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

플로이드: https://www.acmicpc.net/problem/11404

혼자 힘으로 풀어보기

문제 제목: 플로이드

문제 난이도: ★★★☆☆

문제 유형: 최단 경로, 플로이드 워셜

추천 풀이 시간: 40분

플로이드: https://www.acmicpc.net/problem/11404

문제 풀이 핵심 아이디어

[문제 설명]

- 도시의 개수(N)가 최대 100개다.
- 간선의 개수(M)가 최대 100,000개다.
- 모든 도시의 쌍(A, B)에 대해 도시 A에서 B로 가는데 필요한 **최소 비용**을 모두 구해라.

플로이드: https://www.acmicpc.net/problem/11404

문제 풀이 핵심 아이디어

- 모든 도시의 쌍(A, B)에 **대하여 최단 거리를 모두 구해야 한다.**
- **플로이드 워셜(Floyd-Warshall)** 알고리즘을 사용할 수 있다.

  노드의 개수(N)가 최대 100개이므로, 시간 복잡도 $O(N^3)$으로 충분히 해결 가능하다.
- 두 노드 사이의 간선이 여러 개일 수 있으므로, 가장 비용이 적은 간선만 고려한다.

플로이드: https://www.acmicpc.net/problem/11404

소스 코드

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

# 노드의 개수 및 간선의 개수를 입력받기
n = int(input())
m = int(input())
# 2차원 리스트(그래프)를 만들고, 무한으로 초기화
graph = [[INF] * (n + 1) for _ in range(n + 1)]

# 자기 자신으로 가는 비용은 0으로 초기화
for a in range(1, n + 1):
    for b in range(1, n + 1):
        if a == b:
            graph[a][b] = 0

# 각 간선에 대한 정보를 입력 받아, 그 값으로 초기화
for _ in range(m):
    # A에서 B로 가는 비용은 C라고 설정
    a, b, c = map(int, input().split())
    # 가장 비용이 적은 간선만 고려
    graph[a][b] = min(graph[a][b], c)
```

```python
# 점화식에 따라 플로이드 워셜 알고리즘을 수행
for k in range(1, n + 1):
    for a in range(1, n + 1):
        for b in range(1, n + 1):
            graph[a][b] = min(graph[a][b],
                graph[a][k] + graph[k][b])

# 수행된 결과를 출력
for a in range(1, n + 1):
    for b in range(1, n + 1):
        # 도달할 수 없는 경우, 0을 출력
        if graph[a][b] == 1e9:
            print(0, end=" ")
        # 도달할 수 있는 경우 거리를 출력
        else:
            print(graph[a][b], end=" ")
    print()
```

최단 경로: https://www.acmicpc.net/problem/1753

혼자 힘으로 풀어보기

문제 제목: 최단 경로

문제 난이도: ★★★☆☆

문제 유형: 최단 경로, 다익스트라

추천 풀이 시간: 50분

최단 경로: https://www.acmicpc.net/problem/1753

문제 풀이 핵심 아이디어

- 전형적인 방향 그래프에서의 최단 경로 탐색 문제다.
- 하나의 시작점에서 다른 모든 정점까지의 최단 경로를 구해야 한다.

  이때 **노드의 개수(N)**가 최대 20,000개이므로, 다익스트라 최단 경로 알고리즘을 사용한다.
- 우선순위 큐를 활용한 다익스트라 알고리즘으로, $O(ElogV)$의 복잡도를 보장한다.

소스 코드


```python
import sys
# 빠른 입력 함수 사용
input = sys.stdin.readline
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
# 노드의 개수, 간선의 개수를 입력받기
n, m = map(int, input().split())
# 시작 노드 번호 입력받기
start = int(input())
# 각 노드에 연결되어 있는 노드에 대한 정보를 담는 리스트
graph = [[] for i in range(n + 1)]
# 최단 거리 테이블을 모두 무한으로 초기화
distance = [INF] * (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))

# 다익스트라 알고리즘을 수행
dijkstra()
# 각 노드로 가기 위한 최단 거리를 출력
for i in range (1, n + 1):
    # 도달할 수 없는 경우, "INF"를 출력
    if distance[i] == 1e9:
        print("INF")
    # 도달할 수 있는 경우 최단 거리를 출력
    else:
        print(distance[i])
```

도로포장: https://www.acmicpc.net/problem/1162

혼자 힘으로 풀어보기

문제 제목: 도로포장

문제 난이도: ★★★★☆

문제 유형: 최단 경로, 다익스트라, 다이나믹 프로그래밍

추천 풀이 시간: 60분

도로포장: https://www.acmicpc.net/problem/1162

문제 풀이 핵심 아이디어

[문제 설명]

- 1번 노드에서 N번 노드까지 도달하기 위한 최단 거리를 구하는 문제다.
- 노드의 개수(N)가 최대 10,000개 이므로, 다익스트라 최단 경로 알고리즘을 사용해야 한다.
- 이때 **K개의 간선의 비용을 0으로 만들 수 있다는 점**이 본 문제의 특징이다.

도로포장: https://www.acmicpc.net/problem/1162

문제 풀이 핵심 아이디어

[문제 해결 아이디어]

- 본 문제는 **다이나믹 프로그래밍**(dynamic programming)을 사용하여 해결할 수 있다.
- **다익스트라**를 사용하며, 방문하는 각 노드까지의 최단 거리를 DP 테이블에 기록한다.
- 구체적으로 각 노드에 대해 distance[노드 번호][현재까지 포장 횟수]를 갱신한다.

  해당 노드를 **포장할지 안 할지 결정**하는 방식이다.

도로포장: https://www.acmicpc.net/problem/1162

소스 코드


```python
import sys
input = sys.stdin.readline # 빠른 입력 함수 사용
import heapq # 우선순위 큐 라이브러리
INF = int(1e17) # 무한을 의미하는 값으로 큰 값을 설정

def dijkstra(start):
    q = []
    # 시작 노드로 가기 위한 최단 경로는 0으로 설정하여, 큐에 삽입
    heapq.heappush(q, (0, start, 0)) # (비용, 노드 번호, 포장 횟수)
    distance[start][0] = 0
    while q: # 큐가 비어있지 않다면
        # 가장 최단 거리가 짧은 노드에 대한 정보 꺼내기
        dist, now, paved = heapq.heappop(q)
        # 현재 노드가 이미 처리된 적이 있는 노드라면 무시
        if distance[now][paved] < dist: continue
        # 현재 노드와 연결된 다른 인접한 노드들을 확인
        for i in graph[now]:
            # 현재 노드를 거쳐, 다른 노드로 가는 거리가 더 짧으면
            # 1) 포장하지 않는 경우
            cost = dist + i[1]
            if cost < distance[i[0]][paved]:
                distance[i[0]][paved] = cost
                heapq.heappush(q, (cost, i[0], paved))
            # 2) 포장하는 경우(cost 대신에 dist 사용)
            if paved < k and dist < distance[i[0]][paved + 1]:
                distance[i[0]][paved + 1] = dist
                heapq.heappush(q, (dist, i[0], paved + 1))
```

```python
# 노드의 개수, 간선의 개수, 포장 횟수
n, m, k = map(int, input().split())
# 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())
    # 양방향 간선
    graph[a].append((b, c))
    graph[b].append((a, c))

# 최단 거리 테이블 초기화
distance = [[INF] * (k + 1) for _ in range(n + 1)]
dijkstra(1) # 다익스트라 알고리즘을 수행

# 노드 N에 도착하기 위한 최소 거리 출력
result = INF
for i in range(k + 1):
    result = min(result, distance[n][i])
print(result)
```