# 알고리즘 스터디 2주차

- 그래프 심화 -

발표자 장수현

# BFS/DFS + DP

https://www.acmicpc.net/problem/14226

## 이모티콘 성공

☆

골드 V

| 시간 제한 | 메모리 제한 | 제출 | 정답 | 맞힌 사람 | 정답 비율 |
|---|---|---|---|---|---|
| 2 초 | 512 MB | 15576 | 5891 | 3856 | 34.389% |

### 문제

영선이는 매우 기쁘기 때문에, 효빈이에게 스마일 이모티콘을 S개 보내려고 한다.

영선이는 이미 화면에 이모티콘 1개를 입력했다. 이제, 다음과 같은 3가지 연산만 사용해서 이모티콘을 S개 만들어 보려고 한다.

1. 화면에 있는 이모티콘을 모두 복사해서 클립보드에 저장한다.
2. 클립보드에 있는 모든 이모티콘을 화면에 붙여넣기 한다.
3. 화면에 있는 이모티콘 중 하나를 삭제한다.

모든 연산은 1초가 걸린다. 또, 클립보드에 이모티콘을 복사하면 이전에 클립보드에 있던 내용은 덮어쓰기가 된다. 클립보드가 비어있는 상태에는 붙여넣기를 할 수 없으며, 일부만 클립보드에 복사할 수는 없다. 또한, 클립보드에 있는 이모티콘 중 일부를 삭제할 수 없다. 화면에 이모티콘을 붙여넣기 하면, 클립보드에 있는 이모티콘의 개수가 화면에 추가된다.

영선이가 S개의 이모티콘을 화면에 만드는데 걸리는 시간의 최솟값을 구하는 프로그램을 작성하시오.

# BFS/DFS + DP

- emogi에 들어가는 값은 걸리는 시간
  ex) emogi[2][4] = 3
  -> 화면에 있는 이모티콘 2개와 클립보드에 저장된 이모티콘 4개를 만드는데 3초가 걸린다
- 시작점은 화면에 있는 이모티콘 1개만으로 시작하므로 [1, 0]을 큐에 넣은 후, 시작점은 0초로 수정

1. 화면에 있는 이모티콘을 모두 클립보드에 저장
   = 복사: (s, c) -> (s, s)
2. 클립보드에 있는 이모티콘을 모두 화면에 붙여넣기
   = 붙여넣기: (s,c) -> (s+c, c)
3. 화면에 있는 이모티콘 중 하나 삭제
   = 삭제: (s, c) -> (s-1, c)

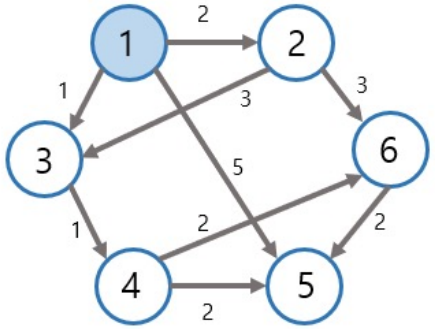
- 모든 연산은 1초 걸리므로 emogi[s][c] + 1 해주는 것

```python
from collections import deque

n = int(input())
emogi = [[-1] * (n+1) for _ in range(n+1)]

def bfs():
    queue = deque([[1, 0]])
    emogi[1][0] = 0

    while queue:
        # 화면에 이모티콘 개수, 클립보드 이모티콘 개수
        s, c = queue.popleft()
        # 화면에 있는 이모티콘을 모두 클립보드에 저장
        if emogi[s][s] == -1:
            queue.append([s, s])
            emogi[s][s] = emogi[s][c] + 1
        # 클립보드에 있는 이모티콘을 모두 화면에 붙여넣기
        if s + c <= n and emogi[s+c][c] == -1:
            queue.append([s+c, c])
            emogi[s+c][c] = emogi[s][c] + 1
        # 화면에 있는 이모티콘 중 하나 삭제
        if s - 1 >= 0 and emogi[s-1][c] == -1:
            queue.append([s-1, c])
            emogi[s-1][c] = emogi[s][c] + 1

bfs()
answer = min([x for x in emogi[n] if x != -1])
print(answer)
```

# 다익스트라 알고리즘

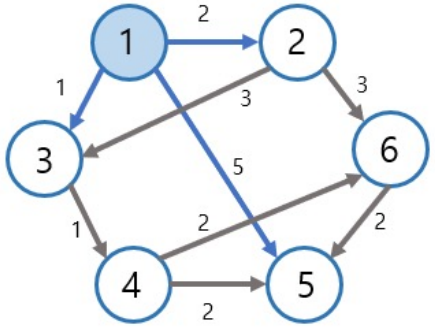
- 한 지점에서 다른 특정 지점까지 최단 경로를 구해야 하는 경우에 사용하는 알고리즘
- 우선순위 큐를 이용하는 것이 좋다 -> 시간복잡도(ElogV)
- 출발 노드가 1개이므로 다른 노드까지 최단 거리를 저장하므로 1차원 리스트를 사용한다

# 다익스트라 알고리즘



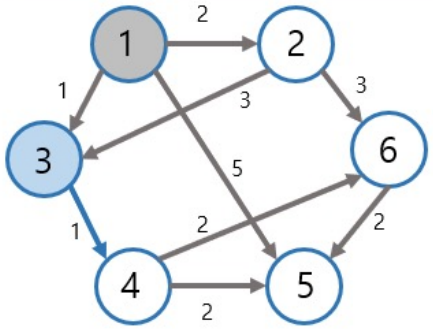

출발 노드 : 1

| 노드 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 거리 | 0 | INF | INF | INF | INF | INF |



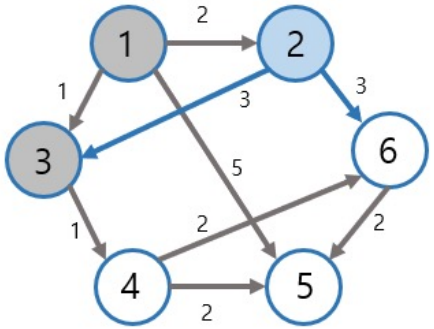

방문하지 않은 노드 중
가장 거리가 짧은 노드 : 1

| 노드 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 거리 | 0 | 2 | 1 | INF | 5 | INF |




방문하지 않은 노드 중
가장 거리가 짧은 노드 : 3

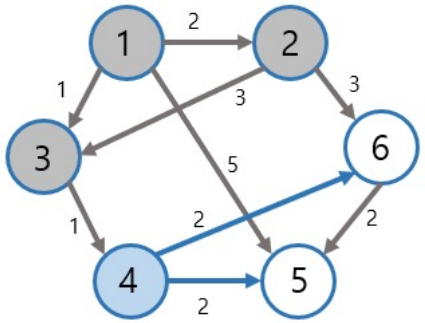
| 노드 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 거리 | 0 | 2 | 1 | 2 | 5 | INF |




방문하지 않은 노드 중
가장 거리가 짧은 노드 : 2

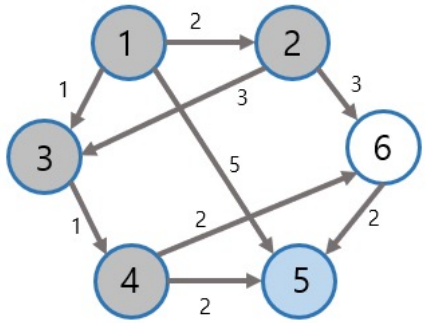
| 노드 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 거리 | 0 | 2 | 1 | 2 | 5 | 5 |

# 다익스트라 알고리즘



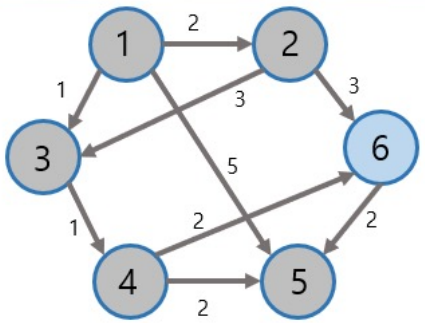

방문하지 않은 노드 중
가장 거리가 짧은 노드 : 4

| 노드 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 거리 | 0 | 2 | 1 | 2 | 4 | 4 |




방문하지 않은 노드 중
가장 거리가 짧은 노드 : 5

| 노드 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 거리 | 0 | 2 | 1 | 2 | 4 | 4 |




방문하지 않은 노드 중
가장 거리가 짧은 노드 : 6

| 노드 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 거리 | 0 | 2 | 1 | 2 | 4 | 4 |

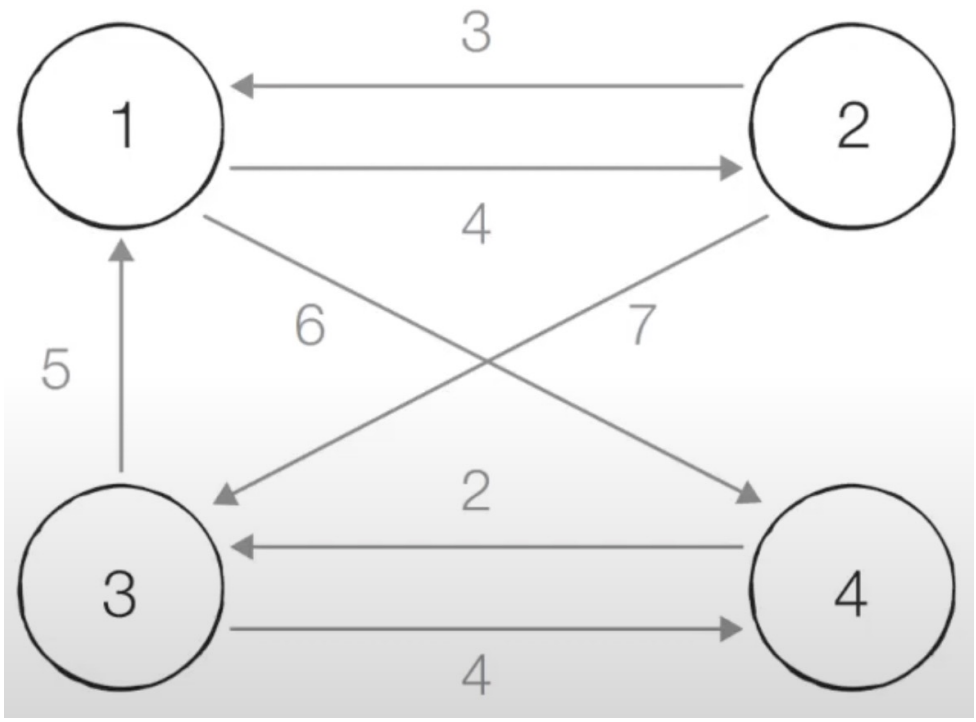
# 다익스트라 알고리즘

```python
def dijkstra(start):
    q=[]
    # 시작 노드는 큐에 삽입하고 최단경로는 0으로 설정
    heapq.heappush(q, (0,start))
    distance[start] = 0

    while q:
        # 가장 최단거리가 짧은 노드에 대한 정보 꺼내기
        dist, now = heapq.heappop(q)
        # 현재 노드가 이미 처리된 적이 있다면 무시
        if distance[now] < dist:
            continue
        # 현재 노드와 연결된 인접한 노드들을 확인
        for i in graph[now]:
            cost = dist + i[1]
            # 현재 노드를 거쳐서, 다른 노드로 이동하는 거리가 더 빠른 경우
            if cost < distance[i[0]]:
                distance[i[0]] = cost
                heapq.heappush(q, (cost,i[0]))
```

# 플로이드 워셜 알고리즘

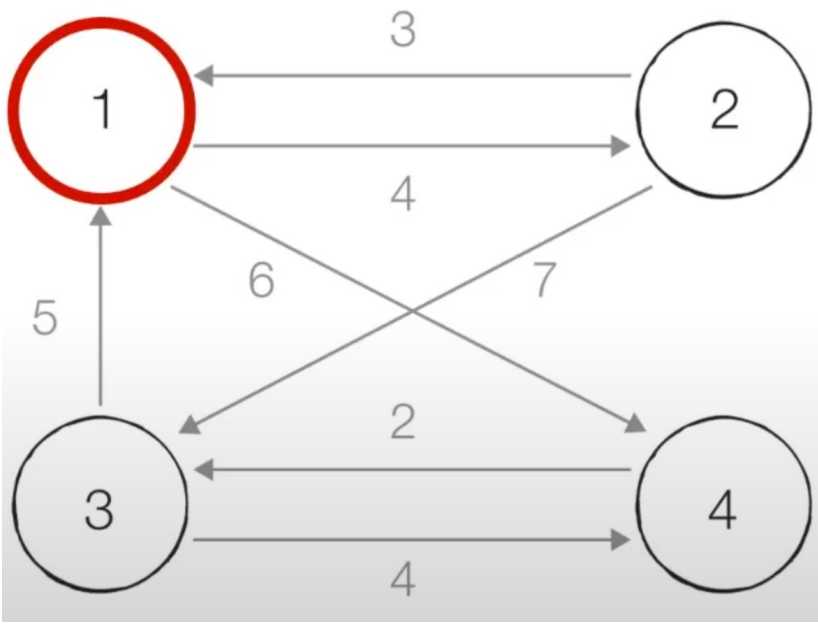
- 모든 지점에서 모든 지점까지 최단 경로를 모두 구해야 하는 경우에 사용하는 알고리즘
- 시간복잡도는 O(N^3), 2차원 리스트에 '최단 거리' 정보를 저장한다
- N(= 노드의 개수)번 만큼의 단계를 반복하며 점화식에 맞게 2차원 리스트를 갱신하므로 다이나믹 프로그래밍이다
- ex) 현재 노드가 1번 노드일 때, A -> 1번 노드 -> B로 가는 모든 경로 확인
- $D_{ab} = \min(D_{ab}, D_{ak} + D_{kb})$
- https://www.youtube.com/watch?v=hw-SvAR3Zqg&list=PLVsNizTWUw7H9_of5YCB0FmsSc-K44y81&index=32

# 플로이드 워셜 알고리즘




| 출발 \ 도착 | 1번 | 2번 | 3번 | 4번 |
|---|---|---|---|---|
| 1번 | 0 | 4 | 무한 | 6 |
| 2번 | 3 | 0 | 7 | 무한 |
| 3번 | 5 | 무한 | 0 | 4 |
| 4번 | 무한 | 무한 | 2 | 0 |

# 플로이드 워셜 알고리즘




$$D_{23} = \min(D_{23}, D_{21} + D_{13})$$
$$D_{24} = \min(D_{24}, D_{21} + D_{14})$$
$$D_{32} = \min(D_{32}, D_{31} + D_{12})$$
$$D_{34} = \min(D_{34}, D_{31} + D_{14})$$
$$D_{42} = \min(D_{42}, D_{41} + D_{12})$$
$$D_{43} = \min(D_{43}, D_{41} + D_{13})$$

**갱신된 최단 거리 테이블**

| | | | |
|---|---|---|---|
| 0 | 4 | 무한 | 6 |
| 3 | 0 | 7 | 9 |
| 5 | 9 | 0 | 4 |
| 무한 | 무한 | 2 | 0 |

# 플로이드 워셜 알고리즘

갱신된 최단 거리 테이블

| | | | |
|---|---|---|---|
| 0 | 4 | 8 | 6 |
| 3 | 0 | 7 | 9 |
| 5 | 9 | 0 | 4 |
| 7 | 11 | 2 | 0 |

$$D_{ab} = \min(D_{ab}, D_{ak} + D_{kb})$$

```
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])
```

# 풀어볼 문제

- 백준 14226 이모티콘 (BFS)
  https://www.acmicpc.net/problem/14226
- 프로그래머스 12978 배달 (다익스트라)
  https://programmers.co.kr/learn/courses/30/lessons/12978
- 백준 11404 플로이드 (플로이드 워셜)
  https://www.acmicpc.net/problem/11404

# 공통문제

- 백준 1987 알파벳 (BFS/DFS)
  https://www.acmicpc.net/problem/1987
- 백준 1504 특정한 최단 경로 (다익스트라)
  https://www.acmicpc.net/problem/1504
- 프로그래머스 49191 순위 (플로이드 워셜)
  https://programmers.co.kr/learn/courses/30/lessons/49191