다익스트라

다익스트라란

다익스트라는 한 노드를 기점으로
다른 노드사이의 최단 거리를
구하는 알고리즘 입니다

과정

1. 출발 노드를 설정

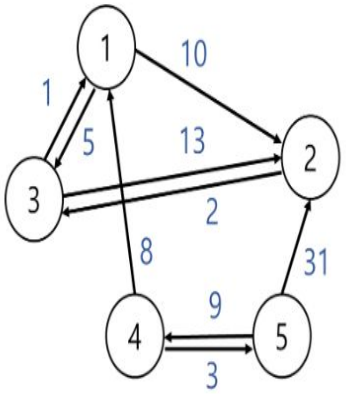
2. 최단 거리 테이블 초기화

3. 방문하지 않은 노드 중에서 최단 거리가 가장 짧은 노드 선택
(이 과정 때문에 우선순위 큐를 사용하는 것이 시간적 이득)

4. 해당 노드를 거쳐 다른 노드로 가는 비용을 계산 후 테이블 갱신

5. 위 3, 4과정 반복

nowVertex = pq = []



	[1]	[2]	[3]	[4]	[5]
check	F	F	F	F	F

	[1]	[2]	[3]	[4]	[5]
dist	∞	∞	∞	∞	∞

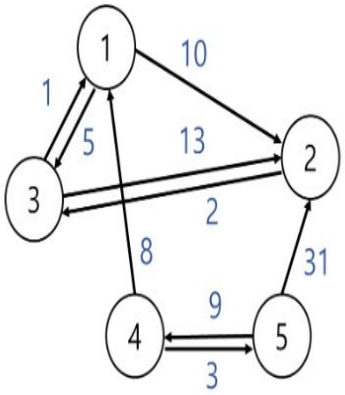
설명에 사용되는 변수들

우선순위 큐: pq

방문 여부 체크 변수: check

각 노드간의 최단 거리: dist

현재 노드를 저장하는 변수: nowVertex



nowVertex = pq = []

	[1]	[2]	[3]	[4]	[5]
check	F	F	F	F	F

	[1]	[2]	[3]	[4]	[5]
dist	∞	∞	∞	∞	∞

다익스트라 시작 전 변수 초기화

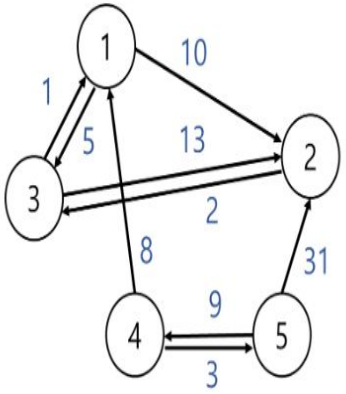
각 노드간의 거리를 저장하는 배열은

거리를 무한으로 초기화 해놓습니다

후에 다익스트라를 진행하면서

dist배열을 계속해서 갱신 할 예정입니다

nowVertex = pq = []



	[1]	[2]	[3]	[4]	[5]
check	F	F	F	F	F

	[1]	[2]	[3]	[4]	[5]
dist	∞	∞	∞	∞	∞

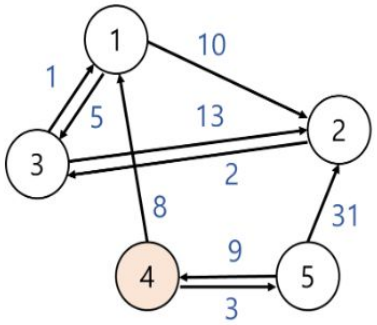
만약 거리가 기존 값보다 큰 경우 혹은

이미 방문한적 있는 경우에는

우선순위 큐에 값을 넣지 않습니다

1) 출발지 4를 우선순위 큐에 넣는다. 출발지이므로 거리는 0이다.

nowVertex = 0 pq = [(4, 0)]



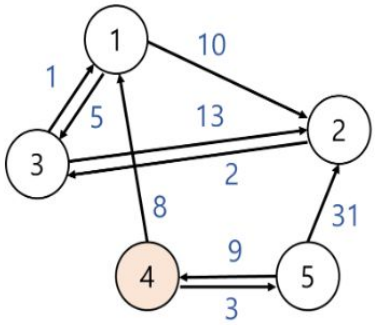
	[1]	[2]	[3]	[4]	[5]
check	F	F	F	T	F

	[1]	[2]	[3]	[4]	[5]
dist	∞	∞	∞	0	∞

1. 처음 시작 노드를 4번으로 설정
2. 4번노드를 방문했다고 변경
3. 거리는 자기 자신이므로 0으로 처리

1) 출발지 4를 우선순위 큐에 넣는다. 출발지이므로 거리는 0이다.

nowVertex = 0 pq = [(4, 0)]



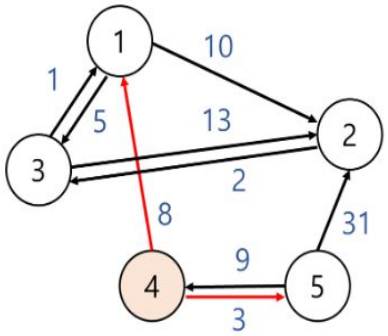
	[1]	[2]	[3]	[4]	[5]
check	F	F	F	T	F

	[1]	[2]	[3]	[4]	[5]
dist	∞	∞	∞	0	∞

1. 처음 시작 노드를 4번으로 설정
2. 4번노드를 방문했다고 변경
3. 거리는 자기 자신이므로 0으로 처리
4. 우선순위 큐에 노드와 거리 저장

2) 우선순위 큐에서 하나 꺼내 `nowVertex`에 저장하고 방문체크를 한다. `nowVertex`을 거쳐 갈 수 있는 정점의 거리가 이전 기록한 값보다 적으면 갱신한다.

nowVertex = 4 pq = [(5, 3), (1, 8)]



	[1]	[2]	[3]	[4]	[5]
check	F	F	F	T	F

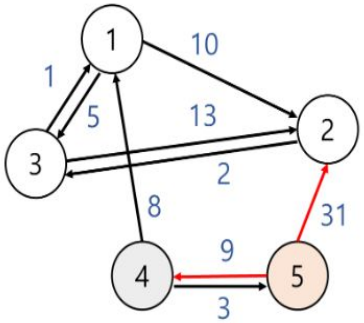
	[1]	[2]	[3]	[4]	[5]
dist	8	∞	∞	0	3

1. 우선 순위 큐에서 값을 하나 빼서 현재 노드를 nowVertex저장
2. 현재 노드와 연결되어 있는 노드들의 거리 값을 갱신
3. 우선 순위 큐에 2개의 노드를 저장한다
4. 이때 우선순위 큐에서는 노드의 거리순으로 우선순위를 정한다

여기까지 1회전

3) 우선순위 큐에 값이 있으므로 하나 꺼내 `nowVertex`에 저장하고 방문 처리한다.

nowVertex = 5 pq = [(1, 8), (2, 34)]



	[1]	[2]	[3]	[4]	[5]
check	F	F	F	T	T

	[1]	[2]	[3]	[4]	[5]
dist	8	34	∞	0	3

1. 우선 순위 큐에서 값을 하나 빼서 현재 노드를 nowVertex저장(방문)

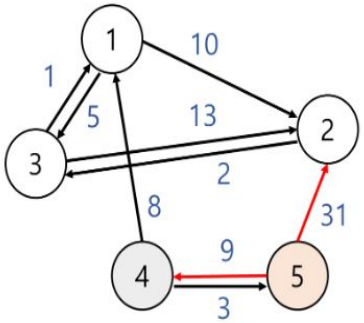
2. 현재 노드(5)와 연결되어 있는 노드들의 거리 값을 갱신

3. 우선 순위 큐에 인접 노드 집어넣기

4. 이때 우선순위 큐에서는 노드의 거리순으로 우선순위를 정한다

3) 우선순위 큐에 값이 있으므로 하나 꺼내 `nowVertex`에 저장하고 방문 처리한다.

nowVertex = 5 pq = [(1, 8), (2, 34)]



	[1]	[2]	[3]	[4]	[5]
check	F	F	F	T	T

	[1]	[2]	[3]	[4]	[5]
dist	8	34	∞	0	3

이때 2번까지의 거리는 4->5까지의

거리3과 5->2까지의 거리31을 더해서

3+31 = 34이므로

2번 노드까지의 거리를 34로 갱신해준다

이 부분에서 알 수 있는 점

노드끼리의 거리를 구할때 중간에 노드가

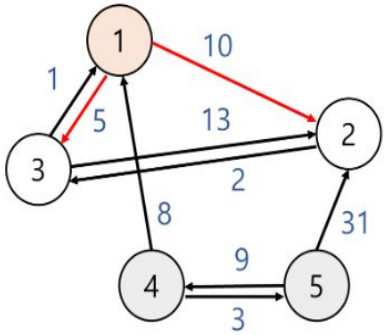
있을 경우 그 노드와 전 노드의 거리

정보를 참고해서 구하게 된다

여기까지 2회전

4) 우선순위 큐에서 값을 꺼낸다.

nowVertex = 1 pq = [(3, 13), (2, 18), (2, 34)]



	[1]	[2]	[3]	[4]	[5]
check	T	F	F	T	T

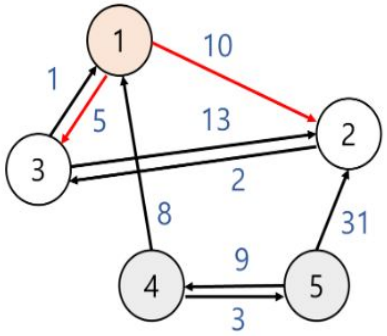
	[1]	[2]	[3]	[4]	[5]
dist	8	18	13	0	3

1. 우선 순위 큐에서 값을 하나 빼서 현재 노드를 nowVertex저장(방문)

2. 현재 노드(1)와 연결되어 있는 노드들의 거리 값을 갱신

4) 우선순위 큐에서 값을 꺼낸다.

nowVertex = 1 pq = [(3, 13), (2, 18), (2, 34)]



	[1]	[2]	[3]	[4]	[5]
check	T	F	F	T	T

	[1]	[2]	[3]	[4]	[5]
dist	8	18	13	0	3

이때 2번노드를 확인하면 4->5->2순으로

방문한 거리값이 3+31 = 34이므로

2번노드 까지의 거리를 34로 저장했지만

4->1->2 순으로 방문한 거리 값이

8+10 = 18이므로 18<34여서

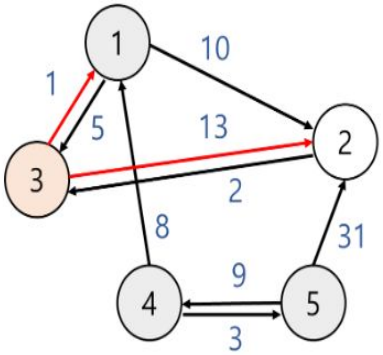
최소 거리를 18로 갱신한다

그 이후에 3번노드의까지의 거리도

위 방법과 동일하게 갱신한다

5) 우선순위 큐에서 값을 꺼낸다.

nowVertex = 3 pq = [(2, 18), (2, 34)]



	[1]	[2]	[3]	[4]	[5]
check	T	F	T	T	T

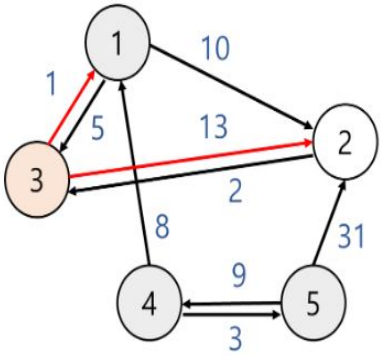
	[1]	[2]	[3]	[4]	[5]
dist	8	18	13	0	3

1. 우선 순위 큐에서 값을 하나 빼서 현재 노드를 nowVertex저장(방문)

2. 현재 노드(3)와 연결되어 있는 노드들의 거리 값을 갱신

5) 우선순위 큐에서 값을 꺼낸다.

nowVertex = 3 pq = [(2, 18), (2, 34)]



	[1]	[2]	[3]	[4]	[5]
check	T	F	T	T	T

	[1]	[2]	[3]	[4]	[5]
dist	8	18	13	0	3

3번 노드와 인접한 노드들의 거리가

3번 노드를 거치지 않는 것이

더 최솟값이므로 우선순위 큐에

값을 저장하지 않습니다

4→3→1→2 : 8+5+13 = 26

4→1→2 : 8+10 = 18

18<26 따라서 갱신을 하지 않습니다

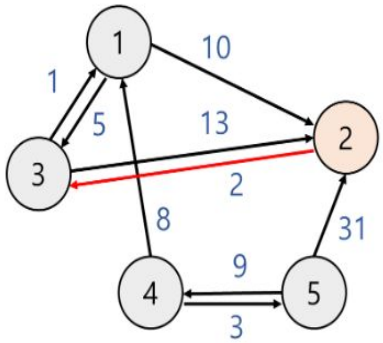
마찬가지로 1번 노드 역시 갱신하지

않고 우선순위 큐에 값을

저장하지 않습니다

6) 우선순위 큐에서 값을 꺼낸다.

nowVertex = 2 pq = [(2, 34)]



	[1]	[2]	[3]	[4]	[5]
check	T	T	F	T	T

	[1]	[2]	[3]	[4]	[5]
dist	8	18	13	0	3

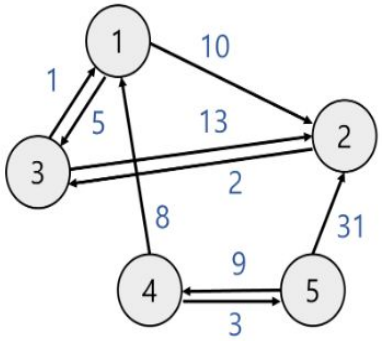
1. 우선 순위 큐에서 값을 하나 빼서 현재 노드를 nowVertex저장(방문)

2. 현재 노드와 연결되어 있는 노드들의 거리 값을 갱신

이 경우 역시 2번노드에서 다른 노드로 향하는 거리가 최솟값이 없으므로 거리 갱신 x, 우선순위 큐 삽입 x

7) 우선순위 큐에서 값을 꺼낸다.

nowVertex = 2 pq = [ ]



	[1]	[2]	[3]	[4]	[5]
check	T	T	F	T	T

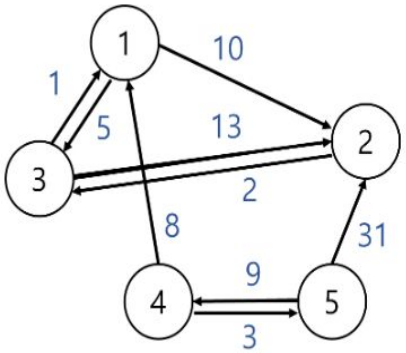
	[1]	[2]	[3]	[4]	[5]
dist	8	18	13	0	3

우선 순위 큐에서 값을 하나를

뺐지만 이미 방문한 노드이므로

그냥 넘긴다

8) 우선순위 큐가 비었으므로 다익스트라 알고리즘을 종료한다. 정점 4에서 출발하여 다른 정점까지 최소 거리는 다음 dist 배열과 같다.



	[1]	[2]	[3]	[4]	[5]
dist	8	18	13	0	3

최종적으로 완성된 최단 거리 배열

유의사항

1. 다익스트라는 음의 가중치가 있을 경우
   작동하지 않습니다

2. 인접행렬로 구현된 다익스트라는
   시간 복잡도가 O(N^2), 우선순위 큐로 구현된
   다익스트라는 시간복잡도가 O(MlogN)입니다
   (M =간선의 개수, N = 노드)

유의사항

3. 설명시에는 check배열을 만들어서 방문여부를 확인했지만

배열이 없이도 확인할 수 있는 방법이 있습니다

이 방법은 다음에 나오는 코드에서 확인하면 될것 같습니다

설명할 문제

1916번 | 제출 | 맞힌 사람 | 숏코딩 | 재채점 결과 | 채점 현황 | 내 제출 | 난이도 기여 | 강의 | 질문 게시판 (140+) | 타이머

최소비용 구하기

성공

골드 V

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
0.5 초	128 MB	69241	21777	14265	32.194%

문제

N개의 도시가 있다. 그리고 한 도시에서 출발하여 다른 도시에 도착하는 M개의 버스가 있다. 우리는 A번째 도시에서 B번째 도시까지 가는데 드는 버스 비용을 최소화 시키려고 한다. A번째 도시에서 B번째 도시까지 가는데 드는 최소비용을 출력하여라. 도시의 번호는 1부터 N까지이다.

입력

첫째 줄에 도시의 개수 N(1 ≤ N ≤ 1,000)이 주어지고 둘째 줄에는 버스의 개수 M(1 ≤ M ≤ 100,000)이 주어진다. 그리고 셋째 줄부터 M+2줄까지 다음과 같은 버스의 정보가 주어진다. 먼저 처음에는 그 버스의 출발 도시의 번호가 주어진다. 그리고 그 다음에는 도착지의 도시 번호가 주어지고 또 그 버스 비용이 주어진다. 버스 비용은 0보다 크거나 같고, 100,000보다 작은 정수이다.

그리고 M+3째 줄에는 우리가 구하고자 하는 구간 출발점의 도시번호와 도착점의 도시번호가 주어진다. 출발점에서 도착점을 갈 수 있는 경우만 입력으로 주어진다.

출력

첫째 줄에 출발 도시에서 도착 도시까지 가는데 드는 최소 비용을 출력한다.

참고 출처

출처1

출처 2