**최단 경로 알고리즘**

- 다이내믹 프로그래밍을 활용한 대표적인 최단 경로 탐색 알고리즘.

- 인공위성 GPS, 소프트웨어 등에서 가장 많이 사용됨.

- 특정한 하나의 정점에서 다른 모든 정점으로 가는 최단 경로를 알려줌.

- 음의 간선은 포함 할 수 없기에, 현실세계에 사용하기 매우 적합한 알고리즘.

- 동적 프로그래밍인 이유는 최단 거리는 여러 개의 최단 거리로 이루어져 있기 때문임.

- 작은 문제가 큰 문제의 부분 집합에 속해 있다고 볼 수 있음.

- 기본적으로 다익스트라는 하나의 최단 거리를 구할 때 그 이전까지 구했던 최단거리 정보를 그대로 사용한다는 특징이 있다. 또한, 다익스트라 알고리즘은 현재까지 알고 있던 최단 경로를 계속해서 갱신한다.

**다익스트라 알고리즘의 작동과정**

1. 출발 노드를 설정한다.
2. 출발 노드를 기준으로 각 노드의 최소비용을 저장한다.
3. 방문하지 않은 노드 중에서 가장 비용이 적은 노드를 선택한다.
4. 해당 노드를 거쳐서 특정한 노드로 가는 경우를 고려하여 최소비용을 갱신한다.
5. 위 과정에서 3번~4번을 반복한다.

***실제로 컴퓨터 안에서 문제를 해결할 시에는 이차원배열 형태로 처리해야한다. 특정 행에서 열로가는 비용을 계산하는 것이 바로 그것이다.***

그러나, heap의 개념을 알지 못한 채로 알고리즘을 구현하면 매번 최단 거리가 가장 짧은 노드를 선형 탐색해야하고, 현재 노드와 연결된 노드를 매번 일일이 확인해야한다. 노드의 개수가 10,000개가 넘어가면 간선이 두개만 된다고 하더라도 1만번을 1만번 확인해야한다. 그러나, 힙 자료구조를 이용하면 특정 노드까지의 최단 거리에 대한 정보를 힙에 담아 처리하므로 출발 노드부터 가장 거리가 짧은 노드를 더 빠르게 찾을 수 있다.

**힙(heap)이란?**

* 우선순위 큐 기능을 구현하고자 할 때 사용되는 자료구조 중 하나이다.
* 파이썬에서는 힙 기능을 위해 heapq 라이브러리를 제공한다.
* 파이썬의 힙은 최소 힘으로 구성되어 있으므로 단순히 원소를 힙에 전부 넣었다가 빼는 것만으로도 시간 복잡도에 오름차순 정렬이 완료됨.

**최소힙과 최대힙**

* 최소힙은 값이 낮은 데이터가 먼저 삭제되고, 최대힙은 값이 큰 데이터가 먼저 삭제된다.
* 파이썬에서는 최대힙을 제공하지 않는다. Heapq 라이브러리를 이용해 최대힙을 구현해야 할 때는 원소의 부호를 임시로 변경하는 방식을 사용한다.

**최단 경로 문제 소스코드**

import sys

input = sys.stdin.readline

sys.setrecursionlimit(10\*\*6)

import heapq

# 정점의 개수 n, 간선의 개수 m

n, m = map(int, input().split())

# 시작 정점의 번호

k = int(input())

# 무한을 의미하는 INF

INF = int(1e9)

# 그래프 초기화

graph = [[] \* (n+1) for \_ in range(n+1)]

# 최단 거리 테이블을 모두 무한으로 초기화

distance = [INF] \* (n+1)

# 간선 정보 입력

for \_ in range(m):

    a, b, c = map(int, input().split())

    # a->b가 c비용

    graph[a].append((b, c))

def dijkstra(start):

    q = []

    # 시작 노드로 가기 위한 최단 경로는 0으로 설정하여, 큐에 삽입

    heapq.heappush(q, (0, start))

    distance[start] = 0

    while q:

        # 가장 최단 거리가 짧은 노드에 대한 정보 꺼내기

        dist, now = heapq.heappop(q)

        # 현재 노드가 이미 처리된 적이 있는 노드라면 무시

        if distance[now] < dist:

            continue

        # 현재 노드와 연결된 다른 인접한 노드들을 확인

        for i in graph[now]:

            cost = dist + i[1]

            # 현재 노드를 거쳐서, 다른 노드로 이동하는 거리가 더 짧은 경우

            if cost < distance[i[0]]:

                distance[i[0]] = cost

                heapq.heappush(q, (cost, i[0]))

# 다익스트라 알고리즘을 수행

dijkstra(k)

# 모든 노드로 가기 위한 최단 거리를 출력

for i in range(1, n+1):

    if distance[i] == INF:

        print("INF")

    else:

        print(distance[i])