2023 Algorithm Study Week 9 - 과제

cheonwon · 방금 전 · 비공개

2023 Algorithm Study smarcle

그래프

노드와 간선들로 구성된 자료구조이며, 트리와 달리 순서가 딱히 상관이 없이 원하는 형태로 탐색이 가능한것 같다. 다만, 구현자체가 매우 빡세고 어렵다.
https://www.acmicpc.net/problem/5567

결혼식 <백준 5567>

문제자체는 매우 간단하며, 풀이방법도 매우 직관적으로 보인다. BFS, 즉 너비우선탐색을 통해 레벨을 전체를 훑고 지나가고, 총 2레벨을 훑으면 된다. 여기서 한 레벨은 각각 그 친구를 몇명 거쳐서 친한지를 의미하게 된다. 즉 친구의 친구는 레벨 2, 자기의 친구는 레벨 1에 위치하며, 레벨 2까지만 너비우선탐색을 진행하고 그 정점수를 출력하면 바로 풀리는 문제이다.
다만, 늘 그렇듯 그래프는 구현이 문제이다.

```
#pragma warning(disable:4996)
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define fresh 0
#define visited 1
#define tree 1
#define cross 2

typedef struct Edge {
    int vNum1, vNum2;
    struct Edge* next;
    int eLabel;
} Edge;
```

```
typedef struct Vertex {
    int vNum;
    struct Vertex* next;
    int vLabel;
} Vertex;

typedef struct Graph {
    Vertex* vertices;
    Edge* edges;
    Edge* adjmtx[101][101];
} Graph;

void init(Graph* graph) {
    graph->vertices = graph->edges = NULL;
    memset(graph->adjmtx, NULL, sizeof(graph->adjmtx));
}

void insertVertex(Graph* graph, int vNumber) {
    Vertex* v = (Vertex*)malloc(sizeof(Vertex));
    v->vNum = vNumber;
    v->vLabel = 0;
    v->next = graph->vertices; //Graph null 초기화 안해줘도 되는진 잘 모르겠음.
    graph->vertices = v;
}

Vertex* findVertex(Graph* graph, int vNumber) {
    Vertex* p = graph->vertices;
    while (p != NULL) {
        if (p->vNum == vNumber) {
            return p;
        }
        p = p->next;
    }
    return NULL;
}

Edge* findEdge(Graph* graph, int vNumber1, int vNumber2) {
    Edge* p = graph->edges;
    while (p != NULL) {
        if (p->vNum1 == vNumber1 && p->vNum2 == vNumber2) {
            return p;
        }
        else if (p->vNum1 == vNumber2 && p->vNum2 == vNumber1) {
            return p;
        }
        p = p->next;
    }
    return NULL;
}

void insertAdjMat(Graph* g, int vNum1, int vNum2, Edge* edge) {
    g->adjmtx[vNum1][vNum2] = edge;
    g->adjmtx[vNum2][vNum1] = edge;
}

void insertEdge(Graph* g, int vNum1, int vNum2) {
    Edge* e = (Edge*)malloc(sizeof(Edge));
    e->vNum1 = vNum1;
```

```
    e->vNum2 = vNum2;
    e->next = g->edges; //간선리스트에 삽입도 하고
    e->eLabel = fresh;
    g->edges = e;

    insertAdjMat(g, vNum1, vNum2, e); //인접행렬과 연결도 함.
}


void processA(Graph* graph, int a, int numV) {
    Vertex* v = findVertex(graph, a);
    if (v == NULL) {
        printf("-1\n");
        return;
    }
    for (int i = 1; i <= numV; i++) {
        if (graph->adjmtx[a][i] != NULL)
            printf("%d ", i);
    }
    printf("\n");
}
void print_all(Graph* g, int numV) {
    for (int i = 1; i <= numV; i++) {
        processA(g, i, numV);
    }
}

void initializeGraph(Graph* g, int numV, int numE) {
    for (int i = numV; i >= 1; i--) {
        insertVertex(g, i);
    }

    int a, b;
    for (int i = 0; i < numE; i++) {
        scanf("%d%d", &a, &b);
        insertEdge(g, a, b);
    }
}

int opposite(Edge* edge, int vNumber) {
    if (edge->vNum1 == vNumber)
        return edge->vNum2;
    else
        return edge->vNum1;
}

int cnt = 0;

void BFS1(Graph* graph, int s, int numV) {
    Vertex* L[100][100];
    memset(L, NULL, sizeof(L));
    Vertex* v = findVertex(graph, s);
    v->vLabel = visited;
    L[0][0] = v;
    int i = 0;
    while (L[i][0] != NULL) {
        int idx = 0;
        for (int a = 0; a < numV; a++) {
```

```
            if (L[i][a] != NULL) { //전 레벨의 원소들
                for (int b = 1; b <= numV; b++) {
                    if (graph->adjmtx[L[i][a]->vNum][b] != NULL) {
                        if (graph->adjmtx[L[i][a]->vNum][b]->eLabel == fresh) {
                            Vertex* w = findVertex(graph, opposite(graph->adjmtx[L[i][a]->vNum
                            if (w->vLabel == fresh) {
                                graph->adjmtx[L[i][a]->vNum][b]->eLabel = tree;
                                w->vLabel = visited;
                                cnt++;
                                L[i + 1][idx++] = w;
                            }
                            else {
                                graph->adjmtx[L[i][a]->vNum][b]->eLabel = cross;
                            }
                        }
                    }
                }
            }
        }
        if (i == 1)
            return;
        i++;
    }
}

void BFS(Graph* graph, int s, int numV) {
    BFS1(graph, s, numV);
}

int main()
{
    Graph g;
    init(&g);
    int numV, numE;
    scanf("%d%d", &numV, &numE);
    initializeGraph(&g, numV, numE);

    BFS(&g, 1, numV);
    printf("%d", cnt);

    return 0;
}
```

인접행렬 방식의 구현이며, 코드자체는 실습시간(그래프 순회 2번)에 했던것과 동일하니 크게 따로 설명하지는 않겠다. 레벨이 2가 되면(0부터 시작하니 1이면) 종료하도록 했고 실제로도 예시는 잘 출력이 된다.
다만 1. 인접행렬의 크기를 더 늘릴수가 없어서 실제 문제의 갯수만큼 입력을 받을 수 없고 (오류남. 이유는 잘 모르겠음), 2. 인접행렬의 크기를 더 늘릴 수 있다고 하더라도 메모리가 아마 초과되어서 메모리 초과 판정을 받을것이다.

인접리스트를 통해서 BFS를 구현한다면 해결 될 것 같다.

cheonwon

뭐든지 한걸음씩. 초보 개발자

이전 포스트

2023 Algorithm Study Week 8 - 과제

0개의 댓글

댓글을 작성하세요

댓글 작성