자료구조 과제 HW#4

Prof.조성현

Asis. 지서연

201711205 황정평

-컴파일 환경

window10, Visualstudio2017, _MSC_VER: 1916, MSYC++14.16

Prob-1.cpp

위상정렬_알고리즘

dfs 기반 위상정렬. dfs로 탐색한 노드들을 역순으로 배열하였다.

1) a노드(시작노드)를 방문한다. 방문 시간을 입력하고, 회색으로 칠해준다.

2) 주변 하얀 노드를 방문한다. 역시 방문 시간을 입력하고, 회색으로 칠해준다.

3) 방문할 주변 하얀 노드가 없을 때, 검은색으로 칠하고 종료 시간은 입력한다.

4) 이전 노드로 올라가서 다시 하얀 노드를 방문한다.

그 후, 종료시간을 기준으로 내림차순 정렬을 해주었다.

-코드 설명

```
#include<iostream>
#include<vector>
#include<string>
#include<algorithm>
#define white 0
#define gray 1
#define black 2
using namespace std;
```

자료구조 string과 vector를, sort해주기 위해 algorithm을 받아온다. 그 후, white gray black 상수를 정의해주었다.

```
class node {
public:
    int entryLevel;
    node* prev;
    vector<node*> nexts;
    int color;
    int d;
    int f;
```

```
    string name;
    int number;
    node(int n, string s) {
        entryLevel = 0;
        number = n;
        name = s;
        prev = nullptr;
        nexts = vector<node*>();
        color = white;
    }

};
```

노드 클래스를 만들어준다. 진입차수, 이전노드, 갈 수 있는 다음노드의 집합, 색, 진입 시간, 회귀 시간, 이름, 숫자 멤버변수를 만들어준다. 숫자와 이름을 초기화해줄 수 있는 생성자를 만든다.

```
int time = 0;
void dfs(node* u) {
    time++;
    u->color = gray;
    u->d = time;
    for (int i = 0; i < u->nexts.size(); i++) {
        if (u->nexts[i]->color == white) {
            u->prev = u;
            dfs(u->nexts[i]);
        }
    }
    time++;
    u->color = black;
    u->f = time;
}
```

시간을 0으로 선언&초기화해주고, dfs함수 시작하면 시간을 증가시키고, 현재노드 시간에 입력해준다. 또한 색도 gray로 바꾸어준다. 그 후, 갈 수 있는 다음노드 중 색이 white인 노드로 간다. 그 노드의 이전노드를 현재노드로 설정해주고, 그 노드를 기점으로 다시 dfs를 해준다. 다음노드들에 대해 다 작업을 해주고, 시간을 증가시키고, 회귀 시간에 시간을 입력해준다. 색 또한 black으로 칠해준다.

```
bool compare(node* a, node* b) {
    return a->number < b->number;
}
bool compare2(node* a, node* b) {
    return a->f > b->f;
}
```

뒤에 sort함수에서 사용하기 위하여 만든 boolean함수이다. compare는 노드의 number를 기준으로 오름차순 정렬을 해준다. compare2는 노드의 회귀 시간을 기준으로 내림차순 정렬을 해준다.

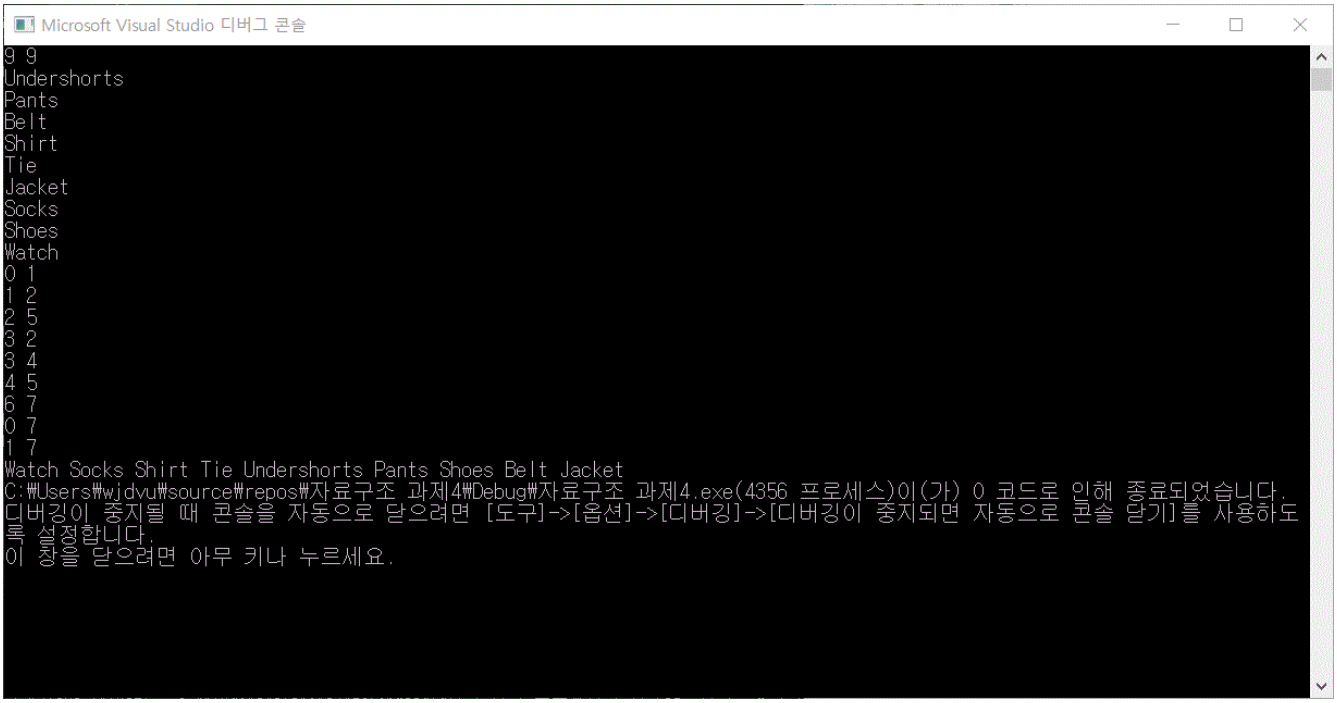
```
int main() {
    vector<node*> nodeTable;
    vector<node*> firstNodeTable;
    int nodeCount;
```

```
    int edgeCount;
    cin >> nodeCount >> edgeCount;
    cin.ignore();
    for (int i = 0; i < nodeCount; i++) {
        string name;
        getline(cin, name);
        node* newnode = new node(i, name);
        nodeTable.push_back(newnode);
    }
    for (int j = 0; j < edgeCount; j++) {
        int a;
        int b;
        cin >> a >> b;
        nodeTable[b]->entryLevel++;
        nodeTable[a]->nexts.push_back(nodeTable[b]);
    }
    for (int k = 0; k < nodeTable.size(); k++) {
        if (nodeTable[k]->nexts.size() > 1) {
            sort(nodeTable[k]->nexts.begin(), nodeTable[k]->nexts.end(), compare);
        }
        if (nodeTable[k]->entryLevel == 0) {
            firstNodeTable.push_back(nodeTable[k]);
        }
    }
    for (int h = 0; h < firstNodeTable.size(); h++) {
        dfs(firstNodeTable[h]);
    }
    sort(nodeTable.begin(), nodeTable.end(), compare2);
    for (int g = 0; g < nodeTable.size(); g++) {
        cout << nodeTable[g]->name << " ";
    }
}
```

노드들을 담는 nodeTable을 선언해준다. 진입 차수가 0인 노드들을 담는 firstNodeTable을 선언해준다. 총 노드의 갯수와 간선의 갯수를 nodeCount, edgeCount에 받는다. geline으로 받기에 앞서 엔터를 받을 수 있기 때문에, cin.ignore()를 해준다. 이제 노드의 갯수만큼 이름을 입력받고 동적 생성하여 nodeTable에 넣어준다. 그 후, 간선의 갯수만큼 번호들을 입력받고, 방향성을 따져 노드의 진입 차수를 올려주고, 노드의 nexts에 해당 노드를 넣어준다. 다음, 모든 노드들을 확인하는데, nexts를 number의 오름차순으로 노드들을 정렬해주고, 진입차수가 0인 노드들을 firstNodeTable에 넣어줍니다. 그리고 firstNodeTable에 있는 모든 노드들에 대하여 dfs를 해줍니다.

마지막으로 nodeTable을 회귀시간 내림차순으로 정렬을 해주고, 순서대로 노드 이름을 출력합니다.

-결과창



마지막 줄을 제외한 윗 줄들이 입력값들이고, 마지막 줄이 출력값이다. 주어진 예제 입출력 그대로 잘 나온것을 확인할 수 있다.

Prob-2.cpp

다익스트라_알고리즘

1. 출발점으로부터의 최단거리를 저장할 배열 d[v]를 만들고, 출발 노드에는 0을, 출발점을 제외한 다른 노드들에는 매우 큰 값 INF를 채워 넣는다. (정말 무한이 아닌, *무한으로 간주될 수 있는 값*을 의미한다.) 보통은 최단거리 저장 배열의 이론상 최대값에 맞게 INF를 정한다. 실무에서는 보통 d의 원소 타입에 대한 최대값으로 설정하는 편.
2. 현재 노드를 나타내는 변수 A에 출발 노드의 번호를 저장한다.
3. A로부터 갈 수 있는 임의의 노드 B에 대해, d[A] + P[A]와 d[B]의 값을 비교한다. INF와 비교할 경우 무조건 전자가 작다.
4. 만약 d[A] + P[A]의 값이 더 작다면, 즉 더 짧은 경로라면, d[B]의 값을 이 값으로 갱신시킨다.
5. A의 모든 이웃 노드 B에 대해 이 작업을 수행한다.
6. A의 상태를 "방문 완료"로 바꾼다. 그러면 이제 더 이상 A는 사용하지 않는다.
7. "미방문" 상태인 모든 노드들 중, 출발점으로부터의 거리가 제일 짧은 노드 하나를 골라서 그 노드를 A에 저장한다.
8. 도착 노드가 "방문 완료" 상태가 되거나, 혹은 더 이상 미방문 상태의 노드를 선택할 수 없을 때까지, 3~7의 과정을 반복한다.

-코드 설명

```
#include<iostream>
#include<vector>
#include<string>
#include<map>
#include<algorithm>
#define max 2147483647
#define white 0
#define gray 1
#define black 2
using namespace std;
```

자료구조 string, vector를 받아오고 map, algorithm 라이브러리도 받아온다. int의 최대 값을 max에 선언해주고 색 white gray black을 선언해준다.

```
class Node {
public:
    int number;
    string name;
    int d = max;
    int color = white;
    vector<Node*> nexts;
    vector<Node*> path;
    Node(int i, string s) {
        number = i;
        name = s;
    }
};
```

노드 클래스를 생성한다. 번호, 이름, 시작노드로 부터의 거리, 색, 갈 수 있는 다음 노드들의 집합, 최단 거리로 왔을 때 거친 노드들을 담는 벡터 멤버 변수를 선언해준다. 번호와 이름을 초기화할 수 있는 생성자를 만든다.

```
void bfs_dij(Node* node, int** adjTable) {
    node->color = gray;
    Node* nextnode = nullptr;
    node->path.push_back(node); // 자기 자신을 경로에 추가

    for (int i = 0; i < node->nexts.size(); i++) {

        if (node->nexts[i]->color == white) {

            if (node->d + adjTable[node->number][node->nexts[i]->number] < node->nexts[i]-
>d) { // 현재 등록된 것 보다, 지금 노드를 경유해 가는게 더 빠를 때
                node->nexts[i]->d = node->d + adjTable[node->number][node->nexts[i]-
>number];
                node->nexts[i]->path = node->path;

            }
            if (nextnode == nullptr) {
                nextnode = node->nexts[i];
                continue;
            }
```

```
            if (adjTable[node->number][nextnode->number] > adjTable[node->number][node->nexts[i]->number]) {
                nextnode = node->nexts[i];
            }
        }
    }
    if (nextnode == nullptr) {
        return;
    }
    node->color = black;
    bfs_dij(nextnode, adjTable);
}
```

인접 행렬과 노드를 인자로 받아온다. 우선 해당 노드는 gray 로 칠해주고, 다음노드는 우선 nullptr로 선언하고, 자기 자신을 경로에 추가한다. 갈 수 있는 다음 노드들에 대하여 그 노드가 흰색일 때, 그 노드의 등록된 거리 값 보다 지금 노드를 경유해 가는게 더 빠를 때, 거리 값을 후자로 업데이트 해주고, 경로 또한 현재 노드의 경로로 초기화해준다. 이후 이제 다음노드를 골라주는데, nullptr이면 현재 보고 있는 다음 노드를 다음노드로 설정해주고, 아니라면 현재노드와 거리가 가장 짧은 노드가 다음노드가 되도록 한다.

다음 노드를 고르지 못하였다면 끝에 다다른 것이기에 return한다.

아니라면 현재 노드를 black으로 칠해주고, 다음 노드에 다시 bfs_dij를 해준다.

```
int main() {
    map<string, Node*> nodeTable; // 노드 사전 생성
    int nodeCount;
    int edgeCount;
    cin >> nodeCount >> edgeCount;
    cin.ignore();
    string start;
    string end;
    cin >> start >> end;
    cin.ignore();

    int** adjTable = new int*[nodeCount]; // 이차원 배열 생성
    for (int i = 0; i < nodeCount; i++) {
        adjTable[i] = new int[nodeCount];
    }

    for (int i = 0; i < nodeCount; i++) { // 노드 추가중
        string name;
        getline(cin, name);
        Node* node = new Node(i, name);
        node->number = i;
        nodeTable.insert(pair<string, Node*>(name, node));
    }

    for (int i = 0; i < edgeCount; i++) { // 간선 추가중
        string a;
        string b;
        int dis;
        cin >> a >> b >> dis;
```

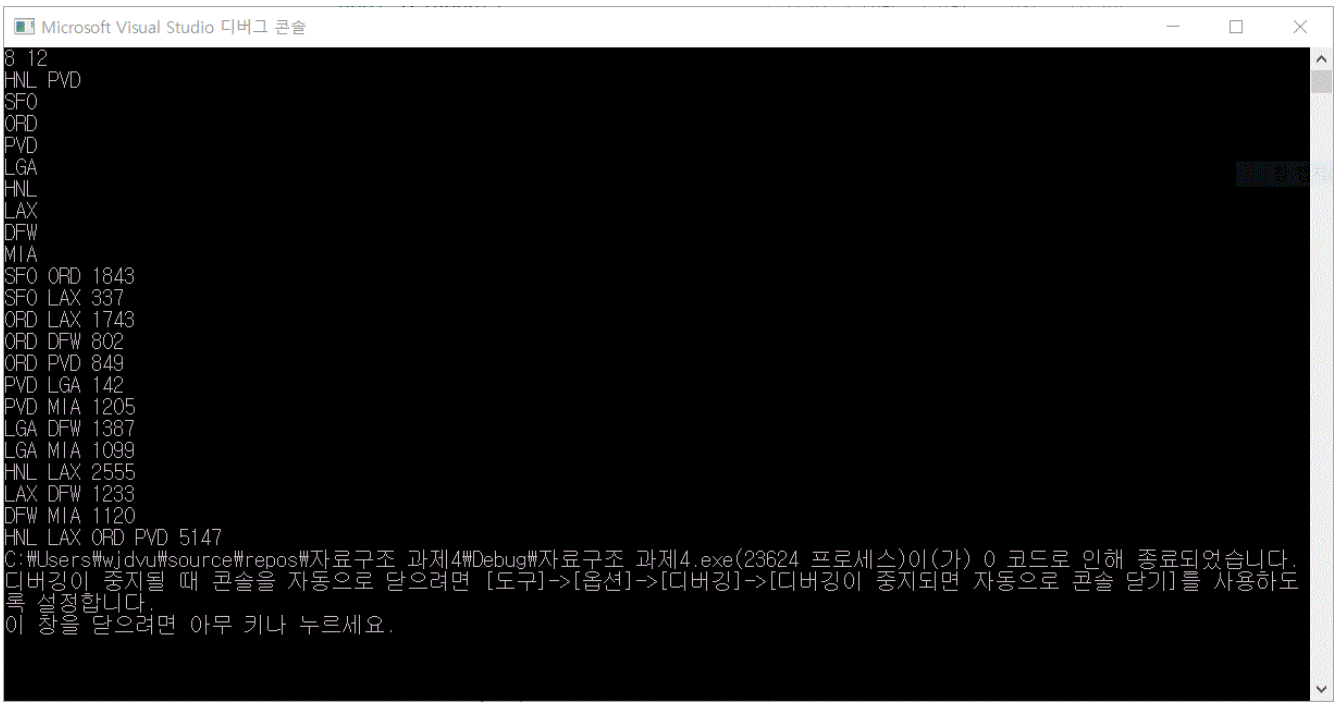
```
        adjTable[nodeTable.find(a)->second->number][nodeTable.find(b)->second->number] =
dis;
        adjTable[nodeTable.find(b)->second->number][nodeTable.find(a)->second->number] =
dis;
        nodeTable.find(a)->second->nexts.push_back(nodeTable.find(b)->second);
        nodeTable.find(b)->second->nexts.push_back(nodeTable.find(a)->second);
    }
    nodeTable.find(start)->second->d = 0;
    bfs_dij(nodeTable.find(start)->second, adjTable);
    for (int i = 0; i < nodeTable.find(end)->second->path.size(); i++) {
        cout << nodeTable.find(end)->second->path[i]->name << " ";
    }
    cout << nodeTable.find(end)->second->d;

}
```

사전형 nodeTable을 생성해준다. 노드의 갯수, 간선의 갯수를 nodeCount, edgeCount에 입력받는다. 시작 노드와 도착 노드를 start, end에 입력받는다. 각 거리값을 저장하는 인접행렬 adjTable을 선언한다. 번호와 이름으로 노드를 동적 생성해주며 이름과 노드를 쌍으로 map nodeTable에 넣어준다. 간선 추가할 때, 표에 거리 값을 입력해주고 각 노드의 nexts에 서로 추가해준다.

시작 노드의 거리 값을 0으로 초기화 해주고, bfs_dij에 시작 노드와 인접행렬을 넣어준다.

그 후, 마지막 노드의 path의 노드 이름들을 순서대로 출력한다. 마지막으로 거리 값을 출력해준다.

-결과창



마지막줄이 출력 값이고, 그 외 줄들이 입력 값이다. 주어진 예제 입출력 그대로 잘 나온것을 확인할 수 있다.