

Topological Sort

학번 : 201002513

이름 : 최 혁수

❑ 실습 2. Topological Sort

- 개요 & 알고리즘
- 주요 소스코드
- 실행결과 분석 & 구현상의 오류 및 한계

❑ 소감문 & 한계성

❑ 전체 소스코드

실습 2. Topological Sort

- 개요 & 알고리즘
- 주요 소스코드
- 실행결과 분석 & 구현상의 오류 및 한계

Topological Sort

□ 개요

정렬(Sort)의 구현방법중 위상정렬(Topological sort) 알고리즘의 개념적 이해와 구현을 통해 그래프(graph)의 선행순서를 위배하지 않고 모든 정점을 탐색하는 방법을 모색한다.

□ 접근방법

- 사용자로부터 vertex의 개수를 직접 입력받아 사용한다.
- 스택(Stack) 기능을 표면적으로만 구현할 'int stack'을 정의해 값을 출력한다.
- 정점의 내차(indegree)를 표현할 'int count'를 정의해 값을 저장한다.
- 방향 그래프(Directed graph)를 노드(Node)를 이용해 리스트(List)로 구현한다.

□ 위상정렬 알고리즘

위상정렬의 알고리즘은 방향그래프(Directed graph)로 표현된 각 정점(vertex)의 위상 선형 정렬을 의미하며, 이를 위한 전제조건으로는 생성된 그래프(Graph)는 사이클(Cycle)이 존재하지 않아야 한다.

위상정렬은 기본적으로 특정 위계가 정의된 구조상에서 위계에 따른 방향으로 정렬하는 방식으로 이를 지키면서 모든 정점을 탐색하는 과정을 반복하며 수행한다.

Step 1. indegree=0인 정점(Vertex)를 찾는다

Step 2. 해당 정점(Vertex)를 출력하고, 연결된 간선(Edge)를 삭제한다.

Step 3. 그래프(Graph)상에 정점(Vertex)가 남아있다면, Step1~3의 과정을 반복한다.

Topological Sort Source Code(1/2)

□ topological.h

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

#define MAX_VERTEX 10 // 최대 정점의 갯수 10로 정의

typedef struct node *node_pointer;
typedef struct node{
    int vertex;
    /* node를 아직 정의하지 않았으므로, struct node* link로 정의해야 오류가 나지 않는다 */
    struct node* link;
} node;

typedef struct headNode{
    int count; // indegree count
    node *link; // headnode[] 에 연결된 노드
}headNode;

headNode graph[MAX_VERTEX]; // 10크기의 graph 생성

void initGraph(int); // 초기화
void insertEdge(); // 정점연결
void createGraph(int, int); // 그래프 생성
void topologicalOrder(int); // 위상정렬 및 출력
void printGraph(int); // 그래프 출력
```

□ void createGraph(int, int)

```
void createGraph(int vertex1, int vertex2){
    node_pointer temp; // 데이터(vertex2)를 갖는 노드 생성
    node_pointer gogo; // 탐색을 위한 노드

    temp = (node_pointer)malloc(sizeof(node));
    temp->vertex = vertex2;
    temp->link = NULL;

    gogo = graph[vertex1].link; // gogo 노드 초기화
    if(graph[vertex1].link == NULL){ // graph[]에 딸린 노드가 없을때
        graph[vertex1].link = temp; // vertex2 노드를 연결
    }
    else{
        while(gogo->link!=NULL){ // 노드가 연결되었을때
            gogo = gogo->link; // 노드의 끝을 탐색
        }
        gogo->link = temp; // 노드의 끝에 연결
    }
    graph[vertex2].count++; // indegree 값 +1
}
```

Topological Sort Source Code(2/2)

❑ void topologicalOrder(int)

```
void topologicalOrder(int vertex){
    node_pointer temp;    // 탐색 노드
    int top=-1;           // stack 의 기능 구현
    int i;

    for(i=0; i<vertex; i++){
        if(graph[i].count == 0){
            graph[i].count = top; // 초기 index=0 에 대한 설정
            top = i;              // top의 값 재정의
        }
    }
    while(top!=-1){
        // 위상정렬이 끝날때
        printf("%d ", top);      // top의 값 출력
        for(temp=graph[top].link; temp!=NULL; temp=temp->link)
            // 인접노드의 데이터(vertex)를 graph[]의 index에 점목하여 indegree값을 1 감소
            graph[temp->vertex].count--;
        top=graph[top].count;    // top 재정의

        /* 앞서 graph[]의 indegree의 재정의값중, 0이 발생할 경우를 탐색 */
        /* 즉, indegree=0이 발생할 경우, top에 해당 graph index값을 top에 재정의 */
        for(i=0; i<vertex; i++){
            if(graph[i].count==0){
                graph[i].count=top;
                top=i;
            }
        }
        printf("\n");
    }
}
```

❑ void printGraph(int)

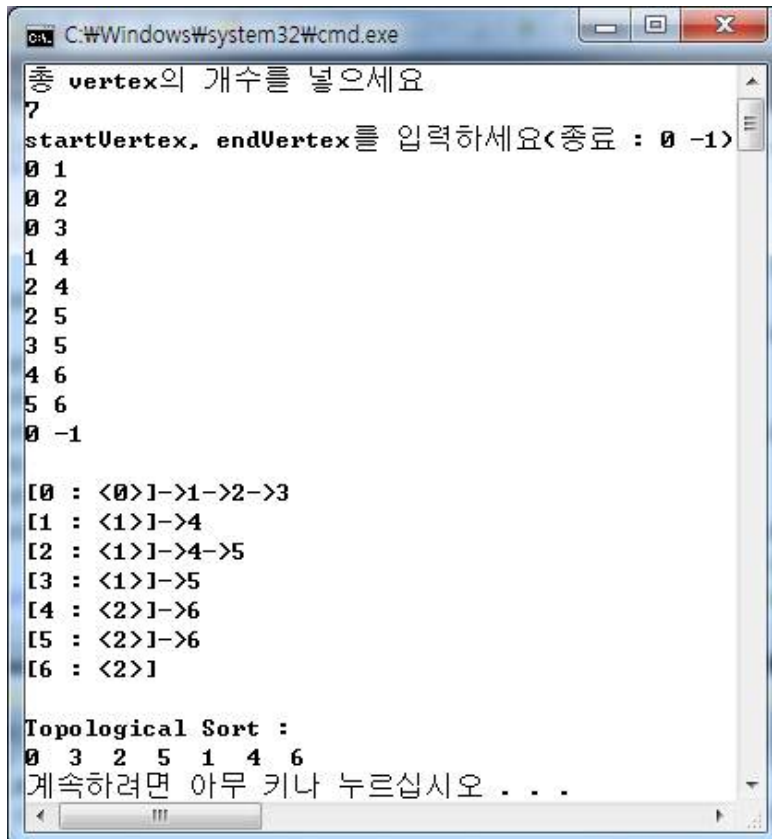
```
void printGraph(int vertex){

    node_pointer temp;
    int i;

    for(i=0; i<vertex; i++){
        printf("[%d : <%d>]", i, graph[i].count);
        for(temp=graph[i].link; temp!=NULL; temp=temp->link)
        {
            printf("->%d", temp->vertex); // graph[]의 연결노드 데이터(vertex)정보를 출력
        }
        printf("\n");
    }
    printf("\n");
}
```

실행결과 분석 & 구현상의 오류 및 한계

□ 실행 결과화면



```
C:\Windows\system32\cmd.exe
총 vertex의 개수를 넣으세요
7
startVertex, endVertex를 입력하세요<종료 : 0 -1>
0 1
0 2
0 3
1 4
2 4
2 5
3 5
4 6
5 6
0 -1

[0 : <0>1->1->2->3
[1 : <1>1->4
[2 : <1>1->4->5
[3 : <1>1->5
[4 : <2>1->6
[5 : <2>1->6
[6 : <2>1

Topological Sort :
0 3 2 5 1 4 6
계속하려면 아무 키나 누르십시오 . . .
```

사용자로부터 입력받은 정점(Vertex)의 연결로 방향 그래프(Directed graph)를 생성한 후, 인접노드를 출력하고 있으며 위상정렬 기능의 수행을 통해 stack 값에 채워진 숫자들을 순차적으로 출력하고 있다.

* 가시적인 표현은 실습과제로 제시한 'Topological Sort.pdf'로 대체한다.

□ 구현상의 오류 및 한계

위 코드는 사용자로부터 Vertex의 개수를 정의받고 있으며, 이는 재정의가 불가능하도록 만들어졌다. 따라서 입력받은 vertex를 기초로 각 기능 함수들이 탐색과정을 수행하고 있으며 만일 vertex의 입력값을 vertex=20; 으로만 정의하여도 topologicalOrder(), createGraph(), printGraph()등의 기능에 한계가 생겨 정상적인 수행이 불가능해진다.

이는 제시된 문제를 해결하는것에 기초로 만들어진 코드로 이 문제해결의 한계치를 벗어나게 되면 정상수행이 불가능하다는 한계성을 가진다.

소감문 & 한계성

소감문 & 한계성

□ 소감문

현 대학 교육과정의 선수과목에 빗대어 접한 위상정렬(Topological sort)를 그래프의 정점방문 기능으로 구현하면서 특정 위계가 정의된 구조상에서의 탐색과정을 확인하였고 이를 확대하여 큰 구조적 특성을 갖춘 기업들이 프로젝트의 기능적인 수행과정을 인지하고 해결할 때 이 위상정렬을 이용해 가시화하면 이로움점을 가질것으로 판단되었다.

또한 구현과정을 정립하면서 그래프 탐색과정인 DFS, BFS를 사용하거나 스택이 아닌 우선순위 큐를 이용해서 위상정렬 구현이 가능하다는 것을 알게되었고 이를 이용해서 만들어보는 기회도 만들 것이다.

□ 한계성

앞서 제시한 정점의 개수에 대한 한계성을 지니고 있으며 그래프(Graph)도 헤더파일에 정의된 MAX_VERTEX:10 으로 크기가 10이라는 한계성을 지니고 있다. 이는 다수의 데이터를 가지는 위상정렬 표현에 불가한 코드이며 이를 해결하기 위해 프로그램의 크기를 사용자가 입력한 데이터의 크기에 비례하여 동적할당을 해야하며, malloc or realloc 등의 기능을 사용할수 있다.

* 해당 문제가 제시된 실습날 과제 PDF 파일에 제시된 Node 코드에서 에러가 발생한 문제는 `typedef struct node`의 정의전에 `node* list` 라는 기능을 정의하고 있어서 에러가 난 것으로 생각되었고, 이를 해결하기 위해 `struct node* list` 로 재정의 하였다.

전체 소스코드

Topological Sort(1/3)

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

#define MAX_VERTEX 10 // 최대 정점의 갯수 10로 정의

typedef struct node *node_pointer;
typedef struct node{
    int vertex;
    struct node* link;
} node;

typedef struct headNode{
    int count; // indegree count
    node *link; // headnode[] 에 연결된 노드
}headNode;

headNode graph[MAX_VERTEX]; // 10크기의 graph 생성

void initGraph(int); // 초기화
void insertEdge(); // 정점연결
void createGraph(int, int); // 그래프 생성
void topologicalOrder(int); // 위상정렬 및 출력
void printGraph(int); // 그래프 출력

/* Graph 초기화 */
void initGraph(int Vertex){
    int i;
    for(i=0; i<Vertex; i++){
        graph[i].count = 0;
        graph[i].link = NULL;
    }
}

/* 정점 연결 및 createGraph() 호출 */
void insertEdge(){
    int vertex1, vertex2;
    printf("startVertex, endVertex를 입력하세요(종료 : 0 -1) : \n");
    while(1){
        scanf("%d %d",&vertex1, &vertex2);
        if(vertex1 == 0 && vertex2 == -1) break; // 종료
        createGraph(vertex1, vertex2); // 그래프 생성
    }
    printf("\n");
}
```

Topological Sort(2/3)

```
void createGraph(int vertex1, int vertex2){
    node_pointer temp; // 데이터(vertex2)를 갖는 노드 생성
    node_pointer gogo; // 탐색을 위한 노드

    temp = (node_pointer)malloc(sizeof(node));
    temp->vertex = vertex2;
    temp->link = NULL;

    gogo = graph[vertex1].link; // gogo 노드 초기화
    if(graph[vertex1].link == NULL){ // graph[]에 딸린 노드가 없을때
        graph[vertex1].link = temp; // vertex2 노드를 연결
    }
    else{
        while(gogo->link!=NULL){ // 노드가 연결되었을때
            gogo = gogo->link; // 노드의 끝을 탐색
        }
        gogo->link = temp; // 노드의 끝에 연결
    }
    graph[vertex2].count++; // indegree 값 +1
}

/* 위상정렬 및 출력 */
void topologicalOrder(int vertex){
    node_pointer temp; // 탐색 노드
    int top=-1; // stack 의 기능 구현
    int i;

    for(i=0; i<vertex; i++){
        if(graph[i].count == 0){
            graph[i].count = top; // 초기 index=0 에 대한 설정
            top = i; // top의 값 재정의
        }
    }
    while(top!=-1){
        // 위상정렬이 끝날때
        printf("%d ", top); // top의 값 출력
        for(temp=graph[top].link; temp!=NULL; temp=temp->link)
            // 인접노드의 데이터(vertex)를 graph[]의 index에 접목하여 indegree값을 1 감소
            graph[temp->vertex].count--;
        top=graph[top].count; // top 재정의

        /* 앞서 graph[]의 indegree의 재정의값중, 0이 발생할 경우를 탐색 */
        /* 즉, indegree=0이 발생할 경우, top에 해당 graph index값을 top에 재정의 */
        for(i=0; i<vertex; i++){
            if(graph[i].count==0){
                graph[i].count=top;
                top=i;
            }
        }
    }
    printf("\n");
}
```

Topological Sort(3/3)

```
void printGraph(int vertex){
    node_pointer temp;
    int i;

    for(i=0; i<vertex; i++){
        printf("[%d : <%d>]", i, graph[i].count);
        for(temp=graph[i].link; temp!=NULL; temp=temp->link)
        {
            printf("->%d", temp->vertex); // graph[]의 연결노드 데이터(vertex)정보를 출력
        }
        printf("\n");
    }
    printf("\n");
}

int main()
{
    int vertex_num;

    printf("총 vertex의 개수를 넣으세요\n");
    scanf("%d", &vertex_num);
    insertEdge();           // 정점 연결 -> createGraph() 호출
    printGraph(vertex_num); // Graph 출력
    printf("Topological Sort : \n");
    topologicalOrder(vertex_num); // 위상정렬 및 출력

    return 0;
}
```