[백준] 5567번: 결혼식

yoool · 방금 전

> 5567번: 결혼식
>
> - 상근이의 친구와 친구의 친구를 결혼식에 초대(상근이는 1)
> - 첫째 줄: 상근이의 동기의 수 n (2 ≤ n ≤ 500)
> - 둘째 줄: 리스트의 길이 m (1 ≤ m ≤ 10000)
> - 다음 줄부터 m개 줄: 친구 관계 a_i b_i (1 ≤ a_i < b_i ≤ n)
> - 상근이의 결혼식에 초대하는 동기의 수 출력

처음에는 그래프를 인접리스트로 표현하여 문제를 풀려고 했지만, 시간초과가 떴다.

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

typedef struct Edge {           // 간선 노드
    int w;                      // 가중치
    int vNum1, vNum2;           // 양끝 정점 번호
    struct Edge* next;
} Edge;

typedef struct IncidentEdge {   // 부착 리스트 노드
    Edge* edge;                 // 부착 간선 주소
```

```
    struct IncidentEdge* next;
} IncidentEdge;

typedef struct Vertex {         // 정점 노드
    int vNum;                   // 정점 번호
    IncidentEdge* iEdges;       // 부착 리스트
    struct Vertex* next;
} Vertex;

typedef struct Graph {
    Vertex* vertices;           // 정점 리스트
    Edge* edges;                // 간선 리스트
} Graph;

void insertVertex(Graph* graph, int vNumber) {
    // 정점 리스트의 앞에 vNumber를 가지는 고립된 새 정점 삽입
    Vertex* newVertex = (Vertex*)malloc(sizeof(Vertex));
    newVertex->vNum = vNumber;
    newVertex->iEdges = NULL;
    newVertex->next = graph->vertices;
    graph->vertices = newVertex;
}

Vertex* findVertex(Graph* graph, int vNumber) {
    // 정점 리스트를 순회하며 vNumber를 가지는 정점 노드 찾아서 반환, 없으면 NULL 반환
    Vertex* current = graph->vertices;
    while (current != NULL) {
        if (current->vNum == vNumber) {
            return current;
        }
        current = current->next;
    }
    return NULL;
}

Edge* findEdge(Graph* graph, int vNumber1, int vNumber2) {
    // 간선 리스트를 순회하며 정점 번호 vNumber1, vNumber2를 연결하는 간선 노드 찾아서 반환
    Edge* current = graph->edges;
    while (current != NULL) {
        // (vNumber1, vNumber2) (vNumber2, vNumber1) 모두 고려
        if ((current->vNum1 == vNumber1 && current->vNum2 == vNumber2) ||
            (current->vNum1 == vNumber2 && current->vNum2 == vNumber1)) {
            return current;
        }
        current = current->next;
    }
    return NULL;
}

int opposite(Edge* edge, int vNumber) {
    // vNumber 정점의 부착 간선 edge의 반대편 정점 번호 반환
    if (edge->vNum1 == vNumber) {
        return edge->vNum2;
    }
    else {
        return edge->vNum1;
    }
}
```

```
void insertOneIncidentEdge(Graph* graph, int vNumber, Edge* Edge) {
    // vNumber 정점의 부착 리스트에 Edge의 주소를 가지는 부착 리스트 노드 삽입
    // 부착 간선 리스트의 반대쪽 정점 번호에 대해 오름차순으로 정렬되도록 삽입

    // vNumber를 가지는 정점 찾기
    Vertex* vertex = findVertex(graph, vNumber);
    if (vertex == NULL) {
        printf("-1\n");
        return;
    }

    // 정점의 부착 리스트를 순회하며,
    // opposite(순회 노드의 edge, vNumber), oppositie(Edge, vNumber) 비교하며
    IncidentEdge* newIncidentEdge = (IncidentEdge*)malloc(sizeof(IncidentEdge));
    newIncidentEdge->edge = Edge;

    IncidentEdge* current = vertex->iEdges;
    IncidentEdge* prev = NULL;
    int oppositeVertex = opposite(Edge, vNumber);

    while (current != NULL && opposite(current->edge, vNumber) < oppositeVertex) {
        prev = current;
        current = current->next;
    }

    // Edge를 가리키는 새 부착 리스트 노드를 적절히 삽입
    if (prev == NULL) {
        newIncidentEdge->next = vertex->iEdges;
        vertex->iEdges = newIncidentEdge;
    }
    else {
        newIncidentEdge->next = current;
        prev->next = newIncidentEdge;
    }
}

void insertIncidentEdges(Graph* graph, int vNumber1, int vNumber2, Edge* newEdge) {
    // insertOneIncidentEdge 사용
    // newEdge가 루프인 경우: vNumber1 정점의 부착 리스트 업데이트
    if (vNumber1 == vNumber2) {
        insertOneIncidentEdge(graph, vNumber1, newEdge);
    }
    // newEdge가 루프가 아닌 경우: vNumber1 정점의 부착 리스트 & vNumber2 정점의 부착 리스트 업데이트
    else {
        insertOneIncidentEdge(graph, vNumber1, newEdge);
        insertOneIncidentEdge(graph, vNumber2, newEdge);
    }
}

void insertEdge(Graph* graph, int vNumber1, int vNumber2, int weight) {
    // 간선 리스트의 앞에 정점 vNumber1, vNumber2를 연결하는 새 간선 삽입
    Edge* newEdge = (Edge*)malloc(sizeof(Edge));
    newEdge->vNum1 = vNumber1;
    newEdge->vNum2 = vNumber2;
    newEdge->w = weight;
    newEdge->next = graph->edges;
    graph->edges = newEdge;
```

```
    // 간선 삽입 시 정점 vNumber1, vNumber2의 부착 리스트 업데이트(insertIncidentEdges)
    insertIncidentEdges(graph, vNumber1, vNumber2, newEdge);
}

int main() {

    Graph graph;
    graph.vertices = NULL;
    graph.edges = NULL;

    int m, n, a, b, i, j, invite = 0;

    scanf("%d", &m);
    for (i = 1; i <= m; i++)
        insertVertex(&graph, i);

    scanf("%d", &n);
    for (i = 1; i <= n; i++) {
        scanf("%d %d", &a, &b);
        insertEdge(&graph, a, b, 1);
    }

    for (i = 2; i <= m; i++) {
        Edge* frend = findEdge(&graph, 1, i);
        if (frend != NULL && frend->w == 1) {
            printf("origin: %d %d\n", 1, i);
            invite++;
            frend->w++;
        }
        else {
            for (j = 2; j <= m; j++) {
                Edge* newfrend1 = findEdge(&graph, 1, j);
                Edge* newfrend2 = findEdge(&graph, i, j);
                if (newfrend1 != NULL && newfrend2 != NULL && newfrend1->w != 0 && newfrend2->
                    insertEdge(&graph, 1, i, 0);
                    printf("new: %d %d\n", 1, i);
                    invite++;
                    break;
                }
            }
        }

    }

    printf("%d", invite);

    return 0;
}
```

그래서 두 번째 방법으로 BFS를 사용해봤다. 비교적 코드 길이가 훨씬 짧았다.
너비 우선 탐색이므로, 친구의 친구까지인 깊이가 2 이하일 때까지 순회하며 사람 수를 구했다.

처음에는 배열로만 작성하다가 메모리를 너무 많이 사용하는 것 같아서 동적할당으로 바꿔주었는데 큰 차이가 나지는 않았다.

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

void BFS(int** graph, int* visited, int n, int v) {
    int front = 0, rear = 0, num = 0;
    int* queue = (int*)malloc(sizeof(int) * (n + 1));
    int* depth = (int*)malloc(sizeof(int) * (n + 1));

    visited[v] = 1;
    queue[rear++] = v;
    depth[v] = 0;

    while (front < rear) {
        int pop = queue[front++];
        for (int i = 1; i <= n; i++) {
            if (visited[i] == 0 && graph[pop][i] == 1) {
                visited[i] = 1;
                depth[i] = depth[pop] + 1;
                if (depth[i] <= 2) {
                    queue[rear++] = i;
                    num++;
                }
            }
        }
    }

    printf("%d", num);

    free(queue);
    free(depth);
}

int main() {
    int n, m, a, b;

    scanf("%d %d", &n, &m);

    int** graph = (int**)malloc(sizeof(int*) * (n + 1));
    for (int i = 0; i <= n; i++) {
        graph[i] = (int*)calloc(n + 1, sizeof(int));
    }

    int* visited = (int*)calloc(n + 1, sizeof(int));

    for (int i = 0; i < m; i++) {
        scanf("%d %d", &a, &b);
        graph[a][b] = 1;
        graph[b][a] = 1;
    }

    BFS(graph, visited, n, 1);
```

```
    for (int i = 0; i <= n; i++)
        free(graph[i]);
    free(graph);
    free(visited);

    return 0;
}
```

김지율

이전 포스트
[백준] 1920번: 수 찾기

0개의 댓글

댓글을 작성하세요

댓글 작성