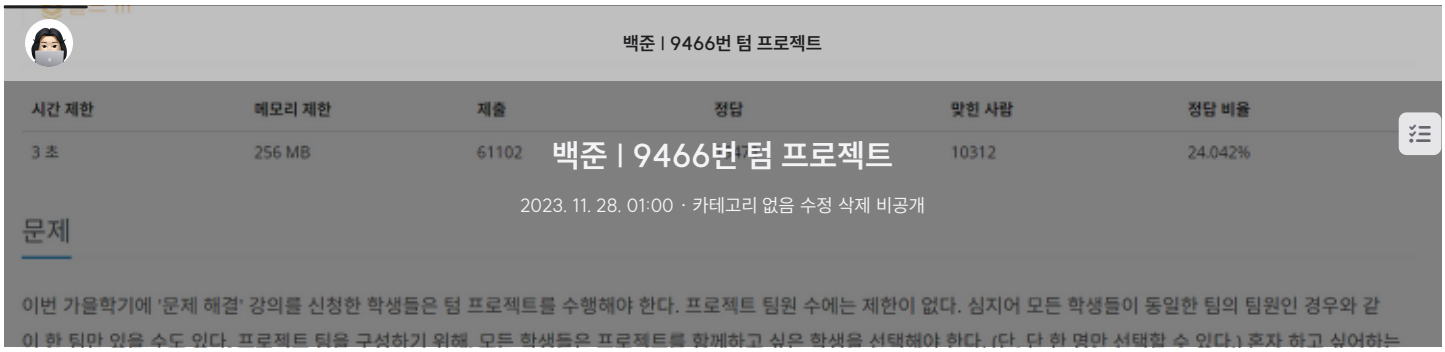





9466번 제출 맞힌 사람 숏코딩 재채점 결과 채점 현황 내 제출 질문 게시판

## 텀 프로젝트

실제 다국어

한국어

골드 III

| 시간 제한 | 메모리 제한 | 제출 | 정답 | 맞힌 사람 | 정답 비율 |
|---|---|---|---|---|---|
| 3 초 | 256 MB | 61102 | 15471 | 10312 | 24.042% |

### 문제

이번 가을학기에 '문제 해결' 강의를 신청한 학생들은 텀 프로젝트를 수행해야 한다. 프로젝트 팀원 수에는 제한이 없다. 심지어 모든 학생들이 동일한 팀의 팀원인 경우와 같이 한 팀만 있을 수도 있다. 프로젝트 팀을 구성하기 위해, 모든 학생들은 프로젝트를 함께하고 싶은 학생을 선택해야 한다. (단, 단 한 명만 선택할 수 있다.) 혼자 하고 싶어하는 학생은 자기 자신을 선택하는 것도 가능하다.

학생들이($s_1$, $s_2$, ..., $s_r$)이라 할 때, r=1이고 $s_1$이 $s_1$을 선택하는 경우나, $s_1$이 $s_2$를 선택하고, $s_2$가 $s_3$를 선택하고,..., $s_{r-1}$이 $s_r$을 선택하고, $s_r$이 $s_1$을 선택하는 경우에만 한 팀이 될 수 있다.

예를 들어, 한 반에 7명의 학생이 있다고 하자. 학생들을 1번부터 7번으로 표현할 때, 선택의 결과는 다음과 같다.

- 소요시간 : 1시간 이상
- 자료 참고 여부 : △ ('알고리즘 수업' 시 사용한 코드 참고)
- 체감 난이도 : 中


**<총평>**

- 입력 받을 수 있는 최대 사람 수를 맞츠면 시간 초과가 나고, 줄이면 런타임 에러가 나는 이도저도 못하는 상황 .. 🤔
- 어렵진 않다고 생각했는데 골드 3인 이유가 있는 건가 ..? 런타임 에러가 말썽이다 ..

---

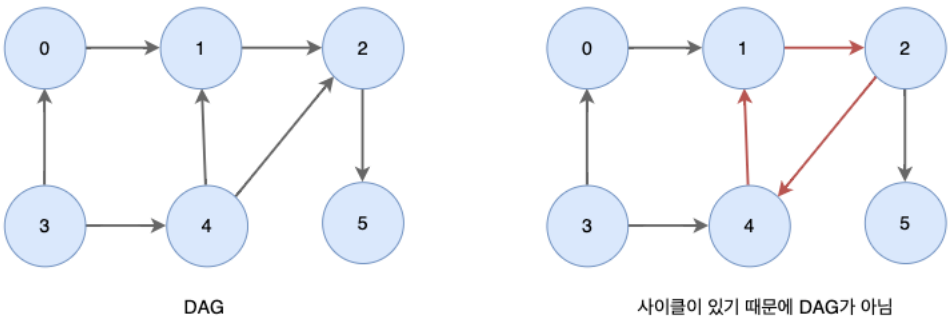
## 📌 9466번

https://www.acmicpc.net/problem/9466


**9466번: 텀 프로젝트**

이번 가을학기에 '문제 해결' 강의를 신청한 학생들은 텀 프로젝트를 수행해야 한다. 프로젝트 팀원 수에는 제한이 없다. 심지어 모든 학생들이 동일한 팀의 팀원인 경우와 같이 한 팀만 있을

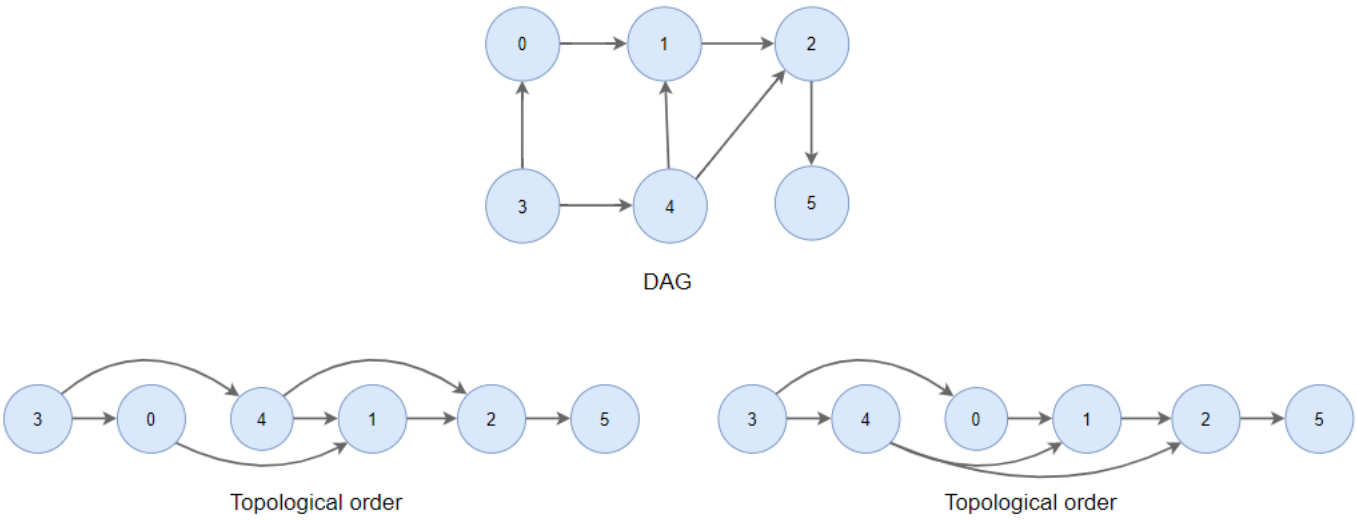
### 방향 그래프




출처 : 티스토리 (https://yoongrammer.tistory.com/86)

- 비순환 방향그래프 (DAG : Directed Acyclic Graph)로 사이클이 없는 방향그래프
- 방향 그래프 알고리즘에서 주요하게 사용되는 것은 큐와 위상정렬
- 방향그래프를 선형으로 정렬하여 위상 순서대로 출력하는 데에 큐를 사용

### 위상 순서 (Topological Sort)




출처 : 티스토리 (https://yoongrammer.tistory.com/86)

- 위상 정렬은 비순환 방향 그래프에서 정점을 선형으로 정렬하는 것이다.
- 위상 정렬 결과, 하나의 DAG에서 하나 이상의 위상 순서가 나올 수 있으며 모든 간선은 오른 쪽만 가리키게 된다.

---

### 📌코드 해석

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

#define NUMV_MAX 99999
#define IN -1
#define OUT 1
```

```
typedef struct Edge {
    int orgIdx;
    int desIdx;
} Edge;

typedef struct IncidentEdge {
    int edgeIdx;
    struct IncidentEdge* next;
} IncidentEdge;

typedef struct Vertex {
    int vName;
    int inDegree;
    IncidentEdge* InIEdges;
    IncidentEdge* OutIEdges;
} Vertex;

typedef struct Graph {
    Vertex* vertices;
    Edge* edges;
} Graph;

typedef struct Node {
    int vIdx;
    struct Node* next;
} Node;

typedef struct Queue {
    Node* front;
    Node* rear;
} Queue;

Graph G;
int n, m; // 정점, 간선 수
int topOrder[NUMV_MAX + 1]; // DAG 여부[0]. 위상 순서[1:n]
Queue Q; // 정점 대기용 큐

int isEmpty();
void enqueue(int vIdx);
int dequeue();
```

- 큐를 사용한 방향 그래프를 만들기 위해 필요한 기본 구조체

```
void initializeGraph() {
    G.vertices = (Vertex*)malloc((NUMV_MAX + 1) * sizeof(Vertex));
}

void insertVertex(int vName, int vIdx) {
    G.vertices[vIdx].vName = vName;
    G.vertices[vIdx].inDegree = 0; // inDegree = 연결된 간선 수 초기화
    G.vertices[vIdx].InIEdges = NULL;
    G.vertices[vIdx].OutIEdges = NULL;
}

int index(int vName) {
    for (int i = 0; i < n; ++i) {
        if (G.vertices[i].vName == vName) {
            return i;
```

```
        }
    }
    return -1; // Vertex not found
}
```

- void initiallizeGraph : 그래프 초기화 함수, 입력 받을 수 있는 최대의 학생 수 만큼 구조체를 동적할당 함.
- void insertVertex : 정점 초기화를 하는 함수
- int index : 입력 받은 vName(요소)가 그래프 내에 있는지 확인하는 함수

```
void addFirst(IncidentEdge** H, int i) {
    IncidentEdge* node = (IncidentEdge*)malloc(sizeof(IncidentEdge));
    node->edgeIdx = i;
    node->next = *H;
    *H = node;
}

void insertIncidentEdge(int vIdx, int eIdx, int in_or_out) {
    if (in_or_out == IN) {
        addFirst(&G.vertices[vIdx].InIEdges, eIdx);
    }
    else {
        addFirst(&G.vertices[vIdx].OutIEdges, eIdx);
    }
}

void insertDirectedEdge(int uName, int wName, int eIndex) {
    int uIdx = index(uName); // 존재하는 정점인지 확인
    int wIdx = index(wName);

    if (uIdx != -1 && wIdx != -1) { // 둘다 존재한다면
        G.edges[eIndex].orgIdx = uIdx;
        G.edges[eIndex].desIdx = wIdx;

        insertIncidentEdge(wIdx, eIndex, IN);
        insertIncidentEdge(uIdx, eIndex, OUT);

        G.vertices[wIdx].inDegree++; // 정점과 연결된 간선의 수 + 1
    }
}
```

- void addFirst : 연결리스트 맨 앞에 노드를 추가하는 함수
  - IncidentEdge 함수를 동적할당 시키고 그 함수의 edgeIdx와 next를 입력받은 요소로 교체한 후에 헤드 포인터 *H를 새로 추가된 노드를 가리키도록 업데이트 (*H = node)
- void insertIncidentEdge : 정점에 대한 인접 간선을 추가하는 함수
  - IN (들어오는 간선) 이라면 > G.vertices[vIdx].InIEdges (정점의 내부로 들어오는 간선을 나타내는 연결 리스트)에 새로운 간선 eIdx 추가
  - OUT (나가는 간선) 이라면 > G.vertices[vIdx].OutIEdges (정점의 내부로 들어오는 간선을 나타내는 연결 리스트)에 새로운 간선 eIdx 추가
- void insertDirectedEdge : insertIncidentEdge 함수를 실행 시키고 inDegree를 증가시키는 함수
  - 입력받은 uName, wName (들어오는 정점, 나가는 정점)이 둘 다 존재한다면, 입력받은 순서의 G.edge 구조체의 orgIdx, desIdx에 입력해준다. 그리고 insertIncidentEdge함수에 요소를 넣고 indegree(들어오는 정점과 연결된 간선의 수)를 증가시킨다.

```
void buildGraph() {
    initializeGraph();

    //printf("Enter the number of vertices: ");
```

```
    scanf("%d", &n);
    int cnt = 1;

    for (int i = 0; i < n; ++i) {
        int vName;
        //  printf("Enter vertex name: ");
        vName = cnt;
        insertVertex(vName, i);
        cnt++;
    }

    cnt = 1;

    //printf("Enter the number of edges: ");
   // scanf("%d", &m);  // 추가: 간선 수 입력 받기

    G.edges = (Edge*)malloc(n * sizeof(Edge));

    for (int i = 0; i < n; ++i) {
        int u, w;
        //  printf("Enter directed edge (u w): ");
        u = cnt;
        scanf("%d", &w);
        insertDirectedEdge(u, w, i);
        cnt++;
    }
}

void topologicalSort() { // 위상 순서 정렬
    int v, w;
    int count = 0;

    for (v = 0; v < n; v++) { // 간선 수 만큼 반복
        if (G.vertices[v].inDegree == 0) {
            enqueue(v);
        }
    }

    while (!isEmpty()) {
        v = dequeue();
        topOrder[++count] = v;

        IncidentEdge* edge = G.vertices[v].OutIEdges;
        while (edge != NULL) {
            w = G.edges[edge->edgeIdx].desIdx;
            G.vertices[w].inDegree--;

            if (G.vertices[w].inDegree == 0) {
                enqueue(w);
            }

            edge = edge->next;
        }
    }

    if (count < n) {
        printf("%d\n", count);
        topOrder[0] = 0; // Graph has a cycle
    }
    else {
        topOrder[0] = 1; // Graph is DAG
    }
}
```

- void buildGraph : 그래프를 만드는 함수
  - 정점의 수를 입력 받고 insertVertex 함수에 수를 하나씩 넣어가며 정점을 그래프에 입력한다.
  - 입력받은 정점 수 만큼 edge 구조체를 동적할당 한 다음 정점 수 만큼 수를 받으며 간선을 연결시킨다. (insertDiretedEdge 함수 사용)
- void topologicalSort : 위상 순서대로 정렬하는 함수 & 사이클에 포함되어 있지 않는 정점 개수 출력
  - inDegree가 0인 정점을 큐에 넣어 초기화 한다.
  - (while 문 이후) BFS 를 수행한다. 큐가 비어있지 않은 동안, 큐에서 정점을 꺼내고 해당 정점의 나가는 간선들을 탐색한다. 해당 간선을 따라간 정점들의 진입 차수를 감소시키고, 진입 차수가 0이 되면 큐에 추가한다.
  - 모든 정점을 방문한 후, 만약 방문한 정점의 수가 그래프의 정점 수와 다르다면 그래프에는 사이클이 존재한다는 것을 의미한다. 따라서, count 수를 기준으로 사이클의 존재 유무를 판단한다.
    - 여기서, count 수는 사이클에 포함되어있지 않는 정점의 개수이고 이 것은 문제에서 출력하길 원하는 수와 같다.

```
int main() {

    int k;
    scanf("%d", &k);

    for (int i = 0; i < k; i++) {

        buildGraph();

        Q.front = NULL;
        Q.rear = NULL;

        topologicalSort();

    }
    free(G.edges);
    free(G.vertices);

    return 0;
}
```

- main 함수 : 반복할 만큼 수를 입력 받고, buildGraph 와 topologicalSort를 실행시켜준다.

---

## 📌 총평

- 알고리즘 자체의 문제인 건지, 아님 다른 불필요한 것이 들어가서 그런건지는 모르겠지만 런타임 에러가 뼈아프다 .. 런타임 에러가 나니까 코드 자체가 틀렸는지 아닌지 알 수 없어서 답답하다 ..



방향그래프 백준 알고리즘

Hyamimi