# WhereIsWoongJae - Team Note



## C++ Templete

```cpp
#include <bits/stdc++.h>
#define all(v) v.begin(), v.end()
using namespace std;
typedef long long ll;
typedef pair<int,int> pii; typedef pair<ll,ll> pll;
typedef tuple<int,int,int> tiii;
const int INF = 0x3f3f3f3f;  // 1061109567
// const ll INF = 0x3f3f3f3f3f3f3f3f;
// const int MOD = 1'000'000'007;
/* ---------------------------------------------------- */
#define MAX

int main(){
    ios_base::sync_with_stdio(0); cin.tie(0);
    return 0;
}
/* ---------------------------------------------------- */
// Random number Generator
#include <bits/stdc++.h>
using namespace std;

random_device rd;
mt19937 gen(rd());
uniform_int_distribution<int> dtb(0, 9999);
int sa = dtb(gen), sb = dtb(gen), a=0, b=0;
dtb(gen)
```

## Python Templete

```python
import sys ; input = lambda : sys.stdin.readline().strip()
from itertools import product
def INT(n) : return int(input()) + n
def MAP(n) : return map(lambda x: int(x)+n, input().split())
def LIST(n) : return list(MAP(n))
def RANGE2(a,b) : return list(product(range(a),range(b)))
def RANGE22(a1,a2,b1,b2) : return list(product(range(a1,a2),range(b1,b2)))
```

[ 재귀 제한 해제 ] sys.setrecursionlimit(10**6)

## Union-Find (Disjoint Set)

```
find(x: int) = root       //  x의 root는 누구인가?
union(x: int, y: int)     //  x와 y의 집합을 합친다
```

```
/* 초기화 */
int root[MAX_SIZE];
int rank[MAX_SIZE];  // 트리의 높이를 저장할 배열
for (int i = 0; i < MAX_SIZE; i++) {
  root[i] = i;
  rank[i] = 0;  // 트리의 높이 초기화
}

/* find(x): 재귀 이용 */
int find(int x) {
  if (root[x] == x) {
      return x;
  } else {  // find 하면서 만난 모든 값의 부모 노드를 root로 만든다.
      return root[x] = find(root[x]);
  }
}

/* union1(x, y): union-by-rank 최적화 */
void union(int x, int y){
   x = find(x);
   y = find(y);

   if(x == y)  // 두 값의 root가 같으면(이미 같은 트리) 합치지 않는다.
     return;
   // 항상 높이가 더 낮은 트리를 높이가 높은 트리(root) 밑에 넣는다.
   if(rank[x] < rank[y]) {
     root[x] = y;  // x의 root를 y로 변경
   } else {
     root[y] = x;  // y의 root를 x로 변경

     if(rank[x] == rank[y])
       rank[x]++;  // 만약 높이가 같다면 합친 후 (x의 높이 + 1)
   }
}
```

```
/* union2(x, y): 두 원소가 속한 트리의 전체 노드의 수 구하기 */
int nodeCount[MAX_SIZE];
for (int i = 0; i < MAX_SIZE; i++)
   nodeCount[i] = 1;

int union2(int x, int y){
   x = find(x);
   y = find(y);

   // 두 값의 root가 같지 않으면
   if(x != y) {
       root[y] = x;  // y의 root를 x로 변경
       nodeCount[x] += nodeCount[y];  // x의 node 수 += y의 node 수
       nodeCount[y] = 1;  // x에 붙은 y의 node 수는 1로 초기화
   }
   return nodeCount[x];  // 가장 root의 node 수 반환
}
```

## Segment Tree

여러 개의 데이터가 존재할 때, 특정 구간의 연산(합 등)을 구할 수 있다.

```
#include <bits/stdc++.h>
#define VAL(x) (x?x->v:iden)
const int SZ = 1<<17;
using namespace std;
using F = int(*)(int,int);

struct Node {
    int v;
    Node *l, *r;
    Node(int v=0, Node *l = NULL, Node *r = NULL)
    : v{v}, l{l}, r{r} {}
};
```

```
struct Seg {
    Node *root;
    const F fn;
    const int iden;
    Seg(F fn, int iden)
    : fn(fn), iden{iden} {
        root = new Node(iden);
    }
    void update(int l, int r) {}
    int query(int l, int r) {}

    private:
    void update(Node *node, int s, int e, int pos, int v) {
        if(s == e) {
            node->v = v; return;
        }
        int m = s + e >> 1;
        Node *&l = node->l, *&r = node->r;
        if(pos<=m) {
            if(!l) l = new Node(iden);
            update(l, s, m, pos, v);
        } else {
            if(!r) r = new Node(iden);
            update(r, m+1, e, pos, v);
        }
        node->v = fn(VAL(l), VAL(r));
    }
    int query(Node *node, int s, int e, int l, int r) {
        if(!node || r<s || l<e) return iden;
        if(l<=s && e<=r) return VAL(node);
        int m = s + e >> 1;
        return fn(query(node->l, s, m, l, r), query(node->r, m+1,
e, l, r));
    }
};
```

[ Python ]

```
# [ 세그먼트 트리 (합) ]
class SegmentTree :
    def __init__(self,arr) :
        self.STarr = []
        length = 1
        while len(arr) > length : length *= 2
        self.length = length

    def init(self,arr) :
        self.STarr = [0] * self.length * 2
        for i in range(len(arr)) :
            self.STarr[self.length+i] = arr[i]

        for i in range(self.length-1,0,-1) :
            self.STarr[i] = self.STarr[2*i] + self.STarr[2*i+1]

    def change(self,idx,value) :
        realLen = self.length + idx
        self.STarr[realLen] = value
        while realLen > 1 :
            realLen //= 2
            self.STarr[realLen] = self.STarr[2*realLen] + self.STarr[2*realLen+1]

    def intervalCalc(self,idxStart,idxEnd) :
        a = self.length + idxStart
        b = self.length + idxEnd + 1
        sml , smr = 0 , 0
        while a<b :
            if a&1 == 1 : sml += self.STarr[a] ; a += 1
            if b&1 == 1 : smr += self.STarr[b-1] ; b -= 1
            a //= 2
            b //= 2
        return sml+smr

    def print(self) :
        print(self.STarr)
```

```
# [ 세그먼트 트리 (합,곱,max,min) ]
class SegmentTree :
    def __init__(self,arr,calcType) :
        length = 1
        while len(arr) > length : length *= 2
        self.STarr = []
        self.length = length
        self.calcType = calcType

    def init(self,arr) :
        if   self.calcType == "+"   : self.STarr = [0] * self.length * 2
        elif self.calcType == "*"   : self.STarr = [1] * self.length * 2
        elif self.calcType == "min" : self.STarr = [10**10] * self.length * 2
        elif self.calcType == "max" : self.STarr = [-10**10] * self.length * 2

        for i in range(len(arr)) :
            self.STarr[self.length+i] = arr[i]

        for i in range(self.length-1,0,-1) :
            if   self.calcType == "+"   : self.STarr[i] = self.STarr[2*i] +
self.STarr[2*i+1]
            elif self.calcType == "*"   : self.STarr[i] = self.STarr[2*i] *
self.STarr[2*i+1]
            elif self.calcType == "min" : self.STarr[i] = min( self.STarr[2*i] ,
self.STarr[2*i+1] )
            elif self.calcType == "max" : self.STarr[i] = max( self.STarr[2*i] ,
self.STarr[2*i+1] )

    def update(self,idx,value) :
        STarrIdx = self.length + idx
        self.STarr[STarrIdx] = value
        while STarrIdx > 1 :
            STarrIdx //= 2
            if   self.calcType == "+"   : self.STarr[STarrIdx] =
self.STarr[2*STarrIdx] + self.STarr[2*STarrIdx+1]
            elif self.calcType == "*"   : self.STarr[STarrIdx] =
self.STarr[2*STarrIdx] * self.STarr[2*STarrIdx+1]
            elif self.calcType == "min" : self.STarr[STarrIdx] =
min( self.STarr[2*STarrIdx] , self.STarr[2*STarrIdx+1] )
            elif self.calcType == "max" : self.STarr[STarrIdx] =
max( self.STarr[2*STarrIdx] , self.STarr[2*STarrIdx+1] )

    def intervalCalc(self,idxStart,idxEnd) :
        leftIdx  = self.length + idxStart
        rightIdx = self.length + idxEnd + 1

        if   self.calcType == "+"   : leftCalc , rightCalc = 0 , 0
        elif self.calcType == "*"   : leftCalc , rightCalc = 1 , 1
        elif self.calcType == "min" : leftCalc , rightCalc = 10**10 , 10**10
        elif self.calcType == "max" : leftCalc , rightCalc = -10*10 , -10**10

        while leftIdx < rightIdx :
            if leftIdx  & 1 == 1 :
                if   self.calcType == "+"   : leftCalc += self.STarr[leftIdx]
                elif self.calcType == "*"   : leftCalc *= self.STarr[leftIdx]
                elif self.calcType == "min" : leftCalc = min( leftCalc ,
self.STarr[leftIdx] )
                elif self.calcType == "max" : leftCalc = max( leftCalc ,
self.STarr[leftIdx] )
                leftIdx  += 1
            if rightIdx & 1 == 1 :
                if   self.calcType == "+"   : leftCalc += self.STarr[rightIdx-1]
                elif self.calcType == "*"   : leftCalc *= self.STarr[rightIdx-1]
                elif self.calcType == "min" : leftCalc = min( leftCalc ,
self.STarr[rightIdx-1] )
                elif self.calcType == "max" : leftCalc = max( leftCalc ,
self.STarr[rightIdx-1] )
                rightIdx -= 1
            leftIdx  //= 2
            rightIdx //= 2

        if   self.calcType == "+"   : return leftCalc + rightCalc
        elif self.calcType == "*"   : return leftCalc * rightCalc
        elif self.calcType == "min" : return min(leftCalc,rightCalc)
        elif self.calcType == "max" : return max(leftCalc,rightCalc)
```

Knapsack =>

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| (0, 0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| W=6, V=13 | 0 | 0 | 0 | 0 | 0 | 0 | 13 | 13 |
| W=4, V=8 | 0 | 0 | 0 | 0 | 8 | 8 | 13 | 13 |
| W=3, V=6 | 0 | 0 | 0 | 6 | 8 | 8 | 13 | ?? |
| W=5, V=12 | | | | | | | | |

j(7)-W(3)

## Knapsack Problem

W : 무게 / v : 가치 / dp[j] : 총 넣은 무게가 j 이하일 때 최대 가치 합

```
for (int i=0; i<N; i++){
    int weight = W[i];
    // 이거 탑다운으로 안하면 같은 물건을 여러번 넣는 경우도 계산됨
    for (int j=K; j>=W[i]; j--){
        dp[j] = max(dp[j], dp[j-weight] + V[i]);
    }
}
```

## LIS (가장 긴 증가하는 부분 수열) with $O(n \log n)$

[5, 1, 6, 2, 7, 3, 8] -> [1, 2, 3, 8] 또는 [5, 6, 7, 8] 뽑는 알고리즘

```
void backtraceLIS (vector<int> &v, vector<int> &before, int idx,
int num) {
    // 역추적으로 출력해야하는경우
    if (idx == -1) return;
    if (before[idx] == num) {
        backtraceLIS(v, before, idx-1, num-1);
        cout << v[idx] << ' ';
    }
    else backtraceLIS(v, before, idx-1, num);
}
int LIS(vector<int> &v){
    int size = v.size();
    vector<int> ans;
    // vector<int> before(size);
    for(int i=0; i<v.size(); i++){
        // B의 마지막 값보다 크다면 B에 push
        if (ans.size()==0 || v[i] > ans.back()){
            ans.push_back(v[i]);
            // before[i]=ans.size()-1;
        }
        // 아니라면 들어갈 자리 탐색
        else{
            int idx = lower_bound(ans.begin(), ans.end(), v[i]) -
ans.begin();
            ans[idx] = v[i];
            // before[i] = idx;
        }
    }
    int len = ans.size();
    // cout << len << '\n';
    // backtraceLIS(v, before, size-1, len-1);
    return len;
}
```

## LCS (Longest Common Substring or Subsequence, 가장 긴 공통 부분수열)

ABCDEF , GBCDFE => BCDF 뽑아내는 알고리즘

```
# da : 현재 a의 원소의 값을 key, 그 때의 인덱스를 value로 갖는
딕셔너리
da = dict().fromkeys([i for i in range(n + 1)], -1)
# da : 현재 b의 원소의 값을 key, 그 때의 인덱스를 value로 갖는
딕셔너리
db = dict().fromkeys([i for i in range(n + 1)], -1)

for i in range(n):
    da[a[i]] = i
    db[b[i]] = i
```

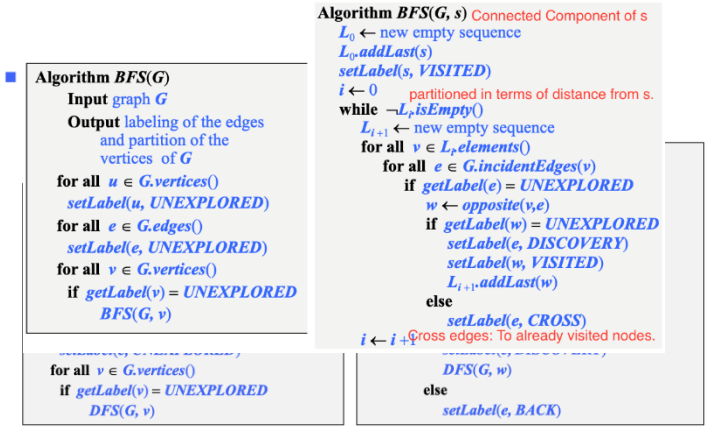
```python
# arr : A를 기준으로 하는 B 배열의 인덱스 배열. 그림 3의 오른쪽 위 배열 그림과 같다
arr = []
for i in range(n):
    arr.append(db[a[i]])

# lis를 구한다 - O(NlogN)
lis = []
for i in range(n):
    if not lis:
        lis.append(arr[i])
    else:
        if arr[i] > lis[-1]:
            lis.append(arr[i])
        else:
            # idx : lower_bound의 인덱스
            idx = bisect.bisect_left(lis, arr[i])
            lis[idx] = arr[i]

# 정답 출력
print(len(lis))
```

## LCA (Lowest Common Ancestor, 최소 공통 조상)

가장 가까이에 있는 공통 조상을 알아내는 알고리즘 / **인터넷 긁어옴**

```cpp
#include <iostream>
#include <cstring>
#include <vector>
using namespace std;
vector<int> v[100001];
int parent[100001][17]; // log2(100000) = 16.61 -> 17
int depth[100001];
int N,M;
void DFS(int cur) {
    for (int i = 0; i < v[cur].size(); i++) {
        int next = v[cur][i];
        if (depth[next] == -1) {
            parent[next][0] = cur;
            depth[next] = depth[cur] + 1;
            DFS(next);
        }
    }
}

void connect() {
    for (int k = 1; k < 17; k++) {
        for (int cur = 1; cur <= N; cur++)
            parent[cur][k] = parent[parent[cur][k - 1]][k-1];
    }
}
int LCA(int a, int b) {
    // 깊이가 더 큰게 a로 오게
    if (depth[a] < depth[b]) {
        int temp = a;
        a = b;
        b = temp;
    }

    int diff = depth[a] - depth[b];

    for (int i = 0; diff != 0; i++) {
        if (diff % 2 == 1)
            a = parent[a][i];
        diff /= 2;
    }

    // 깊이가 같아졌으니 뛰자
    if (a != b) {
       for (int i = 16; i >= 0; i--) {
          if (parent[a][i] != -1 && parent[a][i] != parent[b][i]) {
             a = parent[a][i];
             b = parent[b][i];
          }
       }
       a = parent[a][0];
    }
    return a;
}
```

## BFS / DFS




### Dijkstra (정점 1개 ▸ 모든 정점) $O((V+E)\log V)$

음수 간선 + 음수 사이클 존재 -> 오류

```
vector<vector<pii>> graph;

vector<int> dijkstra(int start){
    vector<int> dist(V + 1, INF);  // INF 로 선언과 동시에 초기화
    priority_queue<pii> pq;  // 우선순위 큐 선언

    dist[start] = 0;
    pq.push({0, start});  // 우선순위 큐에 넣기 (힙큐 -> 순서 중요!!)

    while (!pq.empty()) {
        int cur_dist = -pq.top().first;
        int cur_node = pq.top().second;
        pq.pop();

        // 현재 테이블보다 가중치가 큰 튜플이면 무시
        if (cur_dist > dist[cur_node]) continue;

        for (auto& edge : graph[cur_node]) {
            int nxt_node = edge.first;
            int nxt_dist = edge.second + cur_dist;

            if (nxt_dist < dist[nxt_node]) {  // 유망하면
                dist[nxt_node] = nxt_dist;
                pq.push({-nxt_dist, nxt_node});
            }
        }
    }

    return dist;  // start node ▶ each node 최단거리 담은 벡터 리턴
}
```

### Bellman Ford (정점 1개 ▸ 모든 정점) $O(VE)$

다익보다 느리지만, 음수 간선 가능

```
vector<tuple<int,int,int>> edges;  // {a,b,c} : a -> b 의 비용이 c

pair<bool, vector<ll>> bellmanFord(int start){
    vector<ll> dist (v + 1, INF);
    dist[start] = 0;

    // 전체 v-1 번의 라운드(round)를 반복
    for (int i=0; i<v; i++){
        for (auto edge : edges){
            auto [cur_node, nxt_node, cost] = edge;

            // 현재 간선을 거쳐서 다른 노드로 이동하는 거리가 더 짧은 경우
            if (dist[cur_node] != INF && dist[nxt_node] >
dist[cur_node] + cost){
                dist[nxt_node] = dist[cur_node] + cost;

                // v(노드개수)만큼 반복했을때도 갱신된다면
                if (i == v-1) return {true, dist}; // 음수 순환 존재
            }
        }
    }
    return {false, dist};  // 음수 순환 없고, 최단거리 리스트 반환
}
```

## Floyd Warshall (모든 정점 ▸ 모든 정점) $O(V^3)$

음의 가중치를 가지는 그래프에서도 사용 가능하다!

```
    int graph[n+1][n+1];
    fill(&graph[0][0], &graph[n+1][0], INF);  // INF 로 모두 초기화

    // 자기 자신 거리 0 초기화
    for (int i=1; i<=n; i++){
        graph[i][i] = 0;
    }
    for (int i=0; i<m; i++){
        int a, b, c;
        cin >> a >> b >> c;
        // 시작노드와 끝 노드가 같더라도 간선이 다를 경우 최저값 갱신
        graph[a][b] = min(graph[a][b], c);
    }

    for (int k = 1; k <= n; k++) {
        for (int a = 1; a <= n; a++) {
            for (int b = 1; b <= n; b++) {
                graph[a][b] = min(graph[a][b], graph[a][k] +
graph[k][b]);
            }
        }
    }
```

## MST (Minimum Spanning Tree, 최소 신장 트리)

간선이 많을 때 - 정점 선택 **Prim's Algorithm** with heapq : $O((V+E)\log V)$

간선이 적을 때 - 간선 선택 **Kruskal's Algorithm** : $O(E\log V)$

```
// A C++ program for Prim's Minimum
// Spanning Tree (MST) algorithm. The program is
// for adjacency matrix representation of the graph

#include <bits/stdc++.h>
using namespace std;

// Number of vertices in the graph
#define V 5

// A utility function to find the vertex with
// minimum key value, from the set of vertices
// not yet included in MST
int minKey(int key[], bool mstSet[])
{
    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}

// A utility function to print the
// constructed MST stored in parent[]
void printMST(int parent[], int graph[V][V])
{
    cout << "Edge \tWeight\n";
    for (int i = 1; i < V; i++)
        cout << parent[i] << " - " << i << " \t"
             << graph[i][parent[i]] << " \n";
}

// Function to construct and print MST for
// a graph represented using adjacency
// matrix representation
void primMST(int graph[V][V])
{
    // Array to store constructed MST
    int parent[V];

    // Key values used to pick minimum weight edge in cut
    int key[V];
```

```
    // To represent set of vertices included in MST
    bool mstSet[V];

    // Initialize all keys as INFINITE
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    // Always include first 1st vertex in MST.
    // Make key 0 so that this vertex is picked as first
    // vertex.
    key[0] = 0;

    // First node is always root of MST
    parent[0] = -1;

    // The MST will have V vertices
    for (int count = 0; count < V - 1; count++) {

        // Pick the minimum key vertex from the
        // set of vertices not yet included in MST
        int u = minKey(key, mstSet);

        // Add the picked vertex to the MST Set
        mstSet[u] = true;

        // Update key value and parent index of
        // the adjacent vertices of the picked vertex.
        // Consider only those vertices which are not
        // yet included in MST
        for (int v = 0; v < V; v++)

            // graph[u][v] is non zero only for adjacent
            // vertices of m mstSet[v] is false for vertices
            // not yet included in MST Update the key only
            // if graph[u][v] is smaller than key[v]
            if (graph[u][v] && mstSet[v] == false
                && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    // Print the constructed MST
    printMST(parent, graph);
}

// Driver's code
int main()
{
    int graph[V][V] = { { 0, 2, 0, 6, 0 },
                        { 2, 0, 3, 8, 5 },
                        { 0, 3, 0, 0, 7 },
                        { 6, 8, 0, 0, 9 },
                        { 0, 5, 7, 9, 0 } };

    // Print the solution
    primMST(graph);

    return 0;
}

// This code is contributed by rathbhupendra
```

---

```
int V, E;
vector<tuple<int,int,int>> v;  // {가중치, 부모, 자식}
int parent[MAX];

int findParent(int x){
    if(parent[x] == x) return x;
    return parent[x] = findParent(parent[x]);
}

void unionParent(int a, int b){
    a = findParent(a);
    b = findParent(b);
    parent[b] = a;
}

int MST(){
    sort(v.begin(), v.end());
    for (int i = 1; i <= V; i++) {
```

```
        parent[i] = i;
    }

    int ans = 0, cnt = 0;
    for (auto edge : v){
        auto [cost, next, now] = edge;

        if (findParent(now) == findParent(next)) continue; // 이미 연결되어있다면 PASS

        unionParent(now, next);
        ans += cost;

        if (++cnt == V-1) break;  // V-1 개이면 바로 끝
    }

    return ans;
}
```

## Topological Sorting (위상정렬)

```
int inDegree[MAX];  // 진입차수 (자신으로 오는 노드의 개수)
vector<int> graph[MAX];  // graph[i] = {i 뒤에 오는 것들}

vector<int> topology_sort() {
    queue<int> q;
    vector<int> ans;
    for (int i = 1; i <= N; i++) {
        if (inDegree[i] == 0)
            q.push(i); // 진입차수가 0 인 노드 push
    }
    for (int rep = 0; rep < N; rep++) {
        if (q.empty()) return;

        int now = q.front();
        ans.push_back(now);
        q.pop();

        for (int next : graph[now]) {
            inDegree[next]--;
            if (inDegree[next] == 0)
                q.push(next);
        }
    }
    return ans;
}
```

## SCC (Strong Connected Component, 강한 연결요소) + 2-SAT

From 인터넷

```
const int MAXN = 100;
vector<int> graph[MAXN];
int up[MAXN], visit[MAXN], vtime;
vector<int> stk;
int scc_idx[MAXN], scc_cnt;

void dfs(int nod) {
    up[nod] = visit[nod] = ++vtime;
    stk.push_back(nod);
    for (int next : graph[nod]) {
        if (visit[next] == 0) {
            dfs(next);
            up[nod] = min(up[nod], up[next]);
        }
        else if (scc_idx[next] == 0)
            up[nod] = min(up[nod], visit[next]);
    }
    if (up[nod] == visit[nod]) {
        ++scc_cnt;
        int t;
        do {
            t = stk.back();
            stk.pop_back();
            scc_idx[t] = scc_cnt;
        } while (!stk.empty() && t != nod);
```

```
    }
}

// find SCCs in given directed graph
// O(V+E)
// the order of scc_idx constitutes a reverse topological sort
void get_scc() {
    vtime = 0;
    memset(visit, 0, sizeof(visit));
    scc_cnt = 0;
    memset(scc_idx, 0, sizeof(scc_idx));
    for (int i = 0; i < n; ++i)
        if (visit[i] == 0) dfs(i);
}
```

---

$$f = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3)$$

$$x_1 \to x_2 \ , \ \neg x_2 \to \neg x_1 \ , \ x_2 \to x_3 \ , \ \neg x_3 \to \neg x_2$$

이렇게 SCC 돌리고 나서 $x_i$ 와 $\neg x_1$ 가 한 SCC에 같이 있으면 틀린 것
그렇지 않다면 f를 만족시킬 수 있는 답이 무조건 존재함 -> 2-SAT

---

---

1. 단순 SCC 찾는 문제
2. SCC 응용
3. 2-SAT

1. 단순 SCC 찾는 문제

```
// boj.kr/2150
// http://boj.kr/6694264c637848ceb8a6224bbff2e15a

#include <bits/stdc++.h>
#define ALL(x) x.begin(), x.end()
using namespace std;
struct Kosaraju {
        const int n;
        vector<vector<int>> graph, rgraph, sccs;
        vector<bool> visited;
        vector<int> stk;
        Kosaraju(int n)
        : n(n)  {
                graph.resize(n+1);
                rgraph.resize(n+1);
                visited.resize(n+1);
        }
        void addEdge(int a, int b) {
                graph[a].push_back(b);
                rgraph[b].push_back(a);
        }
        void dfs1(int u) {
                visited[u] = true;
                for(int v: graph[u]) if(!visited[v]) dfs1(v);
                stk.push_back(u);
        }
        void dfs2(int u, vector<int> &vec) {
                visited[u] = false;
                vec.push_back(u);
                for(int v: rgraph[u]) if(visited[v]) dfs2(v, vec);
        }
        void main() {
                for(int i=1;i<=n;++i) if(!visited[i]) dfs1(i);
                while(!stk.empty()) {
                        int top = stk.back(); stk.erase(stk.end()-
1);
                        if(!visited[top]) continue;
                        vector<int> scc;
                        dfs2(top, scc);
                        sort(ALL(scc));
                        sccs.push_back(scc);
                }
        // 문제에서 그냥 이런 순서대로 정렬하라고 함. 이 부분은 꼭
필요하진 않음.
                sort(ALL(sccs), [](const vector<int> &a, const
vector<int> &b) -> bool{
                                return a[0] < b[0];
                                });
```

```
        }
};
int main() {
        cin.tie(NULL); ios_base::sync_with_stdio(false);
        int n, m; cin >> n >> m;
        Kosaraju kos(n);
        while(m--) {
                int a, b; cin >> a >> b;
                kos.addEdge(a, b);
        }
        kos.main();
        cout << kos.sccs.size() << '\n';
        for(const vector<int> &scc: kos.sccs) {
                for(int a: scc) cout << a << ' ';
                cout << "-1\n";
        }
        return 0;
}
```

2. SCC 응용

```
// SCC 응용 - boj.kr/4013
// http://boj.kr/42bbeda008154509bf51b029188ee579

#include <bits/stdc++.h>
using namespace std;

struct Kosaraju {
        const int n;
        vector<vector<int>> graph, rgraph, sccs, scc_graph;
        vector<bool> visited, eatable, scc_eatable;
        vector<int> stk, scc_map, money, ATM;
        Kosaraju(int n): n(n) {
                graph.resize(n+1);
                rgraph.resize(n+1);
                visited.resize(n+1);
                scc_map.resize(n+1);
                eatable.resize(n+1);
                money.resize(n+1);
        }
        void addEdge(int a, int b)
        {
                graph[a].push_back(b);
                rgraph[b].push_back(a);
        }
        void dfs1(int u) {
                visited[u] = true;
                for(int v: graph[u]) if(!visited[v]) dfs1(v);
                stk.push_back(u);
        }
        void dfs2(int u, vector<int> &scc) {
                scc.push_back(u);
                visited[u] = false;
                for(int v: rgraph[u]) if(visited[v]) dfs2(v, scc);
        }
        void debug1() {
                cout << "sccs: " << sccs.size() << '\n';
                for(int i=0,m=sccs.size();i<m;++i) {
                        cout << i << ": ";
                        for(int u: sccs[i]) cout << u << ' ';
                        cout << '\n';
                }
        }
        void debug2() {
                cout << "scc_graph:\n";
                for(int i=0,m=scc_graph.size();i<m;++i) {
                        cout << i << ": ";
                        for(int j: scc_graph[i]) cout << j << ' ';
                        cout << '\n';
                }
                cout << "ATM:\n";
                for(int i=0,m=ATM.size();i<m;++i) {
                        cout << i << ": " << ATM[i] << '\n';
                }
        }
        // condensation graph를 만드는 코드인듯.
        void make_scc_graph() {
                for(int i=1;i<=n;++i) if(!visited[i]) dfs1(i);
                while(!stk.empty()) {
```

```
                        int top = stk.back(); stk.erase(stk.end()-
1);
                        if(!visited[top]) continue;
                        vector<int> scc;
                        dfs2(top, scc);
                        for(int u: scc) scc_map[u] = sccs.size();
                        sccs.push_back(scc);
                }
                int m = sccs.size();
                scc_graph.resize(m);
                scc_eatable.resize(m);
                ATM.resize(m);
                for(int i=0;i<m;++i) {
                        vector<int> vis;
                        vis.push_back(i);
                        visited[i] = true;
                        for(int u: sccs[i]) {
                                ATM[i] += money[u];
                                if(scc_eatable[i] != eatable[u])
scc_eatable[i] = true;
                                for(int v: graph[u])
if(!visited[scc_map[v]]) {

scc_graph[i].push_back(scc_map[v]);
                                        vis.push_back(scc_map[v]);
                                        visited[scc_map[v]] = true;
                                }
                        }
                        for(int v: vis) visited[v] = false;
                }
        }
        int solve(int start) {
                make_scc_graph();
                //debug1();
                //debug2();
                int m = sccs.size();
                vector<int> dp(m);
                for(int i=m-1;~i;--i) {
                        int a = 0;
                        for(int j: scc_graph[i]) if(scc_eatable[j])
{
                                scc_eatable[i] = true;
                                a = max(a, dp[j]);
                        }
                        dp[i] = scc_eatable[i] ? ATM[i] + a : 0;
                }
                return dp[scc_map[start]];
        }
};

int main() {
        cin.tie(NULL); ios_base::sync_with_stdio(false);
        int m, n; cin >> m >> n;
        Kosaraju kos(m);
        while(n--) {
                int a, b; cin >> a >> b;
                kos.addEdge(a,b);
        }
        for(int i=1;i<=m;++i) {
                cin >> kos.money[i];
        }
        int s, p; cin >> s >> p;
        while(p--) {
                int a;
                cin >> a;
                kos.eatable[a] = true;
        }
        cout << kos.solve(s);

        return 0;
}
```

3. 2-SAT

```
// 2-SAT
// boj.kr/11281
// http://boj.kr/fdb1c3d1d332448481ba623a08fb6b60

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

```cpp
using namespace std;

// 각 변수(literal)는 1 부터 n 까지의 숫자로 표현됨.
// -1~-n 은 1~n 의 negation 을 나타냄.
struct SAT2 {
    const int n; // number of boolean variables. 변수의 개수
    int m; // 절의 개수
    vector<vector<int>> g, rg, sccs;
    vector<int> sccMap, stk;
    vector<bool> vis;
    SAT2(int n)
    : n(n), m(0) {
        int num = (n+1)<<1;
        g.resize(num);
        rg.resize(num);
        sccMap.resize(num);
        vis.resize(num);
    }
        // vertex 의 값. 우리가 아는 리터럴을 여기서 쓰이는 값으로
변환함.
        // e.g. vtx(2) = |2|*2 + 0 = 4, vtx(-1) = |-1|*2 + 1 = 3
    int vtx(int literal) {
        return abs(literal)<<1|(literal<0);
    }
        // 여기서 쓰이는 literal 의 representation 을 우리가 아는
literal 로 변환.
        // vtx 의 역함수라고 보면 됨.
    int lit(int vertex) {
        return (vertex>>1) * (vertex&1?-1:1);
    }
        // 2-SAT 절(clause) 추가.
    void addClause(int a, int b) {
        int na = vtx(-a), nb = vtx(-b);
        a = vtx(a), b = vtx(b);
        g[na].push_back(b);
        g[nb].push_back(a);
        rg[a].push_back(nb);
        rg[b].push_back(na);
    }
        // 첫 번째 DFS 로 directed graph 탐색.
    void dfs1(int u) {
        vis[u] = true;
        for(int v: g[u]) if(!vis[v]) dfs1(v);
        stk.push_back(u);
    }
    void dfs2(int u) {
        vis[u] = false;
        sccs[m].push_back(u);
        sccMap[u] = m;
        for(int v: rg[u]) if(vis[v]) dfs2(v);
    }
        // strongly connected components 를 만든다.
    void makeSccs() {
        int num = (n+1)<<1;
        for(int i=2;i<num;++i) if(!vis[i]) dfs1(i);
        for(auto i=stk.rbegin(),e=stk.rend();i!=e;++i) if(vis[*i])
{
            sccs.push_back(vector<int>());
            dfs2(*i);
            ++m;
        }
    }
        // check the satisfiability of the problem.
    bool check() {
        for(int i=1;i<=n;++i) {
            if(sccMap[vtx(i)] == sccMap[vtx(-i)]) return false;
        }
        return true;
    }
        // 주어진 식을 true 로 만들 수 있는 경우, 그걸 true 로 만드는
xi 의 값을 x1 부터 xn 까지 출력. 아닌 경우, 빈 vector 를 반환.
    vector<bool> solve() {
        makeSccs();
        if(!check()) return vector<bool>(); // 빈 vector 반환.
        vector<bool> ret(n+1);
        vector<bool> rret(n+1);
        for(int i=0, l;i<m;++i) {
            l = lit(sccs[i].front());
            if(l>0 && ret[l] || l<0 && rret[-l]) continue;
```

```
            for(int x: sccs[i]) {
                x = lit(x);
                if(x>0) rret[x] = true;
                else ret[-x] = true;
            }
        }
        return ret;
    }
};

int main() {
    cin.tie(NULL); ios_base::sync_with_stdio(false);
    int m, n; cin >> m >> n;
    SAT2 sat(m);
    while(n--) {
        int a, b; cin >> a >> b;
        sat.addClause(a, b);
    }
    vector<bool> ans = sat.solve();
    if(ans.empty()) cout << 0;
    else {
        cout << 1 << '\n';
        for(int i=1;i<=m;++i) cout << ans[i] << ' ';
    }

    return 0;
}
```

## Flow Network (최대 유량) with Dinic

그래프에서 두 정점 사이에 얼마나 많은 유량을 보낼 수 있는가?

```
const int vertexSZ = 1000;  // in out 분할이라면 2 배
const int SZ = vertexSZ+5, bias = vertexSZ/2;
int SRC = vertexSZ+1, SINK = vertexSZ+2;


struct NetworkFlow{  // use Dinic
    using FlowType = int;

    struct Edge{ int to, rev; FlowType cap; };
    vector<Edge> graph[SZ];
    int level[SZ], work[SZ];

    // 마지막 인자를 안쓰면 유방향, cap 과 같게 쓰면 무방향(양쪽 cap
같음)
    void addEdge(int _from, int _to, FlowType _cap, FlowType
_caprev = 0){
        graph[_from].push_back({_to, (int)graph[_to].size(),
_cap});
        graph[_to].push_back({_from, (int)graph[_from].size()-1,
_caprev});
    }

    void initGraph(){ // for Test Case
        for (int i=0; i<SZ; i++) graph[i].clear();
    }

    bool BFS(int S, int T){  // make level graph
        memset(level, 0, sizeof(level));
        queue<int> q; q.push(S); level[S] = 1;
        while(!q.empty()){
            int now = q.front(); q.pop();
            for(const auto &next : graph[now]){
                if(!level[next.to] && next.cap) q.push(next.to),
level[next.to] = level[now] + 1;
            }
        }
        return level[T];
    }
    FlowType DFS(int now, int T, FlowType flow){  // find Blocking
Flow
        if(now == T) return flow;
        for(; work[now] < (int)graph[now].size(); work[now]++){
            auto &next = graph[now][work[now]];
            if(level[next.to] != level[now] + 1 || !next.cap)
continue;
            FlowType ret = DFS(next.to, T, min(flow, next.cap));
```

```
            if(!ret) continue;
            next.cap -= ret;
            graph[next.to][next.rev].cap += ret;
            return ret;
        }
        return 0;
    }
    FlowType maxFlow(int S = SRC, int T = SINK){
        FlowType ret = 0, minFlow;
        while(BFS(S, T)){
            memset(work, 0, sizeof(work));
            while((minFlow = DFS(S, T, INF))) ret += minFlow;
        }
        return ret;
    }
} nf;
```

## Bipartite Matching (이분 매칭) $O(VE)$

결과값 : 얼마나 많은 쌍이 나왔는지 최대 쌍 카운트

```
int N, M;
vector<int> works[MAX];
bool visited[MAX];  // dfs 했는가?
int human[MAX];

bool dfs(int cur){
    if (visited[cur]) return false;
    visited[cur] = true;
    for(int work : works[cur]){
        // work 에 대한 사람이 아직 없다면 그 일 매칭
        // dfs 결과 그 사람이 다른곳으로 배정이 되었다면 매칭완료
        if (human[work] == 0 || dfs(human[work])) {
            human[work] = cur;
            return true;
        }
    }
    return false;
}

int bipartite_match() {
    int ret = 0;
    for (int i=1; i<=N; i++) {
        memset(visited, 0, sizeof(visited));
        if (dfs(i)) ret++;
    }
    return ret;
}
```

## MCMF (Min-Cost Max-Flow,최소 비용 최대 유량)

간선에 용량 뿐만 아니라, 또다른 가중치인 비용(cost)이 존재할 때

```
const int vertexSZ = 1000;  // in out 분할이라면 2 배
const int SZ = vertexSZ+10, bias = vertexSZ/2;
int SRC = vertexSZ+1, SINK = vertexSZ+2;

struct MCMF{  // use Dinic

    // using CostType = int; const CostType EPS = 0;  // cost :
int
    using CostType = double; const CostType EPS = 1e-8;  // cost :
double
    using FlowType = int;

    struct Edge{ int to, rev; FlowType cap; CostType cost; };
    vector<Edge> graph[SZ];
    void addEdge(int _from, int _to, FlowType _cap, CostType
_cost){
        graph[_from].push_back({_to, (int)graph[_to].size(), _cap,
_cost, });
        graph[_to].push_back({_from, (int)graph[_from].size()-1, 0,
-_cost});
    }

    bool inQ[SZ];
    CostType costs[SZ]; //dijkstra
    bool spfa(int S, int T) {
        memset(costs, 0x3f, sizeof(costs));  // int
```

```
        // fill(costs, costs+SZ, INF);  // double
        memset(inQ, false, sizeof(inQ));
        queue<int> q;
        q.push(S);
        inQ[S] = true;
        costs[S] = 0;
        while (!q.empty()) {
            int now = q.front();
            q.pop();
            inQ[now] = false;
            for (auto edge: graph[now]) {
                if (edge.cap > EPS && costs[edge.to] > costs[now] +
edge.cost + EPS) {
                    costs[edge.to] = costs[now] + edge.cost;
                    if (!inQ[edge.to]) inQ[edge.to] = true,
q.push(edge.to);
                }
            }
        }
        return costs[T] < INF;  // costs[SINK]가 갱신되었다면 true
    }
    bool chk[SZ];
    int work[SZ];
    FlowType dfs(int now, int T, FlowType flow){
        chk[now] = true;
        if(now == T) return flow;
        for(; work[now] < (int)graph[now].size(); work[now]++){
            auto &nxt = graph[now][work[now]];
            if(!chk[nxt.to] && costs[nxt.to] == costs[now] +
nxt.cost && nxt.cap){
                FlowType ret = dfs(nxt.to, T, min(flow, nxt.cap));
                if(ret){
                    nxt.cap -= ret; graph[nxt.to][nxt.rev].cap +=
ret;
                    return ret;
                }
            }
        }
        return 0;
    }
    pair<CostType, FlowType> run(int S = SRC, int T = SINK) {
        CostType cost = 0;
        FlowType flow = 0;
        while (spfa(S, T)) {
            memset(chk, 0, sizeof chk);
            memset(work, 0, sizeof work);
            FlowType now = 0;
            while (true) {
                now = dfs(S, T, INF);
                if (now==0) break;
                cost += costs[T] * now;
                flow += now;
                memset(chk, 0, sizeof chk);
            }
        }
        return {cost, flow};
    }

    void initGraph(){ // 테스트케이스를 위한 그래프 초기화
        for (int i=0; i<SZ; i++)
            graph[i].clear();
    }
} mcmf;
```

## 선분 교차 판정

```
#define x first
#define y second
using dot = pair<int, int>;

dot operator + (const dot &a, const dot &b){ return { a.x + b.x,
a.y + b.y }; }
dot operator - (const dot &a, const dot &b){ return { a.x - b.x,
a.y - b.y }; }
double operator * (const dot &a, const dot &b){ return a.x*a.x +
a.y*a.y; }
double operator / (const dot &a, const dot &b){ return a.x*b.y -
b.x*a.y; }
dot operator * (const dot &a, double b){ return { a.x * b, a.y *
b }; }
dot operator / (const dot &a, double b){ return { a.x / b, a.y /
b }; }

int ccw(const dot &A, const dot &B, const dot &C){
    auto res = (B - A) / (C - B);
    return (res > 0) - (res < 0);
}
ll ccwLine(dot A1, dot A2, dot B1, dot B2){  // 선분 A 와 선분 B ccw
    return (A2.x-A1.x)*(B2.y-B1.y) - (A2.y-A1.y)*(B2.x-B1.x);
}

double dist(dot a, dot b){
    double dx = a.x - b.x;
    double dy = a.y - b.y;
    return dx*dx + dy*dy;
}
ll dist2(dot A, dot B){  // 거리의 제곱 연산
    ll dx = A.x-B.x;
    ll dy = A.y-B.y;
    return dx*dx + dy*dy;
}


// 선분이 교차하는지
bool Cross(dot s1, dot e1, dot s2, dot e2){
    auto cw1 = ccw(s1, e1, s2) * ccw(s1, e1, e2);
    auto cw2 = ccw(s2, e2, s1) * ccw(s2, e2, e1);
    if(cw1 == 0 && cw2 == 0){
        if(e1 < s1) swap(s1, e1);
        if(e2 < s2) swap(s2, e2);
        return !(e1 < s2 || e2 < s1);
    }
    return cw1 <= 0 && cw2 <= 0;
}

// 어디서 교차하는지 Call by reference 로 알려줌
bool Cross(dot s1, dot e1, dot s2, dot e2, dot &res){
    if(!Cross(s1, e1, s2, e2)) return false;
    double det = (e1 - s1) / (e2 - s2);
    if(abs(det) < 1e-10) return false;  // 부동소수로 인한 0 이상 값
방지
    res = s1 + (e1 - s1) * ((s2 - s1) / (e2 - s2) / det);
    return true;
}
```

## Convex Hull + Rotating Calipers

( 선분 교차 판정 코드 적은 다음에 작성할 것. )

```
struct ConvexHull{
    vector<dot> dots;
    void build(vector<dot> &a){
        swap(a[0], *min_element(a.begin(), a.end()));
        sort(a.begin()+1, a.end(), [&](const dot &s, const dot &e){
            double cw = ccw(a[0], s, e);
            if(cw != 0) return cw > 0;
            return dist(a[0], s) < dist(a[0], e);
```

```
        });
        dots.clear();
        for(auto i : a){
            while(dots.size() >= 2 && ccw(dots[dots.size()-2],
dots.back(), i) <= 0) dots.pop_back();
            dots.push_back(i);
        }
        // assert 함수는 size가 2 이하라면 다각형이 안만들어지므로
오류 발생
        // assert(dots.size() >= 3 && "Can't make ConvexHull");
    }
    // 다각형 내부에 점이 있는지
    bool contain(const dot &p) const {
        int i = lower_bound(dots.begin()+1, dots.end(), p,
[&](const dot &a, const dot &b){
            double cw = ccw(dots[0], a, b);
            if(cw) return cw > 0;
            return dist(dots[0], a) < dist(dots[0], b);
        }) - dots.begin();
        if(i == dots.size()) return 0;
        if(i == 1) return ccw(dots[0], p, dots[1]) == 0 && dots[0]
<= p && p <= dots[1];
        int t1 = ccw(dots[0], p, dots[i]) * ccw(dots[0], p, dots[i-
1]);
        int t2 = ccw(dots[i], dots[i-1], dots[0]) * ccw(dots[i],
dots[i-1], p);
        if(t1 == -1 && t2 == -1) return 0;
        return ccw(dots[0], p, dots[i-1]) != 0;
    }
    // 다각형 내부에 다각형이 있는지
    bool contain(const ConvexHull &h) const {
        for(const auto &i : h.dots) if(!contain(i)) return false;
        return true;
    }
    // 넓이를 리턴하는 함수
    double area() const {
        double ret = 0;
        for(int i=0; i<dots.size(); i++){
            int j = i + 1; if(j == dots.size()) j = 0;
            ret += dots[i].x * dots[j].y;
            ret -= dots[j].x * dots[i].y;
        }
        return abs(ret) * 0.5;
    }
    // 선분을 리턴하는 함수
    vector<pair<dot, dot>> makeEdges() const {
        vector<pair<dot, dot>> ret;
        for(int i=0; i+1<dots.size(); i++)
ret.emplace_back(dots[i], dots[i+1]);
        if(dots.size() > 1) ret.emplace_back(dots.back(), dots[0]);
        return ret;
    }
    // 회전하는 캘리퍼스
    double rotatingCallipers(){
        ll ans = 0;  // 불필요한 연산 방지를 위해 일단 제곱으로 저장 후
나중에 루트 씌우기
        int C = 1;
        int dotSZ = dots.size();
        for (int A=0; A<dotSZ; A++){
            // 양수(반시계 방향)라면 C를 높이기
            while(ccwLine(dots[A], dots[(A+1)%dotSZ], dots[C],
dots[(C+1)%dotSZ])>0){
                C = (C+1)%dotSZ;
            }
            // 시계방향이 된다면 더이상 C를 높여봤자 작아짐
            ans = max(ans, dist2(dots[A], dots[C]));  // 최대값 갱신
(이후에 A가 바뀜)
        }
        return sqrt(ans);
    }
};
```

---

```
[ Python ]

def CCW( P1 , P2 , P3 ) :
    P1P2 = [ P2[0]-P1[0] , P2[1] - P1[1] ]
    P2P3 = [ P3[0]-P2[0] , P3[1] - P2[1] ]
```

```
    return P1P2[0] * P2P3[1] - P1P2[1] * P2P3[0]



lowerHull , upperHull = [] , []
def ConvexHull( P ) :
    global lowerHull , upperHull
    for p in P :
        if len(lowerHull) < 2 : lowerHull.append(p)
        else :
            while len(lowerHull)>=2 and CCW( lowerHull[-2] ,
lowerHull[-1] , p ) <= 0 : lowerHull.pop()
            lowerHull.append(p)
    for p in P[::-1] :
        if len(upperHull) < 2 : upperHull.append(p)
        else :
            while len(upperHull)>=2 and CCW( upperHull[-2] ,
upperHull[-1] , p ) <= 0 : upperHull.pop()
            upperHull.append(p)

def Distance(P1,P2) :
    return ( (P1[0]-P2[0])**2 + (P1[1]-P2[1])**2 ) ** 0.5

def RotatingCallipers(hull) :
    L = len(hull)
    if L == 2 : return [0,1]
    pos1 , pos2 = 0 , 1
    maxPos = [0,1]
    while pos1 < L :
        V1 = [ hull[(pos1+1)%L][0]-hull[pos1%L][0] ,
hull[(pos1+1)%L][1]-hull[pos1%L][1] ]
        V2 = [ hull[(pos2+1)%L][0]-hull[pos2%L][0] ,
hull[(pos2+1)%L][1]-hull[pos2%L][1] ]
        ccwValue = V1[0] * V2[1] - V1[1] * V2[0]
        if ccwValue >= 0 : pos2 += 1
        else : pos1 += 1
        if Distance(hull[maxPos[0]%L],hull[maxPos[1]%L]) <
Distance(hull[pos1%L],hull[pos2%L]) :
            maxPos = [ pos1 , pos2 ]
    maxPos[0] %= L
    maxPos[1] %= L
    return maxPos
```

## KMP (Knuth-Morris-Pratt Algorithm)

문자열 검색 알고리즘 / from 인터넷

```
#include <cstdio>
#include <vector>
using namespace std;
int const NMAX = 1000001;
char a[NMAX], b[NMAX];
int t[NMAX];

int main() {
    gets(a); gets(b);

    int pos = 2, cnd = 0;
    while (b[pos - 1]) {
        if (b[pos - 1] == b[cnd])
            t[pos++] = ++cnd;
        else if (cnd)
            cnd = t[cnd];
        else
            t[pos++] = 0;
    }

    int m = 0, i = 0;
    vector<int> ans;
    while (a[m + i]) {
        if (a[m + i] == b[i]) {
            if (!b[++i]) ans.push_back(m);
        } else if (i) {
            m += i - t[i];
            i = t[i];
        } else {
            ++m;
        }
    }

    printf("%lu\n", ans.size());
    for (int i : ans)
        printf("%d ", i + 1);
```

```
    return 0;
}
```

---

```
// Failure Function을 단순하게 사용하는 KMP 부분문자열찾기 알고리즘
// boj.kr/1786
// http://boj.kr/3889538a0d1f412a808245550dbb8fd8

#include <bits/stdc++.h>
using namespace std;

vector<int> failure(const string &s) {
    int n = s.size();
    vector<int> f(n);
    for(int i=1,j=0;i<n;++i) {
        while(j>0 && s[j] != s[i]) j = f[j-1];
        if(s[j] == s[i]) f[i] = ++j;
    }
    return f;
}

int main() {
    cin.tie(NULL); ios_base::sync_with_stdio(false);
    string t, s;
    getline(cin, t);
    getline(cin, s);
    vector<int> f = failure(s);
    vector<int> ans;

    for(int i=0,n=t.size(),m=s.size(),j=0;i<n;++i) {
        while(j>0 && s[j] != t[i]) j = f[j-1];
        if(s[j] == t[i]) ++j;
        if(j == m) {
            ans.push_back(i+1-m+1);
            j = f[m-1];
        }
    }

    cout << ans.size() << '\n';
    for(int i: ans) cout << i << ' ';

    return 0;
}
```

## Trie

```
#include <bits/stdc++.h>
using namespace std;
struct Node {
    bool terminal;
    map<char, Node*> children;
    Node()
        : terminal(false) {}
};

class Trie {
public:
    Node *node;
    Trie()
        : node(new Node()) {}
    void insert(const string &key) {
        Node *tgt = node;
        int n = key.size();
        for(int i=0;i<n;++i) {
            if(tgt->children[key[i]] == NULL) tgt->children[key[i]]
= new Node();
            tgt = tgt->children[key[i]];
        }
    tgt->terminal = true;
    }
    bool find(const string &key) {
        Node *tgt = node;
        int n = key.size();
        for(int i=0;i<n;++i) {
            tgt = tgt->children[key[i]];
            if(tgt == NULL) return false;
        }
```

```
        return tgt->terminal;
    }
};

int main() {
    int m, n;
    cin.tie(NULL); ios_base::sync_with_stdio(false);
    cin >> n >> m;
    Trie trie;
    for(int i=0;i<n;++i) {
        string s;
        cin >> s;
        trie.insert(s);
    }
    int cnt = 0;
    for(int i=0;i<m;++i) {
        string s;
        cin >> s;
        if(trie.find(s)) ++cnt;
    }
    cout << cnt;
    return 0;
}
```

---

[ Python ]

T = Trie()
T.insert('app')
T.search(Str) -> Bool
T.startsWith(Str) -> None 혹은 List[Str]

```
class Node(object) :
    def __init__(self,key,data=None) :
        self.key , self.data , self.children = key , data , {}

class Trie :
    def __init__(self) :
        self.head = Node(None)
    def insert(self,string) :
        node = self.head
        for char in string :
            if char not in node.children : node.children[char] =
Node(char)
            node = node.children[char]
        node.data = string

    def search(self,string) :
        node = self.head
        for char in string :
            if char in node.children : node = node.children[char]
            else : return False
        if node.data : return True
        else : return False

    def startsWith(self,prefix) :
        preNode = self.head
        words = []
        for p in prefix :
            if p in preNode.children : preNode =
preNode.children[p]
            else : return None
        preNode = [preNode]
        newNode = []
        while True :
            for node in preNode :
                if node.data : words.append(node.data)
                newNode.extend( list(node.children.values()) )
            if len(newNode) != 0 :
                preNode = newNode
```

```
            newNode = []
        else : break
    return words
```

## Sqrt Decomposition + Mo's Algorithm

Mo's - 업데이트가 없는 구간 쿼리들을 빠르게 처리하는 알고리즘
쿼리 : 구간의 합, 최댓값, 최솟값 등
아이디어 : 앞선 쿼리로 인해 계산된 값을 최대한 활용하자!

```cpp
int sqrtN;
struct Query{
    int idx, s, e;
    bool operator < (const Query &x) const {
        if(s/sqrtN != x.s/sqrtN) return s/sqrtN < x.s/sqrtN;
        return e < x.e;
    }
};
int res;
void Plus(int x){}  // 구현 필요
void Minus(int x){}  // 구현 필요

// main 함수에서 쿼리 받기부터
    vector<Query> Q(M);
    for(int i=0; i<M; ++i){
        cin >> Q[i].s >> Q[i].e;
        Q[i].idx = i;
    }
    sort(all(Q));  // 쿼리 정렬

    // 이전 쿼리의 결과를 이용해 계산해나가기
    int s=0, e=0;
    for(int i=0;i<M;++i){
        while(s<Q[i].s) Minus(A[s++]);
        while(s>Q[i].s) Plus(A[--s]);
        while(e<Q[i].e) Plus(A[++e]);
        while(e>Q[i].e) Minus(A[e--]);
        ans[Q[i].idx] = res;
    }
```

## Computing Combination.

If P >= 10억, and n and r <= 1천만, then you may evaluate the fac and inv until 1000만.

```
const ll BIG = 1000000007LL;

struct Lucas{ // init : O(P), query : O(log P)
    const size_t P;
    vector<ll> fac, inv;
    ll Pow(ll a, ll b) {
        ll res=1;
        b%=P-1;
        while(b<0) b += P-1;
        for(; b; b>>=1, a=a*a%P) if(b & 1) res = res * a % P;
        return res;
    }
    Lucas(size_t P, size_t N) : P(P), fac(N), inv(N) {
        fac[0]=1; for(int i=1;i<N;i++) fac[i]=fac[i-1]*i%P;
        inv[N-1] = Pow(fac[N-1], P-2);
        for(int i=N-2;~i;i--) inv[i]= inv[i+1] * (i+1) % P;
    }
    ll small(ll n, ll r) const {
        return r<=n ? fac[n] * inv[r] % P * inv[n-r] % P : 0LL;
    }
    ll calc(ll n, ll r) const {
        if(n<r || n<0 || r<0) return 0LL;
        if(!n || !r || n==r) return 1LL;
        return small(n%P, r%P) * calc(n/P, r/P) % P;
    }
} L(BIG, 151000);
```

## FFT (Python)

A = [1,3,2] , B = [2,1] , C = []

FFTMultiply(A,B,C)

=> C = [2,7,7,2,0,0,0,0]

```
import math
def FFT(a,invert) :
    n = len(a) # n = 2^k
    j = 0
    for i in range(1,n) :
        bit = n >> 1
        while j >= bit : j -= bit ; bit >>= 1
        j += bit
        if i < j : a[i] , a[j] = a[j] , a[i]
    L = 2
    while L <= n :
        ang = ( 2 * math.pi / L ) * ( -1 if invert else 1 )
        wL = complex( math.cos(ang) , math.sin(ang) )
        for i in range(0,n,L) :
            w = complex(1,0)
            for j in range( 0 , L//2 ) :
                u = a[i+j] ; v = a[i+j+L//2] * w
                a[ i+j       ] = u + v
                a[ i+j + L//2 ] = u - v
                w *= wL
        L <<= 1
    if invert :
        for i in range(n) : a[i] /= n

def FFTMultiply(a,b,res) :
    n = 1
    while n < max( len(a) , len(b) ) : n <<= 1
    n *= 2
    a += [0] * ( n - len(a) )
    b += [0] * ( n - len(b) )
    FFT(a,False) ; FFT(b,False)
    for i in range(n) : a[i] *= b[i]
    FFT(a,True)
    for i in range(n) : a[i] = a[i].real
    res += [0] * ( n - len(res) )
    for i in range(n) :
        if a[i] > 0 : p = 0.5
        else : p = -0.5
        res[i] = int( a[i].real + p )
```

## Hopcroft-Karp

```cpp
// Hopcroft-Karp
// Bipartite Matching in O(E sqrt(V))
// boj.kr/3736
// http://boj.kr/6b3c32c0913c454a9f1d3020c258e7dc

#include <bits/stdc++.h>
using namespace std;
vector<int> graph[10101];
int n, match[10101], inv_match[10101], dist[10101];

void input() {
    for(int i=0;i<10101;++i) graph[i].clear();
    for(int i=0;i<n;++i) {
        int a, nb;
        scanf("%d: (%d)", &a, &nb);
        for(int j=0;j<nb;++j) {
            int b;
            scanf("%d", &b);
            graph[a].push_back(b-n);
        }
    }
}

bool bfs() {
    queue<int> q;
    for(int a=0;a<n;++a) {
        if(match[a] == n) {
            dist[a] = 0;
            q.push(a);
        } else {
            dist[a] = INT_MAX;
        }
    }
    dist[n] = INT_MAX;
    while(!q.empty()) {
        int a = q.front(); q.pop();
        if(dist[a] < dist[n]) {
            for(int b: graph[a]) {
                if(dist[inv_match[b]] == INT_MAX) {
                    dist[inv_match[b]] = dist[a] + 1;
                    q.push(inv_match[b]);
                }
            }
        }
    }
    return dist[n] != INT_MAX;
}

bool dfs(int a) {
    if(a!=n) {
        for(int b: graph[a]) {
            if(dist[inv_match[b]] == dist[a] + 1) {
                if(dfs(inv_match[b])) {
                    inv_match[b] = a;
                    match[a] = b;
                    return true;
                }
            }
        }
        dist[a] = INT_MAX;
        return false;
    }
    return true;
}

int hopcroftKarp() {
    // initialize all to -1: set each byte to 0xFF.
    // memset(match, -1, sizeof(match));
    // memset(inv_match, -1, sizeof(inv_match));
    fill(match, match+n, n);
    fill(inv_match, inv_match+n, n);

    int matching = 0;
    while(bfs()) {
        for(int a=0;a<n;++a) {
            if(match[a] == n && dfs(a)) {
                matching++;
            }
        }
    }
    return matching;
}

int main() {
    // cin.tie(NULL); ios_base::sync_with_stdio(false);
    while(scanf("%d", &n)>0) {
        input();
        printf("%d\n", hopcroftKarp());
    }

    return 0;
}
```