Chapter 10.
최단 경로 알고리즘
핵심 유형 문제풀이

핵심 유형 문제풀이 | 다양한 문제를 접하며 코딩 테스트에 익숙해지기

강사 나동빈

Chapter 10. 최단 경로 알고리즘

핵심 유형 문제풀이

혼자 힘으로 풀어보기

문제 제목: 특정한 최단 경로

문제 난이도: ★★★☆☆

문제 유형: 최단 경로, 다익스트라

추천 풀이 시간: 50분

특정한 최단 경로: https://www.acmicpc.net/problem/1504

문제 풀이 핵심 아이디어

- 노드의 개수가 800개이므로, **다익스트라**를 활용해 최단 경로를 계산할 수 있다.
- 임의로 주어진 두 개의 정점 A와 B를 반드시 통과하는 최단 경로를 계산한다.
- 따라서 아래의 두 경우 중에서 더 짧은 경우를 계산하면 된다.

1. 1 → A → B → N
2. 1 → B → A → N

- 이를 위해 총 3번의 다익스트라 알고리즘을 수행하면 된다.

1. 노드 1에서 출발하여 A, B에 도착할 때
2. 노드 A에서 출발하여 B, N에 도착할 때
3. 노드 B에서 출발하여 A, N에 도착할 때

소스 코드


```python
import sys
input = sys.stdin.readline # 빠른 입력 함수 사용
import heapq # 우선순위 큐 라이브러리
INF = int(1e9) # 무한을 의미하는 값으로 10억을 설정

def dijkstra(start):
    q = []
    # 시작 노드로 가기 위한 최단 경로는 0으로 설정하여, 큐에 삽입
    heapq.heappush(q, (0, start))
    distance[start] = 0
    while q: # 큐가 비어있지 않다면
        # 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 거쳐, 다른 노드로 가는 거리가 더 짧으면
            if cost < distance[i[0]]:
                distance[i[0]] = cost
                heapq.heappush(q, (cost, i[0]))

# 노드의 개수, 간선의 개수를 입력받기
n, m = map(int, input().split())
# 각 노드에 연결되어 있는 노드에 대한 정보를 담는 리스트를 만들기
graph = [[] for i in range(n + 1)]
# 모든 간선 정보를 입력받기
for _ in range(m):
    a, b, c = map(int, input().split())
    # a번 노드에서 b번 노드로 가는 비용이 c라는 의미
    graph[a].append((b, c))
    graph[b].append((a, c))
# 꼭 거쳐야 하는 a와 b 노드 입력받기
a, b = map(int, input().split())

distance = [INF] * (n + 1) # 테이블 초기화
dijkstra(1) # 다익스트라 알고리즘을 수행
d_1_to_a = distance[a]
d_1_to_b = distance[b]
distance = [INF] * (n + 1) # 테이블 초기화
dijkstra(a) # 다익스트라 알고리즘을 수행
d_a_to_b = distance[b]
d_a_to_n = distance[n]
distance = [INF] * (n + 1) # 테이블 초기화
dijkstra(b) # 다익스트라 알고리즘을 수행
d_b_to_a = distance[a]
d_b_to_n = distance[n]

route1 = d_1_to_a + d_a_to_b + d_b_to_n
route2 = d_1_to_b + d_b_to_a + d_a_to_n
result = min(route1, route2)
if result >= INF: print(-1)
else: print(result)
```

거의 최단 경로: https://www.acmicpc.net/problem/5719

혼자 힘으로 풀어보기

문제 제목: 거의 최단 경로

문제 난이도: ★★★★☆

문제 유형: 최단 경로, 다익스트라

추천 풀이 시간: 60분

문제 풀이 핵심 아이디어

- 본 문제는 **다익스트라** 알고리즘을 활용해 해결할 수 있는 문제다.
- **"최단 경로에 포함되지 않는 도로로만 이루어진 경로 중에서 가장 짧은 것"**을 찾자.

[문제 해결 방법]

1. 먼저 다익스트라 알고리즘을 이용해 최단 경로를 한 번 계산한다.
2. 이후에 BFS를 이용해 "최단 경로들"에 포함된 모든 간선을 찾는다.
   참고로 최단 경로는 하나가 아닐 수 있으며, 모든 간선을 다 찾아야 한다.
3. 결과적으로 그러한 간선들을 모두 지운 뒤에, 다시 다익스트라를 사용한다.

[참고]

- 기존의 다익스트라 함수를 전혀 변경하지 않고, 그대로 사용할 수 있다.
  다만, 경로 추적을 위한 $BFS()$ 함수를 구현해야 한다.

소스 코드 1)


```python
import sys
# 빠른 입력 함수 사용
input = sys.stdin.readline
from collections import deque # 큐 라이브러리
import heapq # 우선순위 큐 라이브러리
INF = int(1e9) # 무한을 의미하는 값으로 10억을 설정


def dijkstra():
    q = []
    # 시작 노드로 가기 위한 최단 경로는 0으로 설정하여, 큐에 삽입
    heapq.heappush(q, (0, start))
    distance[start] = 0
    while q: # 큐가 비어있지 않다면
        # 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 거쳐, 다른 노드로 가는 거리가 더 짧으면
            if cost < distance[i[0]]:
                distance[i[0]] = cost
                heapq.heappush(q, (cost, i[0]))
```

```python
# 최단 경로 역추적 함수
def bfs():
    q = deque()
    visited = set() # 특정한 노드 방문 여부
    q.append(end) # 도착 지점(end)을 큐에 삽입
    removes = set() # 삭제할 간선들(결과)
    while q:
        now = q.popleft()
        # 모든 최단 경로를 확인하기 위해 break 대신 continue
        if now == start:
            continue
        for i in reversed_graph[now]: # 현재 노드와 연결된 간선들
            cost = distance[i[0]] + i[1]
            # 최단 경로에 포함된 간선인 경우 삭제 목록에 추가
            if cost == distance[now]:
                removes.add((i[0], now))
                # 각 "직전 노드"는 한 번씩만 방문
                if i[0] not in visited:
                    q.append(i[0])
                    visited.add(i[0])
    return removes
```

소스 코드 2)

들여쓰기

```
while True:
    # 노드의 개수, 간선의 개수를 입력받기
    n, m = map(int, input().split())
    if n == 0 and m == 0: # 테스트 케이스 종료
        break
    # 시작 노드와 도착 노드 입력받기
    start, end = map(int, input().split())
    # 각 노드에 연결되어 있는 노드에 대한 정보를 담는 리스트를 만들기
    graph = [[] for i in range(n)]
    # 경로 추적을 위한 역순 그래프
    reversed_graph = [[] for i in range(n)]

    # 모든 간선 정보를 입력받기
    for _ in range(m):
        a, b, c = map(int, input().split())
        # a번 노드에서 b번 노드로 가는 비용이 c라는 의미
        graph[a].append((b, c))
        reversed_graph[b].append((a, c))

    # 최단 거리 테이블을 모두 무한으로 초기화
    distance = [INF] * n
    # 다익스트라 알고리즘을 수행
    dijkstra()
        # 최단 경로 역추적
        removes = bfs()
        new_graph = [[] for i in range(n)]
        for a in range(n):
            for b, c in graph[a]:
                # 삭제 목록에 포함되지 않은 간선만 넣기
                if (a, b) not in removes:
                    new_graph[a].append((b, c))
        graph = new_graph

        # 최단 거리 테이블을 모두 무한으로 초기화
        distance = [INF] * n
        # 다익스트라 알고리즘을 수행
        dijkstra()

        # 도달할 수 없는 경우, -1을 출력
        if distance[end] == INF:
            print(-1)
        # 도달할 수 있는 경우 거리를 출력
        else:
            print(distance[end])
```

개코전쟁: https://www.acmicpc.net/problem/2325

혼자 힘으로 풀어보기

문제 제목: 개코전쟁

문제 난이도: ★★★★☆

문제 유형: 최단 경로, 다익스트라

추천 풀이 시간: 60분

개코전쟁: https://www.acmicpc.net/problem/2325

문제 풀이 핵심 아이디어

- 본 문제에서는 양방향 그래프가 주어진다.
- **하나의 간선을 제거하여 1번 정점에서 N번 정점으로 가는 최단 거리를 최대화**해야 한다.

  하나의 간선을 제거할 때는 양방향의 길(간선)이 모두 제거되는 것으로 이해할 수 있다.
- 노드의 개수가 1,000개 이상이므로, 본 문제는 다익스트라를 이용해 해결해야 한다.

개코전쟁: https://www.acmicpc.net/problem/2325

문제 풀이 핵심 아이디어

[문제 해결 아이디어]

- 모든 간선을 하나씩 지우면서 매번 다익스트라를 수행해 보면, 너무 많은 시간이 소요된다.
- **"최단 경로에 포함되지 않은 간선을 제거하면, 최단 거리는 변하지 않는다"**는 사실에 주목하자.
- 따라서 모든 최단 경로를 구한 뒤에, 최단 경로들에 포함된 간선들을 하나씩 제거해 보자.
  하나씩 지워보면서, 다시 다익스트라를 호출하여 문제를 해결할 수 있다.

개코전쟁: https://www.acmicpc.net/problem/2325

문제 풀이 핵심 아이디어

[문제 해결 방법]

1. 먼저 다익스트라 알고리즘을 이용해 최단 경로를 한 번 계산한다.
2. 이후에 BFS를 이용해 "최단 경로들"에 포함된 모든 간선을 찾는다.

   참고로 최단 경로는 하나가 아닐 수 있으며, 모든 간선을 다 찾아야 한다.
3. 결과적으로 그러한 간선들을 하나씩 지우면서, 매번 다익스트라를 호출한다.

[참고]

- 기존의 다익스트라 함수를 전혀 변경하지 않고, 그대로 사용할 수 있다.

  다만, 경로 추적을 위한 $BFS()$ 함수를 구현해야 한다.

개코전쟁: https://www.acmicpc.net/problem/2325

소스 코드 1)


```python
import sys
# 빠른 입력 함수 사용
input = sys.stdin.readline
from collections import deque # 큐 라이브러리
import heapq # 우선순위 큐 라이브러리
INF = int(1e9) # 무한을 의미하는 값으로 10억을 설정

# 일반적인 다익스트라와 동일하지만, a ↔ b 간선은 무시하는 함수
def dijkstra(a, b):
    q = []
    # 시작 노드로 가기 위한 최단 경로는 0으로 설정하여, 큐에 삽입
    heapq.heappush(q, (0, start))
    distance[start] = 0
    while q: # 큐가 비어있지 않다면
        # 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]:
            # a ↔ b 간선은 무시
            if i[0] == a and now == b: continue
            elif i[0] == b and now == a: continue
            cost = dist + i[1]
            # 현재 노드를 거쳐, 다른 노드로 가는 거리가 더 짧으면
            if cost < distance[i[0]]:
                distance[i[0]] = cost
                heapq.heappush(q, (cost, i[0]))
```

```python
# 최단 경로 역추적 함수
def bfs():
    q = deque()
    visited = set() # 특정한 노드 방문 여부
    q.append(end) # 도착 지점(end)을 큐에 삽입
    removes = set() # 삭제할 간선들(결과)
    while q:
        now = q.popleft()
        # 모든 최단 경로를 확인하기 위해 break 대신 continue
        if now == start:
            continue
        for i in graph[now]: # 현재 노드와 연결된 간선들 확인
            cost = distance[i[0]] + i[1]
            # 최단 경로에 포함된 간선인 경우 삭제 목록에 추가
            if cost == distance[now]:
                removes.add((i[0], now))
                # 각 "직전 노드"는 한 번씩만 방문
                if i[0] not in visited:
                    q.append(i[0])
                    visited.add(i[0])
    return removes
```

개코전쟁: https://www.acmicpc.net/problem/2325

소스 코드 2)


```python
# 노드의 개수, 간선의 개수를 입력받기
n, m = map(int, input().split())
# 시작 노드와 도착 노드
start, end = 1, n
# 각 노드에 연결되어 있는 노드에 대한 정보를 담는 리스트
graph = [[] for i in range(n + 1)]

# 모든 간선 정보를 입력받기
for _ in range(m):
    a, b, c = map(int, input().split())
    # a번 노드에서 b번 노드로 가는 비용이 c라는 의미
    graph[a].append((b, c))
    graph[b].append((a, c))
```

```python
# 최단 거리 테이블을 모두 무한으로 초기화
distance = [INF] * (n + 1)
# 다익스트라 알고리즘을 수행
dijkstra(-1, -1)

# 최단 경로 역추적 수행
# 모든 최단 경로에 포함된 간선 쌍 (a, b)들을 계산
removes = bfs()

result = 0
# 모든 최단 경로에 포함된 간선 쌍 (a, b)들을 확인
for a, b in removes:
    # 최단 거리 테이블을 모두 무한으로 초기화
    distance = [INF] * (n + 1)
    # a ↔ b 간선은 무시하는 다익스트라 알고리즘을 수행
    dijkstra(a, b)
    result = max(result, distance[end])
print(result)
```