컴퓨터네트워크

과제 #04 문제 및 보고서

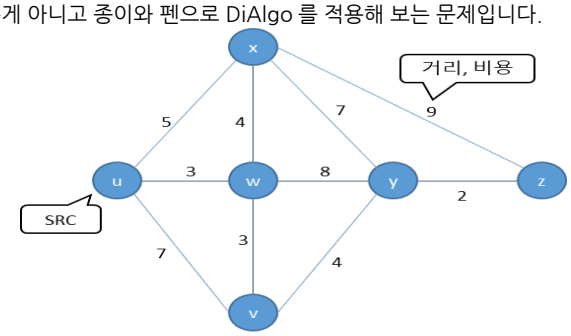
이름	김대욱
학번	202255513
소속 학과/대학	김대욱
분반	061 (담당교수: 김태운)

<Part I> 최단 경로 계산하기

[Q 1] 최단 경로 계산하기 [배점: 20]

강의자료에서 "**Dijkstra's algorithm**" 내용을 참고하여, 아래의 네트워크(=그래프)에 대해 **Dijkstra** 알고리즘을 적용 하시오. (**SRC**: 출발지 노드)

* 이번 문제는, DiAlgo 를 구현하는게 아니고 종이와 펜으로 DiAlgo 를 적용해 보는 문제입니다.



문제 1) 위 그래프에 대해, 강의자료 [참고: **Dijkstra's algorithm**]에 있는 [Table 1, p.50]과 같은 테이블을 완성 하시오([Table 1] DiAlgo 단계별 계산 결과 테이블).

문제 2) 위 그래프에 대해, 강의자료 [참고: **Dijkstra's algorithm**]에 있는 [Table 2, p.50]와 같은 테이블을 완성 하시오([Table 2] 최종 라우팅 테이블).

답변 1)

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxy					12,y
5	uwxyz					

답변 2)

Destination	Link
v	(u,w)
w	(u,w)
x	(u,x)
y	(u,w)
z	(u,w)

<Part II> 최단 경로 구현하기 (C 언어)

Dijkstra 최단 경로 알고리즘을 구현해 보겠습니다. 참고로, 아래의 구현 과정에서 사용하는 표기법 및 알고리즘은 강의자료에 있는 Dijkstra's algorithm 과 약간 다를 수 있습니다. 이번 과제에서 구현하게 될 Dijkstra 최단 경로 알고리즘은 아래와 같습니다('←'기호는 우측의 값을 좌측에 할당하는 연산을 의미합니다). 아래의 [최단 경로 알고리즘]에서 **하늘색**으로 표시된 부분을 하나씩 구현하는 과제입니다. 참고로, 네트워크는 그래프(G)로 표현할 수 있고, 그래프 G 는 정점(= 노드, 네트워크 호스트)과 간선(= 링크, 두 호스트 간 직접 연결)의 집합으로 구성됩니다.

```
입력: 전체 네트워크에 대한 연결 상태 및 비용을 저장한 그래프 G (단, 비용 ≥ 0)
      G 에는 int n(정점의 개수)과 int[][] cost(간선의 비용, 거리)를 저장
      출발지 노드의 인덱스 int v
출력: int distance[n] 배열
참고. distance[u]: 출발지 v 로부터 u 까지 최단 경로의 총 경로 비용
```

```
shortest_path(G, v): // 출발지가 v 인 경우, 모든 노드로의 최소 비용 경로 계산
      S' ← {v}
      For: 각 정점 w ∈ G
            distance[w] ← cost[v][w]; // 초기화

      While: 모든 정점이 집합 S 에 포함될 때 까지
            print_status // 아래에 설명...
            집합 S'에 속하지 않는 정점 중, 최소 distance 를가지는 정점 u 를 choose
            S' ← S' ∪ {u} // u 를 집합 S'에 추가
            For: u 에 인접하고 S'에 없는 각 정점 z 에 대해서
                  If: distance[u] + cost[u][z] < distance[z]
                        distance[z] ← distance[u] + cost[u][z]
```

[Algo. 1] Dijkstra 최단 경로 알고리즘

G 는 네트워크 그래프를 표현한 구조체이며, 2 차원 배열(=매트릭스) 형태로 그래프 구조 및 각 간선에 할당된 비용(=거리)를 표현합니다. 그래프에서 두 노드가 인접(=직접 연결)한 경우에만 0 또는 양수로 표현된 cost(비용)값을 매트릭스 cost[x][y] 위치에 저장합니다. 예를 들어 0 번과 3 번 노드가 직접 연결(=인접)되어 있고, 둘 간의 비용(=거리)이 3 인 경우, G 의 cost[0][3]=3 입니다. 만약, 두 개의 노드가 인접하지 않다면, 둘 간의 비용(=거리)는 무한대(INF)로 표현합니다. 그리고, 자기 자신으로의 거리(비용)는 0 입니다 (즉, cost[i][i] = 0).

[Q 2] Dijkstra 최단 경로 알고리즘 구현 1: 구조체 표현 [배점: 10]

그래프를 저장하기 위한 구조체 만들기: 아래의 구조체는 1) 네트워크에 존재하는 정점/호스트의 수를 나타내는 n 과, 2) 직접 이웃한 (= 인접한) 두 정점 간의 거리 비용을 저장하고 있는 cost 매트릭스로 구성되어 있습니다. 여기서, cost 매트릭스에는 직접 이웃한 정점 간의 거리를 0 또는 양의 정수로, 나머지는 INF 로 거리(비용)를 표현합니다.

```c
#define MAX_NODES 10
typedef struct GraphType {
    int n; // 네트워크에 존재하는 정점의 개수
    int cost[MAX_NODES][MAX_NODES]; // 두 이웃한 정점간 거리/비용
} GraphType;
```

구조체를 활용해서 네트워크 표현하기: 아래의 [그림 1.1] 그래프를 GraphType 구조체로 표현하는 방법은 [그림 1.2]와 같습니다.

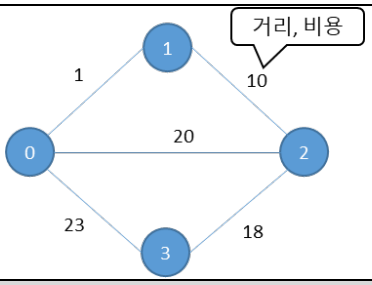
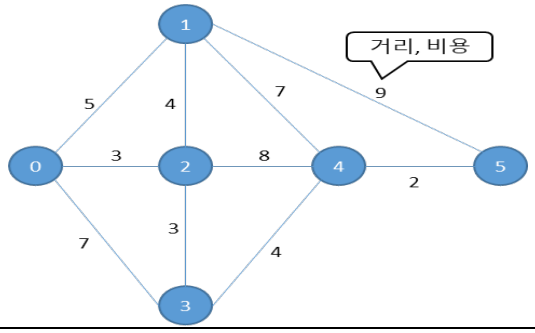


그림 1.1

```c
GraphType G = { 4,
{{0, 1, 20, 23},
{1, 0, 10, INF},
{20, 10, 0, 18},
{23, INF, 18, 0}}
};
```

그림 1.2

문제) 아래의 그래프를 GraphType 구조체로 코딩하세요. [그림 1.2]와 같이 코드를 캡처하고, 아래에 첨부 하시오. 참고로, INF 는 매우 큰 정수로 설정된 매크로 상수 입니다.



답변)

```c
GraphType G = {
    6,
    {
        {0, 5, 3, 7, INF, INF},
        {5, 0, 4, INF, 7, 9},
        {3, 4, 0, 3, 8, INF},
        {7, INF, 3, 0, 4, INF},
        {INF, 7, 8, 4, 0, 2},
        {INF, 9, INF, INF, 2, 0},
    }
};
```

[Q 3] Dijkstra 최단 경로 알고리즘 구현 2: print_status [배점: 10]

최단 경로 프로그램은 아래와 같이 두 개의 배열을 전역 변수로 선언하여 사용합니다.

```
int distance[MAX_NODES]; // 시작 정점으로 부터 각 정점까지의 거리 비용
int found[MAX_NODES]; // 최소 거리 계산이 완료된 정점을 표시
```

'distance' 배열은 출발 정점을 기준으로 각 정점까지 도달하는데 소요되는 거리 비용을 저장하는 1 차원 배열 입니다. 지금까지 알아낸 최단 경로를 기준으로, 각 노드로 가는 최단 경로의 총 비용(거리)를 저장합니다. 'found'는 출발 정점을 기준으로 최소 거리 계산이 완료된 정점을 표시하기 위한 배열이며, 예를 들어 found[u]가 1 이라면 출발지로부터 정점 u 까지의 최소 거리 계산이 완료되었다는 의미입니다 (= S' 집합에 u 가 포함되어 있다는 의미 입니다). 'found' 배열에는 TRUE 또는 FALSE 가 저장되는데, 다음과 같이 매크로 변수를 선언된 상수입니다: #define TRUE 1 그리고 #define FALSE 0

프로그램 실행 시, 아래와 같이 각 STEP 별 결과를 터미널에 출력하는 void print_status (GraphType* g, int step) {...} 함수를 구현 하시오. INF 는 *로 출력하세요. STEP 이란, [Algo. 1]의 while 반복문에서 한번의 iteration 을 의미합니다). 출발 노드가 1 번 노드인 경우 출력은 아래와 같습니다. 출발노드를 1 번 또는 본인이 원하는 번호로 설정하세요.

```
STEP 0
        distance: 1     0     10     *
        found:    0     1     0      0
STEP 1
        distance: 1     0     10     24
        found:    1     1     0      0
STEP 2
        distance: 1     0     10     24
        found:    1     1     1      0
```

문제) print_status 함수의 코드를 캡처하여 아래에 첨부하세요

답변)

```
void print_status(GraphType* g, int step) {
    printf("STEP %d:\n", step);
    printf("\tdistance:\t");
    for (int i = 0; i < g->n; i++) {
        if (distance[i] == INF) {
            printf("*\t");
        } else {
            printf("%d\t", distance[i]);
        }
    }
    printf("\n\t   found:\t");
    for (int i = 0; i < g->n; i++) {
        printf("%d\t", found[i]);
    }
    printf("\n");
}
```

[Q 4] Dijkstra 최단 경로 알고리즘 구현 3: choose [배점: 10]

[Algo. 1]에서 while 문의 매 Iteration 마다, 최단 경로 계산이 완료되지 않은 정점 중에서 최소의 거리 비용을 가진 정점을 선택해야 합니다. 이를 위해 `int choose(int distance[], int n, int found[]) {...}`를 구현하세요. 이 함수는 정점의 인덱스(정수 값)를 리턴합니다.

문제) choose 함수의 코드를 캡처하여 아래에 첨부하세요.

답변)

```
int choose(int distance[], int n, int found[]) {
    int cmp = INF, ans = 0;
    for (int i = 0; i < n; i++) {
        if (found[i] == False && distance[i] < cmp) {
            cmp = distance[i];
            ans = i;
        }
    }
    return ans;
}
```

[Q 5] Dijkstra 최단 경로 알고리즘 구현 4: shortest_path [배점: 10]

최단 경로 계산 프로그램의 main 함수는 다음과 같이 구성되어 있습니다.

```
int start_node_index = 1;
GraphType g = { 4,
{{0, 1, 20, 23},
{1, 0, 10, INF},
{20, 10, 0, 18},
{23, INF, 18, 0}}
};
shortest_path(&g, start_node_index);
```

여기서 `void shortest_path(GraphType* g, int start_node) {...}` 함수는 [Algo. 1]을 구현한 함수입니다. 함수 호출 시, 네트워크 구성 정보가 저장된 GraphType 구조체 포인터, 그리고 시작 정점의 (정수) 인덱스 값을 인자로 전달합니다. 위의 [Algo. 1]을 참고하여 shortest_path 함수를 구현하세요.

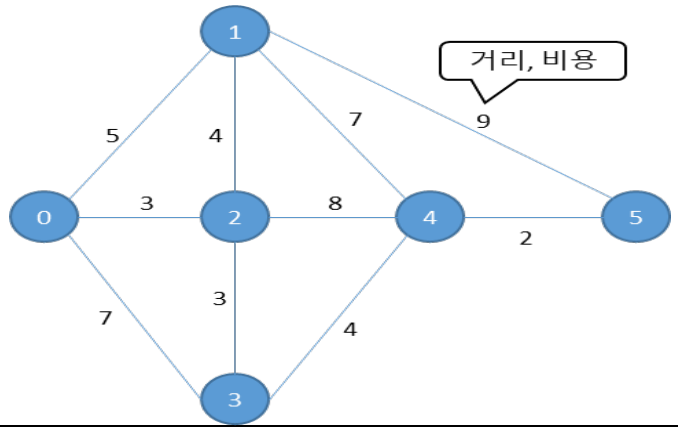
문제) shortest_path 함수 코드를 캡처하여 아래에 첨부하세요.

답변)

```
void shortest_path(GraphType* g, int start_node) {
    found[start_node] = True;
    for (int i = 0; i < g->n; i++) {
        distance[i] = g->cost[start_node][i];
    }
    for (int i = 0; i < g->n; i++) {
        print_status(g, i);
        int now = choose(distance, g->n, found);
        found[now] = True;
        for (int next = 0; next < g->n; next++) {
            if (distance[now] + g->cost[now][next] < distance[next]) {
                distance[next] = distance[now] + g->cost[now][next];
            }
        }
    }
}
```

[Q 6] Dijkstra 최단 경로 알고리즘 테스트 1 [배점: 20]

아래의 그래프로 최단 경로 계산 프로그램을 실행하고, 터미널 출력을 캡처하여 아래에 첨부하세요.
출발 노드는 1 번 또는 본인이 원하는 번호로 설정하세요.

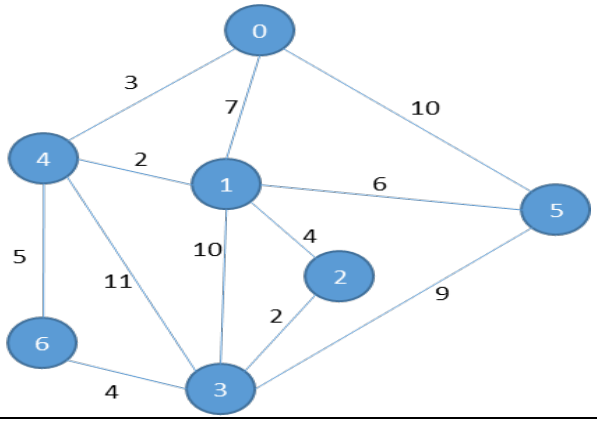


답변)

```
STEP 0:
        distance:       5       0       4       *       7       9
           found:       0       1       0       0       0       0
STEP 1:
        distance:       5       0       4       7       7       9
           found:       0       1       1       0       0       0
STEP 2:
        distance:       5       0       4       7       7       9
           found:       1       1       1       0       0       0
STEP 3:
        distance:       5       0       4       7       7       9
           found:       1       1       1       1       0       0
STEP 4:
        distance:       5       0       4       7       7       9
           found:       1       1       1       1       1       0
STEP 5:
        distance:       5       0       4       7       7       9
           found:       1       1       1       1       1       1
```

[Q 7] Dijkstra 최단 경로 알고리즘 테스트 2 [배점: 20]

아래의 그래프로 최단 경로 계산 프로그램을 실행하고, 터미널 출력을 캡처하여 아래에 첨부하세요.
출발 노드는 1 번 또는 본인이 원하는 번호로 설정하세요.



답변)

```
STEP 0:
        distance:       7       0       4       10      2       6       *
           found:       0       1       0       0       0       0       0
STEP 1:
        distance:       5       0       4       10      2       6       7
           found:       0       1       0       0       1       0       0
STEP 2:
        distance:       5       0       4       6       2       6       7
           found:       0       1       1       0       1       0       0
STEP 3:
        distance:       5       0       4       6       2       6       7
           found:       1       1       1       0       1       0       0
STEP 4:
        distance:       5       0       4       6       2       6       7
           found:       1       1       1       1       1       0       0
STEP 5:
        distance:       5       0       4       6       2       6       7
           found:       1       1       1       1       1       1       0
STEP 6:
        distance:       5       0       4       6       2       6       7
           found:       1       1       1       1       1       1       1
```