);
    for (int i = 1; i < N; i++) {
        p[i].p = p[i].x - p[0].x;
        p[i].q = p[i].y - p[0].y;
    }
    // 0번을 제외한 점들을 반시계 방향으로 정렬
    sort(p + 1, p + N);
    stack<int> S;
    // 스택에 처음 2개의 점을 넣음
    S.push(0); S.push(1);
    int next = 2;
    // 모든 점을 훑음
    while (next < N) {
        // 스택에 2개 이상의 점이 남아있는 한...
        while (S.size() >= 2) {
            int first, second;
            first = S.top();
            S.pop();
            second = S.top();
            // 스택 최상단 점 2개와 다음 점의 관계가 CCW일 때까지 스택 pop
            if (ccw(p[second], p[first], p[next]) > 0) {
                S.push(first);
                break;
            }
        }
        // 다음 점을 스택에 넣음
        S.push(next++);
    }
    // 이제 스택에 컨벡스 헐 정점들이 순서대로 쌓여 있음
    printf("%d\n", S.size());
}
```

FFT – Convolution

```
const double PI = acos(-1); // PI 값을 지정해 놓음
typedef complex<double> cpx;
void FFT(vector<cpx>& f, cpx w) {
    int n = f.size();
    if (n == 1) return; //base case
    vector<cpx> even(n >> 1), odd(n >> 1);
    for (int i = 0; i < n; i++) {
        if (i & 1) odd[i >> 1] = f[i];
        else even[i >> 1] = 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;
    }
}
/*
input : a => A's Coefficient, b => B's Coefficient
output : A * B
*/
vector<cpx> mul(vector<cpx> a, vector<cpx> b) {
    int n = 1;
    while (n <= a.size() || n <= b.size()) n <<= 1;
    n <<= 1;
    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); //result is integer
        c[i] = cpx(round(c[i].real()), round(c[i].imag()));
    }
    return c;
}
```

```
signed main() {
    ios_base::sync_with_stdio(0); cin.tie(0);
    int n; cin >> n;
    vector<int> A(n + n), B(n);
    for (int i = 0; i < n; i++) cin >> A[i];
    for (int i = n - 1; i >= 0; i--) cin >> B[i];
    for (int i = 0; i < n; i++) A[i + n] = A[i];
    vector<cpx> a, b;
    for (auto i : A) a.push_back(cpx(i, 0));
    for (auto i : B) b.push_back(cpx(i, 0));
    vector<cpx> c = mul(a, b);
    long long ans = 0;
    for (int i = 0; i < c.size(); i++) {
        ans = max<long long>(ans, round(c[i].real()));
    }
    cout << ans;
}
```

Vector

```
const double PI = 2.0 * acos(0.0);
const double EPSILON = 1e-9;
struct vector2
{
    double x, y;
    explicit vector2(double x = 0, double y = 0) : x(x), y(y) {}

    bool operator == (const vector2& rhs) const{return x == rhs.x && y == rhs.y;}
    bool operator < (const vector2& rhs) const{return x != rhs.x ? x < rhs.x : y < rhs.y;}
    vector2 operator + (const vector2& rhs) const{return vector2(x + rhs.x, y + rhs.y);}
    vector2 operator - (const vector2& rhs) const {return vector2(x - rhs.x, y - rhs.y);}
    vector2 operator * (double rhs) const {return vector2(x * rhs, y * rhs);}

    double norm() const  { return hypot(x, y); }
    vector2 normalize() const   {return vector2(x / norm(), y / norm());}
    double polar() const    {return fmod(atan2(y, x) + 2 * PI, 2 * PI); }
    double dot(const vector2& rhs) const{ return x * rhs.x + y * rhs.y;}
    double cross(const vector2& rhs) const {return x * rhs.y - y * rhs.x;}
    vector2 project(const vector2& rhs) const{
        vector2 r = rhs.normalize();
        return r * r.dot(*this);
    }
};
```

C++ Reference)

```
#include <bits/stdc++.h>
#define endl "\n"
#define ll long long
#define INF 987654321
#define MAX 51
#define MOD 1000000000
#define int ll
using namespace std;
typedef pair<int, int> p;

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);
}
```

Vector Erase

```
arr.erase(unique(arr.begin(), arr.end()), arr.end());
```

Heap

```
priority_queue<int> maxheap;
priority_queue<int, vector<int>, greater<int>> minheap;
```

Lower_bound, Upper_bound

```
vector<int> arr = { 1,2,3,4,5,6,6,6 };  //ans: 5
cout << lower_bound(arr.begin(), arr.end(), 6) - arr.begin();
```