Algorithm Study *01.27.2024 김홍주*

#1238.파티 /Gold3

Q. #1238.파티 (Gold 3)

N개의 숫자로 구분된 각각의 마을에 한 명의 학생이 살고 있다.

어느 날 이 **N명의 학생이 X (1 ≤ X ≤ N)번 마을에 모여서** 파티를 벌이기로 했다.

이 마을 사이에는 **총 M개의 단방향 도로**들이 있고 **i번째 길을 지나는데 Ti(1 ≤ Ti ≤ 100)의 시간**을 소비한다.

각각의 학생들은 파티에 참석하기 위해 걸어가서 다시 그들의 마을로 돌아와야 한다. 하지만 이 학생들은 워낙 게을러서 최단 시간에 오고 가기를 원한다.

이 도로들은 단방향이기 때문에 아마 그들이 *오고 가는 길이 다를지도 모른다*.

N명의 학생들 중 **오고 가는데(왕복) 가장 많은 시간을 소비하는 학생**은 누구일지 구하여라.

<입력 조건 >

- 첫째 줄에 **N(1 ≤ N ≤ 1,000), M(1 ≤ M ≤ 10,000), X** 입력
- 다음 M개의 줄에 i번째 도로의 **시작점, 끝점**, 위 도로의 **소요시간 Ti** 입력
- 시작점 = 끝점 같은 도로는 없음
- 시작점과 한 도시 A 에서 다른 도시 B로 가는 도로 개수는 최대 1개
- 모든 N명의 학생은 집에서 X에 갈 수 있고 , X에서 집으로 돌아올 수 있는 데이터만 입력으로 주어진다.

<출력 조건>

- 첫 번째 줄에 N명의 학생들 중
  왕복 시간이 가장 오래 걸리는 학생의 소요시간을 출력한다.

Q. #1238.파티 (Gold 3)

N개의 숫자로 구분된 각각의 마을에 한 명의 학생이 살고 있다.

어느 날 이 **N명의 학생이 X (1 ≤ X ≤ N)번 마을에 모여서** 파티를 벌이기로 했다.

이 마을 사이에는 **총 M개의 단방향 도로**들이 있고 **i번째 길을 지나는데 Ti(1 ≤ Ti ≤ 100)의 시간**을 소비한다.

각각의 학생들은 파티에 참석하기 위해 걸어가서 다시 그들의 마을로 돌아와야 한다. 하지만 이 학생들은 워낙 게을러서 최단 시간에 오고 가기를 원한다.

이 도로들은 단방향이기 때문에 아마 그들이 *오고 가는 길이 다를지도 모른다*.

N명의 학생들 중 **오고 가는데(왕복) 가장 많은 시간을 소비하는 학생**은 누구일지 구하여라.

<입력 조건 >

- 첫째 줄에 **N(1 ≤ N ≤ 1,000), M(1 ≤ M ≤ 10,000), X** 입력
- 다음 M개의 줄에 i번째 도로의 **시작점, 끝점**, 위 도로의 **소요시간 Ti** 입력
- 시작점 = 끝점 같은 도로는 없음
- 시작점과 한 도시 A 에서 다른 도시 B로 가는 도로 개수는 최대 1개
- 모든 N명의 학생은 집에서 X에 갈 수 있고 , X에서 집으로 돌아올 수 있는 데이터만 입력으로 주어진다.

<출력 조건>

- 첫 번째 줄에 N명의 학생들 중
  왕복 시간이 가장 오래 걸리는 학생의 소요시간을 출력한다.

[예제]

```
# 입력
4 8 2 # N, M , X
1 2 4 #시작, 끝 , 소요시간
1 3 2
1 4 7
2 1 1
2 3 5
3 1 2
3 4 4
4 2 3
```

```
# 출력
10
```

Hint

0. 왕복 최단 거리 계산 : MAX({ N마을 -> 파티장} + { 파티장 X ->N 마을})

1. 다익스트라 (Dijkstra) *# 최단거리 , # 그리디 , # 우선순위 큐*

- 그래프에서 한 정점(X)에서 다른 정점(A,B,C..) 까지 가는 각각의 최단 경로 구하는 방법
- **동작 과정**

1. 출발 노드 설정 : X
2. 최단 거리 테이블 초기화 *distance([1,2,3 .. n]) = INF

노드 번호 N	1(start)	2	3	4	…
최단 거리	0	INF	INF	INF	

*각 노드에 대한 최단 거리를 담은 1차원 리스트

3. 노드들 중

   (1) 방문 하지 않았고

   (2) 현재 가장 거리가 짧은 노드를 선택

   - 구현 방법 : 완전 탐색(O(V^2)) / **우선순위 큐 by heapque(O(ElogV))**

     : min(Cost(X -> a,b,c…)) & not visited => a 선택

4. 출발점(X)에서 위 노드(a) 를 경유해서 다른(인접) 노드 (Y) 로 가는 거리 계산 => 최단 거리 테이블 업데이트

   : *Cost(X->a -> Y) < Cost(X->Y) => update!*

Hint

1-2. 다익스트라 (Dijkstra) *# 최단거리 , # 그리디 , # 우선순위 큐*

```python
import heapq

def dijkstra(s):
    #2. 최단거리 테이블 초기화
    D = [float('inf')] * (N+1)
    D[s] = 0
    q = []                      # 최단 거리 테이블을 heap으로 구현
    heapq.heappush(q, (0, s))       # heap에 (가중치, 노드) 형식으로 삽입

    #3. 현재 출발점과 가장 가깝고 ,방문 안한 노드(경유지) 찾기
    while q:
        dist, now = heapq.heappop(q)    # 최소힙이므로 가중치가 가장 작은 값이 pop
        if D[now] >= dist:          # 이미 최솟값 구했는지 확인(방문여부확인)

        #4. 인접한 노드 중 now를 경유할 때 더 작은 값이면 최단거리 테이블 갱신 & 큐 삽입
            for v, val in city[now]:    # 연결된 노드(Y)들 확인
                if dist + val < D[v]:   # 경유 방법이 가중치가 더 작은 값이면 갱신
                    D[v] = dist + val
                    heapq.heappush(q, (dist + val, v)) # 큐에 삽입
    return D

dijkstra(start) # 1.출발점 설정
```

- 그래프에서 한 정점(X)에서 다른 정점(A,B,C..) 까지 가는 각각의 최단 경로 구하는 방법
- **동작 과정**

1. 출발 노드 설정 : X *distance([1,2,3 .. n]) = INF
2. 최단 거리 테이블 초기화

노드 번호 N	1(start)	2	3	4	...
최단 거리	0	INF	INF	INF	

*각 노드에 대한 최단 거리를 담은 1차원 리스트

3. 현재 노드와 연결된 노드 중
   (1) 방문 하지 않았고
   (2) 가장 거리가 짧은 노드를 선택
   - 구현 방법 : **우선순위 큐 by heapque(O(ElogV))**
4. 현재 노드(X)가 위 노드(a) 를 경유해서 다른(인접) 노드 (Y) 로 가는 거리 계산
   => 최단 거리 테이블 업데이트

My Solution

```python
import sys
import heapq
# 1. 인접 리스트 field 만들기 (단방향)
input = sys.stdin.readline
INF = int(1e9) #1<=Time<=100
N , M , start_town = map(int , input().split()) # 시작 note
# 각 road (edge)와 Time  정복가 담길 리스트 만들기
field = [[] for _ in range(N+1) ] # idx : 1~ N+1
#무한으로 최단 거리 테이블 초기화
to_X_distance = [ [INF for _ in range(N+1)] for k in range(N+1)]
# 1-2. 모든 road 및  time 정보 받기

for m in range(M) :
    start , end , time = map(int, input().split())
    field[start].append((end, time))


# 다익스트라
def dijkstra(start , distance ) :
    q = [] # 우선순위 큐
    #1. 시작 노드에 대해 최단경로 = 0 , 큐 삽입(시간 = 0 , 노드)
    heapq.heappush(q , (0,start))
    distance[start] = 0
    #2. q가 비어 있기 전까지
    while q :
        # 가장 최단 거리 짧은 노드에 대한 정보 추출
        time , now  = heapq.heappop(q) # A -> now(중간)
        # 현재 노드가 이미 처리 = 방문 여부 확인
        if distance[now] < time :
            continue
        # 현재 노드와 연결된 다른 인접 노드 확인
        for near_road  , near_time in field[now] :
            duration = near_time + time #  A -> now -> B
            #기존 방법 보다  현재 노드 경유해서 갈때 시간이 적게 걸릴때
            # 큐 삽입 & 최단거리 테이블 업데이트
            if duration < distance[near_road] :
                distance[near_road] = duration # 업데이트
                heapq.heappush(q , (duration , near_road)) # 큐에 넣기

# 다익스트라 수행
for i in range(1,N+1) :
    to_X_distance[i] = dijkstra(i,to_X_distance[i] )

# result[N] = distance( N -> X ) + distance(X-> N) # N의 왕복 최단 거리
result = [0 for _ in range(N+1)]
for town in range(1,N+1):
    result[town] = to_X_distance[town][start_town] + to_X_distance[start_town][town]


print(max(result))
```

1. 인접 리스트 (단방향) 만들기
2. 최단 거리 테이블 초기화

: **1차 => 2차 리스트**

3. 다익스트라

: **Start 마을 -> N개 마을의 최단 거리 테이블 반환**

My Solution

```python
import sys
import heapq
# 1. 인접 리스트 field 만들기 (단방향)
input = sys.stdin.readline
INF = int(1e9) #1<=Time<=100
N , M , start_town = map(int , input().split()) # 시작 note
# 각 road (edge)와 Time 정복가 담길 리스트 만들기
field = [[] for _ in range(N+1) ] # idx : 1~ N+1
#무한으로 최단 거리 테이블 초기화
to_X_distance = [ [INF for _ in range(N+1)] for k in range(N+1)]
# 1-2. 모든 road 및 time 정보 받기

for m in range(M) :
    start , end , time = map(int, input().split())
    field[start].append((end, time))


# 다익스트라
def dijkstra(start , distance ) :
    q = [] # 우선순위 큐
    #1. 시작 노드에 대해 최단경로 = 0 , 큐 삽입(시간 = 0 , 노드)
    heapq.heappush(q , (0,start))
    distance[start] = 0
    #2. q가 비어 있기 전까지
    while q :
        # 가장 최단 거리 짧은 노드에 대한 정보 추출
        time , now  = heapq.heappop(q) # A -> now(중간)
        # 현재 노드가 이미 처리 = 방문 여부 확인
        if distance[now] < time :
            continue
        # 현재 노드와 연결된 다른 인접 노드 확인
        for near_road  , near_time in field[now] :
            duration = near_time + time #  A -> now -> B
            #기존 방법 보다  현재 노드 경유해서 갈때 시간이 적게 걸릴때
            # 큐 삽입 & 최단거리 테이블 업데이트
            if duration < distance[near_road] :
                distance[near_road] = duration # 업데이트
                heapq.heappush(q , (duration , near_road)) # 큐에 넣기
```

```python
# 다익스트라 수행
for i in range(1,N+1) :
    to_X_distance[i] = dijkstra(i,to_X_distance[i] )

# result[N] = distance( N -> X ) + distance(X-> N) # N의 왕복 최단 거리
result = [0 for _ in range(N+1)]
for town in range(1,N+1):
    result[town] = to_X_distance[town][start_town] + to_X_distance[start_town][town]


print(max(result))
```

4. N -> N 의 최단 거리 테이블 업데이트 (N-1 다익스트라)

Town(X=2)	0	1	2	3	4
0					
1 -> n		0	4	2	6
2-> n		1	0	3	7
3-> n		2	6	0	4
4 -> n		4	3	6	0

N ->X : 집에서 파티장까지 최단 거리

X -> N : 집 도착 최단 거리

	1	3	4
N->X	4	6	3
X->N	1	3	7
왕복	5	9	10

다른 Solution 1.

: 역방향 그래프 구축해서 다익스트라 2번 사용하기(Pypy3: 296m)

```python
import sys
import heapq
INF = 1e10
def input(): return sys.stdin.readline().rstrip()

def dijkstra(s, edge):
    dist = [INF] * (n+1)
    q = []
    heapq.heappush(q,(s,0))
    while q:
        w,d = heapq.heappop(q)
        if dist[w] < d:
            continue
        for nxt,c in edge[w]:
            if dist[nxt] > d + c:
                dist[nxt] = d+c
                heapq.heappush(q,(nxt,d+c))
    return dist

n,m,x = map(int,input().split())
graph = [[] for _ in range(n+1)]
reverse_graph = [[] for _ in range(n+1)]
for _ in range(m):
    a,b,c = map(int,input().split())
    graph[a].append((b,c))
    reverse_graph[b].append((a,c))
node2x = dijkstra(x,reverse_graph)
x2node = dijkstra(x,graph)

print(max([x2node[i] + node2x[i] for i in range(1,n+1) if i != x]))
```

다른 Solution 2.

: 다익스트라 **N-1번 & 1차원 최단거리 테이블 사용**

```python
import heapq

def dijkstra(s):
    D = [float('inf')] * (N+1)
    D[s] = 0
    q = []                      # 최단 거리 테이블을 heap으로 구현
    heapq.heappush(q, (0, s))       # heap에 (가중치, 노드) 형식으로 삽입
    while q:
        dist, now = heapq.heappop(q)    # 최소힙이므로 가중치가 가장 작은 값이 pop
        if D[now] >= dist:          # 이미 최솟값 구했는지 확인
            for v, val in city[now]:    # 연결된 노드들 확인
                if dist + val < D[v]:   # 가중치가 더 작은 값이면 갱신
                    D[v] = dist + val
                    heapq.heappush(q, (dist + val, v))
    return D

N, M, X = map(int, input().split())
city = [[] for _ in range(N+1)]
for _ in range(M):
    a, b, t = map(int, input().split())
    city[a].append([b, t])

# 1. X -> N 마을로 돌아가는 최단 거리 테이블
ans = dijkstra(X)
ans[0] = 0

#2.왕복 최단 거리 구하기
for i in range(1, N+1):
    if i != X:
        res = dijkstra(i) # N(X 제외) -> 최단 거리
        ans[i] += res[X] # 그 중 N -> X 가는 길 더함

print(max(ans))
```

Assignment

[백준#4480 녹색 옷 입은 애가 젤다지?] Gold4

- 문제 : https://www.acmicpc.net/problem/4485

#다익스트라