노트북: 첫 번째 노트북
만든 날짜: 2020-11-18 오후 10:56
URL: https://continuous-development.tistory.com/195

Algorithm

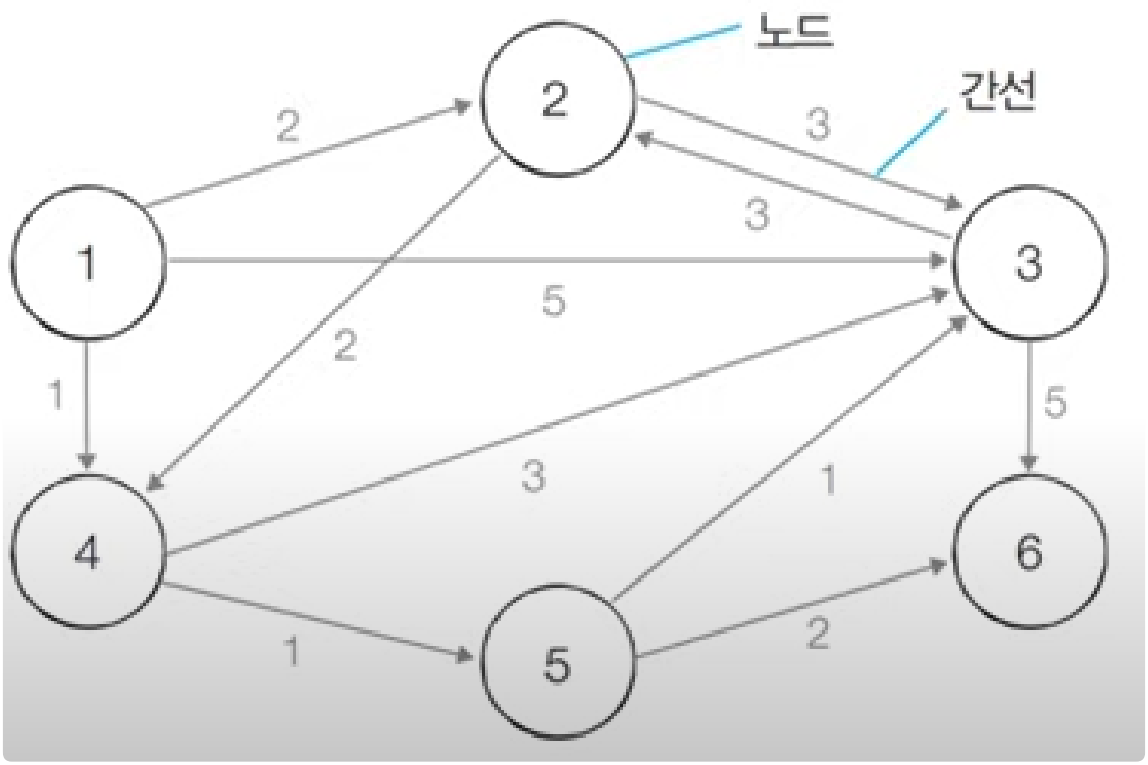
[Algorithm] 30강 : 다익스트라 최단 경로 알고리즘

2020. 11. 18. 22:56 수정 삭제 공개

최단경로 문제

1.1 최단 경로 문제란?

- 최단 경로 알고리즘은 가장 짧은 경로를 찾는 알고리즘을 의미
- 다양한 문제 상황
  - 한 지점에서 다른 한 지점까지의 최단 경로
  - 한 지점에서 다른 모든 지점까지의 최단 경로
  - 모든 지점에서 다른 모든 지점까지의 최단 경로
- 각 지점은 그래프에서 노드로 표현
- 지점 간 연결된 도로는 그래프에서 간선으로 표현



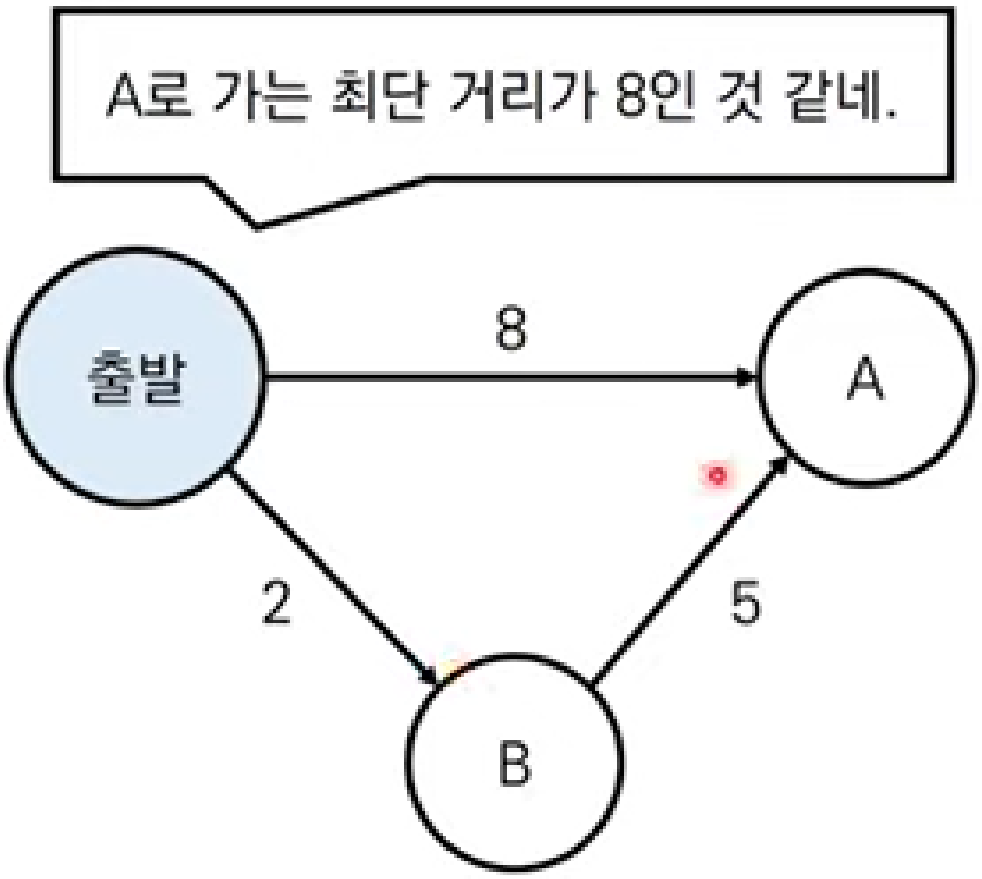
1.2 다익스트라 최단 경로 알고리즘 개요

- 특정한 노드에서 출발하여 다른 모든 노드로 가는 최단 경로를 계산
- 다익스트라 최단 경로 알고리즘은 음의 간선이 없을 때 정상적으로 동작
  - 현실 세계의 도로(간선)는 음의 간선으로 표현되지 않는다.
- 다익스트라 최단 경로 알고리즘은 **그리디 알고리즘**으로 분류
  - 매 상황에서 가장 비용이 적은 노드를 선택해 임의의 과정을 반복한다.

1.3 알고리즘 동작 과정

1. 출발 노드를 설정
2. 최단 거리 테이블을 초기화
3. 방문하지 않은 노드 중에서 최단 거리가 가장 짧은 노드를 선택
4. 해당 노드를 거쳐 다른 노드로 가는 비용을 계산하여 최단 거리 테이블을 갱신
5. 위 과정에서 3번과 4번을 반복

알고리즘 동작 과정에서 최단 거리 테이블은 각 노드에 대한 현재까지의 최단 거리 정보를 가지고 있습니다.
처리과정에서 더 짧은 경로를 찾으면 경로를 갱신합니다.



1.4 다익스트라 알고리즘의 특징

- 그리디 알고리즘: 매 상황에서 방문하지 않은 가장 비용이 적은 노드를 선택해 임의의 과정을 반복
- 단계를 거치며 한 번 처리된 노드의 최단 거리는 고정되어 더 이상 바뀌지 않는다.
    - 한 단계당 하나의 노드에 대한 최단 거리를 확실히 찾는 것으로 이해할 수 있다.
- 다익스트라 알고리즘을 수행한 뒤에 테이블에 각 노드까지의 최단 거리 정보가 저장된다.
    - 완벽한 형태의 최단 경로를 구하려면 소스코드에 추가적인 기능을 더 넣어야 한다.
- 단계마다 방문하지 않은 노드 중에서 최단 거리가 가장 짧은 노드를 선택하기 위해 매 단계마다 1차원 테이블의 모든 원소를 확인(순차 탐색) 한다.

1.5 다익스트라 알고리즘 구현

```
import sys
input = sys.stdin.readline

INF = int(1e9) # 무한을 의미하는 값으로 10 억을 설정

# 노드의 개수, 간선의 개수를 입력받기
n,m = map(int, input().split())

#시작 노드번호를 입력받기
start = int(input())

#각 노드에 연결되어 있는 노드에 대한 정보를 담는 리스트를 만들기
graph [[]for in range(n+1)]

# 방문한 적이 있는지 체크하는 목적의 리스트를 만들기
visited = [False] * (n+1)

#최단 거리 테이블을 모두ㅡ 무한으로 초기화
distance = [INF] * (n+1)

# 모든 간선 정보를 입력 받기
for _in range(m):
    a,b,c = map(int, input().split())
    # a 번 노드에서 b 번 노드로 가는 비용이 c 라는 의미
    graph[a].append((b,c))

#방문하지 않은 노드 중에서, 가장 최단 거리가 짧은 노드의 번호를 반환
def get_smallest_node():
    min_value = INF
    index = 0 # 가장 최단 거리가 짧은 노드(인덱스)
    for i in range(1,n+1):
        if distance[i] < min_value and not vistited[i]:
            min_value = distance[i]
            index = i
        return index

def dijjkstra(start):
    #시작 노드에 대해서 초기화
    distance[start] = 0
    visited[start] = True
    for j in graph[start]:
        distance[j[0]] = j[1]
    # 시작 노드를 제외한 전체 n-1 개의 노드에 대해 반복
    for i in range(n-1):
        #현재 최단 거리가 가장 짧은 노드를 꺼내서, 방문처리
        now = get_smallest_node()
        visited[now] = True
        #현재 노드와 연결된 다른 노드를 확인
        for j in graph[now]:
            cost = distance[now] + j[1]
            # 현재 노드를 거쳐서 다른 노드로 이동하는 거리가 더 짧은 경우
            if cost < distance[j[0]]:
                distance[j[0]] = cost
```

```
# 다익스트라 알고리즘을 수행
dijkstra(start)

# 모든 노드로 가기 위한 최단 거리를 출력
for i in range(1, n+1):
    # 도달할 수없는 경우, 무한 이라고 출력
    if distance[i] == INF:
        print("INFINITY")
    #도달할 수 있는 경우 거리를 출력
    else:
        print(distance[i])
```

1.6 간단한 구현 방법 성능 분석

- **총 O(V) 번에 걸쳐서 최단 거리가 가장 짧은 노드를 매번 선형 탐색해야 한다.**
- **따라서 전체 시간 복잡도는 O(V 제곱)이다**
- **일반ㅇ적으로 코딩 테스트의 최단 경로 문제에서 전체 노드의 개수가 5,000개 이하라면 이 코드로 해결할 수 있지만 10000개가 넘어가 면사 사용할 수 없다.**

2. 우선순위 큐

2-1 우선순위 큐란?

- 우선순위가 가장 높은 데이터를 가장 먼저 삭제하는 자료구조
- 예를 들어 여러 개의 물건 데이터를 자료구조에 넣었다가 가치가 높은 물건 데이터부터 꺼내서 확인해야 하는 경우 우선순위 큐를 이용할 수 있다.

2-2 힙(heap)

- 우선순위 큐를 구현하기 위해 사용하는 자료구조 중 하나
- 최소 힙과 최대합이 있다.
- 다익스트라 최단 경로 알고리즘을 포함해 다양한 알고리즘에 사용

우선순위 큐 구현 방식	삽입 시간	삭제 시간
리스트	$O(1)$	$O(N)$
힙(Heap)	$O(logN)$	$O(logN)$

2.3 최소 힙 구현 예제

```
impoert heapq

# 오름차순 힙정력

def headpsort(iterable):
    h = []
    result = []


    #모든 원소를 차례대로 힙에 삽입
    for value in iterable:
        heapq.heappush(h,value)


    #힙에 삽입된 모든 원소를 차례대로 꺼내어 담기
    for i in range(len(h)):
        result.append(heapq.heappop(h))
    return result

result = heapsort([1,3,5,7,9,2,4,6,8,0])
print(Result)
=>0,1,2,3,4,5,6,7,8,9
```

2.4 최대 힙 구현 에제

```
impoert heapq

# 오름차순 힙정력

def headpsort(iterable):
    h = []
    result = []


    #모든 원소를 차례대로 힙에 삽입
    for value in iterable:
        heapq.heappush(h,-value)
```

```
    # 힙에 삽입된 모든 원소를 차례대로 꺼내어 담기
    for i in range(len(h)):
        result.append(-heapq.heappop(h))
    return result

result = heapsort([1,3,5,7,9,2,4,6,8,0])
print(Result)
=>9,8,7,6,5,4,3,2,1
```

2.5 개선된 구현 방법

- 단계마다 방문하지 않은 노드 중에서 최단거리가 가장 짧은 노드를 선택하기 위해 힙 자루 구죠 이용
- 다익스트라 알고리즘 동작하는 기본 원리
    - 현재 가장 가까운 노드를 저장해 놓기 위해 힙 자료구조 이용
    - 현재의 최단 거리가 가장 짧은 노드를 선택해야 하므로 최소 힙 사용

2.6 개선된 구현 방법

```
import sys
input = sys.stdin.readline

INF = int(1e9) # 무한을 의미하는 값으로 10 억을 설정

# 노드의 개수, 간선의 개수를 입력받기
n,m = map(int, input().split())

#시작 노드번호를 입력받기
start = int(input())

#각 노드에 연결되어 있는 노드에 대한 정보를 담는 리스트를 만들기
graph [[]for in range(n+1)]

# 방문한 적이 있는지 체크하는 목적의 리스트를 만들기
visited = [False] * (n+1)

#최단 거리 테이블을 모두ㅡ 무한으로 초기화
distance = [INF] * (n+1)

# 모든 간선 정보를 입력 받기
for _in range(m):
    a,b,c = map(int, input().split())
    # a 번 노드에서 b 번 노드로 가는 비용이 c 라는 의미
    graph[a].append((b,c))
```

```
=
def dijjkstra(start):
    #시작 노드에 대해서 초기화
    distance[start] = 0
    while q:
        dist , now = heapq.heappop(q)
        if distance[now] < dist:
            continue\
        for i in graph[now]:
            cost = dist + [i]
            if cost < distance[i[0]]:
                distance[i[0]] = cost
                heapq.heappush(q,(cost,i[0]))

# 더ㅏ익스트라 알고리즘을 수행
dijkstra(start)

# 모든 노드로 가기 위한 최단 거리를 출력
for i in range(1, n+1):
    # 도달할 수없는 경우 , 무한 이라고 출력
    if distance[i] == INF:
        print("INFINITY")
    #도달할 수 있는 경우 거리를 출력
    else:
        print(distance[i])
```

2.7 개선된 구현 방법 성능 분석

- 힙 자료구조를 이용하는 다익스트라 알고리즘의 시간 복잡도는 O(ElogV)
- 노드를 하나씩 꺼내 검사하는 반복문은 노드의 개수 V 이상의 횟수로는 처리되지 않는다.
    - 결과적으로 현재 우선순위 큐에서 꺼낸 노드와 연결된 다른 노드들을 확인하는 총 횟수는 최대 간선의 개수만큼 연산이 수행
- 직관적으로 전체 과정은 E개의 원소를 우선순위 큐에 넣었다가 모두 빼내는 연산과 매우 유사
    - 시간 복잡도 O(ElogE)로 판단

이 자료는 동빈 나 님의 **이코 테** 유튜브 영상을 보고 정리한 자료입니다.

참고 : www.youtube.com/watch?v=m-9pAwq1o3w&list=PLRx0vPvlEmdAghTr5mXQxGpHjWqSz0dgC

'**Algorithm**' 카테고리의 다른 글□

[Algorithm] 30강 : 다익스트라 최단 경로 알고리즘□

[Algorithm] 29강 : 다이나믹 프로그래밍의 기초 문제 풀이□

[Algorithm] 28강 : 다이나믹 프로그래밍의 정의와 구현□

[Algorithm] 27강 : 이진 탐색 기초 문제 풀이□

[Algorithm] 26강 : 이진 탐색 알고리즘 정의와 구현□

[Algorithm] 25강 : 정렬 알고리즘 복잡도 비교 및 기본 문제□

다익스트라 최단 경로 알고리즘 최단 경로 알고리즘 최단경리 알고리즘

나아무늘보

혼자 끄적끄적하는 블로그 입니다.