



송실대학교 컴퓨터학부

알고리즘 2021 (나)

과제 5

이름	윤 주호
학번	20201866
출석 번호	225

# 프로그램 개요

## (1) 프로그램 설명

Traveling Salesman Problem 알고리즘을 이해하고, 직접 구현해본다.

손으로 푼 결과와 비교하면서 알고리즘과 익숙해진다.

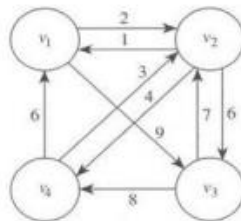
## (2) 과제 명세.

### 알고리즘 2021 과제 5 (10/22까지)

0. 10월 22일 (금) 오후 5:00 제출 마감 (스마트캠퍼스에서 반별로 온라인 제출)

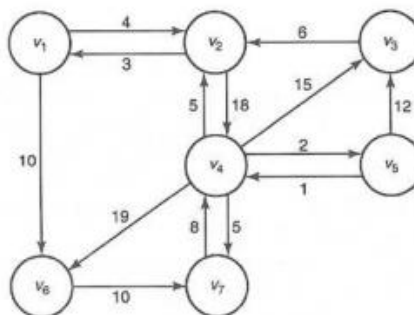
과제 파일명은 '[출석번호]과제4\_홍길동\_00000000.zip'으로 제출

1. 3장 PPT의 61쪽에 있는 Traveling Salesperson 문제를 손으로 작성해서 제출하시오.



2. 위 문제를 프로그래밍으로 구현하고, 손으로 작성한 결과와 비교하시오.

3. 아래의 가중치 포함 방향 그래프에서  $v_1$ 에서 출발해서 모든 도시를 지나고 다시 돌아오는 최적 일주 경로를 프로그램을 이용해서 구하시오.



## <흐름도>

Main 함수의 흐름도는 다음과 같다.

### 1. void input();

Traveling Salesperson Problem 을 풀기 위해서 데이터를 입력 받는다.

처음 노드의 개수를 입력하고, 인접 행렬까지 입력한다.

### 2. void tsp(int start, int i, int sum, int count);

재귀적 방식을 이용해 가능한 모든 경우를 탐색한다. 인접 행렬이 0 인 경우는 그래프가 연결되어 있지 않거나 vertex 의 시작점과 끝점이 같다는 뜻이므로 바로 반복문을 빠져나간다.

### 3. void travel();

Tsp 를 재귀적으로 호출해 이동하는 weight 를 더해서 minimum\_cost 를 구하는 함수이다.

### 4. void Path();

경로를 출력하는 함수. Travel 을 성공적으로 하지 못한 경우에는 minimum\_vertex 를 0 으로 설정한다.

### 5. void output();

우리가 현재 적응하고 있는 tsp 의 근본적인 정답을 구하는 함수이다.

우리는 경로를 출력하고 그에 대한 tsp 의 값을 출력할 것이다.

# <1 번> 손으로 푼 결과

노트에 볼펜으로 작성하고, 스캔한 뒤 붙여 넣었습니다.

[225] 20201866 윤국호 1번.

$V_i \rightarrow \boxed{\phantom{A}} \rightarrow V_1$ 로 보자.  
A

중간에 거리는 노드들의 집합을 A라고 하자.

	1	2	3	4
1	0	2	9	$\infty$
2	1	0	6	4
3	$\infty$	7	0	8
4	6	3	$\infty$	0

**① A =  $\emptyset$ 인 경우**

$D[V_2][A] = [V_2, V_1] = 1$

$D[V_3][A] = [V_3, V_1] = \infty$

$D[V_4][A] = [V_4, V_1] = 6$

**③ A = 2개의 노드로 구성**

(1) A는  $\{V_2, V_3\}$ 이다.

$D[V_4][\{V_2, V_3\}] = \min_{j \in \{2,3\}} (W[4][j] + D[V_j][\{V_2, V_3\} - \{V_4\}])$   
 $= \min(W[4][2] + D[V_2][\{V_3\}], W[4][3] + D[V_3][\{V_2\}])$   
 $\Rightarrow \min(3 + \infty, \infty + 8) = \infty$

(2) A는  $\{V_2, V_4\}$ 이다.

$D[V_3][\{V_2, V_4\}] = \min(W[3][2] + D[V_2][\{V_4\}], W[3][4] + D[V_4][\{V_2\}])$   
 $= \min(7 + 10, 8 + 6) = 12$

(3) A는  $\{V_3, V_4\}$ 이다.

$D[V_2][\{V_3, V_4\}] = \min(W[2][3] + D[V_3][\{V_4\}], W[2][4] + D[V_4][\{V_3\}])$   
 $\Rightarrow \min(6 + 14, 4 + \infty) = 20$

**② A = 1개의 노드로 감성**

(1) A가  $V_2$ 이다.

$D[V_3][\{V_2\}] = \min_{j \in \{2\}} (W[3][j] + D[V_j][\{V_2\} - \{V_3\}])$   
 $\Rightarrow W[3][2] + D[V_2][\emptyset] = 7 + 1 = 8$

$D[V_4][\{V_2\}] = 3 + 1 = 4$

(2) A가  $V_3$ 다.

$D[V_2][\{V_3\}] = 6 + \infty = \infty$

$D[V_4][\{V_3\}] = \infty + \infty = \infty$

(3) A가  $V_4$ 다.

$D[V_2][\{V_4\}] = 4 + 6 = 10$

$D[V_3][\{V_4\}] = 8 + 6 = 14$

**④ A = 3개의 노드로 구성**

A는  $\{V_2, V_3, V_4\}$ 이다.

$D[V_1][\{V_2, V_3, V_4\}] = \min_{j \in \{2,3,4\}} (W[1][j] + D[V_j][\{V_2, V_3, V_4\} - \{V_1\}])$   
 $\Rightarrow \min(W[1][2] + D[V_2][\{V_3, V_4\}], W[1][3] + D[V_3][\{V_2, V_4\}], W[1][4] + D[V_4][\{V_2, V_3\}])$   
 $= \min(2 + 20, 9 + 12, \infty + \infty) = 21$

확인하기

$\Rightarrow V_1 \rightarrow V_3 \rightarrow V_4 \rightarrow V_2 \rightarrow V_1$  이 최단 경로이며  
길이는 21이다.

## <2 번과 3 번> TSP Algorithm.

<코드의 실행 결과이다>

 Microsoft Visual Studio 디버그 콘솔

```
방문할 vertex의 개수: 4
```

```
인접행렬(adjacent matrix) 입력
```

```
0 2 9 0
```

```
1 0 6 4
```


```
0 7 0 8
```

```
6 3 0 0
```

```
최소비용(minimum cost): 21
```

```
Optimal tour의 Path: 1 -> 3 -> 4 -> 2 -> 1
```

비교 : 손으로 푼 1 번과 동일한 경로와 비용이 나오는 것을 확인할 수 있다.

 Microsoft Visual Studio 디버그 콘솔

```
방문할 vertex의 개수: 7
```

```
인접행렬(adjacent matrix) 입력
```

```
0 4 0 0 0 10 0
```

```
3 0 0 18 0 0 0
```

```
0 6 0 0 0 0 0
```

```
0 5 15 0 2 19 6
```

```
0 0 12 1 0 0 0
```

```
0 0 0 0 0 0 10
```

```
0 0 0 8 0 0 0
```

```
최소비용(minimum cost): 51
```

```
Optimal tour의 Path: 1 -> 6 -> 7 -> 4 -> 5 -> 3 -> 2 -> 1
```

## <3 번과 4 번> 소스코드.

<TSP 알고리즘의 소스 코드이다>

```
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
#include<cstdio>
using namespace std;

int adjacent_matrix[10][10]; // 초기 그래프에 대한 인접행렬
int visited[10]; // 방문한 vertex
int v_num; // vertex의 개수
int minimum_cost = 987654321;

// Traveling Salesperson Problem을 풀기위한 데이터를 입력하는 함수
void input() {

    printf("방문할 vertex의 개수: ");
    scanf_s("%d", &v_num); // 방문할 vertex의 개수 입력

    printf("\n인접행렬(adjacent matrix) 입력");
    for (int i = 0; i < v_num; i++) {
        for (int j = 0; j < v_num; j++) {
            scanf_s("%d", &adjacent_matrix[i][j]);
            // 2차원 배열을 이용하여 초기 그래프에 대한 인접행렬 입력
        }
    }
}

void tsp(int start, int i, int sum, int count) {

    // 모든 vertex를 다 탐색하고 vertex가 다시 start로 온 경우
    if (count == v_num && start == i) {
        if (minimum_cost > sum) // minimum_cost보다 계산한 sum의 cost가 더 작은 경우
            minimum_cost = sum; // sum을 minimum_cost로
        return;
    }

    for (int j = 0; j < v_num; j++) {
        if (adjacent_matrix[i][j] == 0)
            // 인접행렬이 0인 경우
            // 1. 그래프가 연결되어있지 않음
            // 2. 시작점과 끝점이 같음
            continue; // 반복문을 빠져나감

        if (!visited[j] && adjacent_matrix[i][j] > 0) {
            visited[j] = 1; // 방문 표시
```

탐색

```
        sum = sum + adjacent_matrix[i][j]; // 인접행렬의 값을 weight에 더함

        if (sum <= minimum_cost) { // sum이 최소비용보다 작을 경우에 탐색
            tsp(start, j, sum, count + 1); // 재귀함수를 통해 깊이 우선
        }

        // 재귀호출을 위해 초기값으로 세팅
        visited[i] = 0; // 방문 표시를 지움
        sum = sum - adjacent_matrix[i][j]; // sum 초기화
    }
}

// depth_first_search 함수를 여러번 호출하여 minimum_cost를 구하는 함수
void travel() {
    for (int i = 0; i < v_num; i++) { // 각각의 vertex에서 시작하는 경우
        tsp(i, i, 0, 0);
    }
}

// 순회하며 minimum cost일 때의 vertex를 기록하는 함수
int travel_vertex(int c) {
    int count, minimum_vertex = 999;
    int minimum = 999, temp;

    for (count = 0; count < v_num; count++) { // 각각의 vertex에서 시작하는 경우
        if ((adjacent_matrix[c][count] != 0) && (visited[count] == 0)) { // weight가
            있고, 방문하지 않은 경우

                if (adjacent_matrix[c][count] < minimum) { // 인접행렬의 weight가
                    munimum보다 작은 경우

                        minimum = adjacent_matrix[count][0] +
adjacent_matrix[c][count]; // minimum 재설정
                    }

                    temp = adjacent_matrix[c][count];
                    minimum_vertex = count;
                }
            }

            if (minimum != 999) { // tour를 하며 minimum값을 재설정 한 경우
                minimum_cost = minimum_cost + temp;
            }

            return minimum_vertex; // 최소비용일 때의 vertex 리턴
        }

// Traveling Salesperson Problem의 정답을 출력
void output() {
    printf("최소비용(minimum cost): %d\n", minimum_cost); // minimum cost 출력
}
```

```

        printf("WnOptimal tour의 Path:");
        Path(0);
        printf("Wn");
    }

// 경로 출력하는 함수
void Path(int vertex) {
    int minimum_vertex; // 최소 비용일 때의 vertex
    visited[vertex] = 1; // 방문 표시

    printf(" %d ->", vertex + 1);

    minimum_vertex = travel_vertex(vertex); // 순회하며 return 한 vertex를 minimum
vertex로 설정

    if (minimum_vertex == 999) {
        // tour를 하지 못한 경우
        minimum_vertex = 0;
        printf(" %d", minimum_vertex + 1);
        minimum_cost = minimum_cost + adjacent_matrix[vertex][minimum_vertex];
        return;
    }

    Path(minimum_vertex); // 재귀적으로 호출하며 경로 출력
}

int main(void) {
    input(); // 초기값 입력
    travel(); // 최소비용 계산
    output(); // 결과값 출력
    return 0;
}

```