Chapter. 06

기본 탐색 알고리즘

핵심 유형 문제풀이

FAST CAMPUS
ONLINE
유형별 문제풀이

강사. 나동빈

Chapter. 06

기본 탐색 알고리즘(핵심 유형 문제풀이)

FAST CAMPUS
ONLINE
나동빈 강사.

I 혼자 힘으로 풀어 보기

공유기 설치: https://www.acmicpc.net/problem/2110

문제 제목: 공유기 설치

문제 난이도: 중(Medium)

문제 유형: 이진 탐색

추천 풀이 시간: 40분

공유기 설치: https://www.acmicpc.net/problem/2110

I 문제 풀이 핵심 아이디어

- 집의 개수 N은 최대 200,000이며, 집의 좌표 X는 최대 1,000,000,000입니다.
- 이진 탐색을 이용하여 $\boldsymbol{O(N * logX)}$에 문제를 해결할 수 있습니다.
- 가장 인접한 두 공유기 사이의 최대 Gap을 이진 탐색으로 찾습니다.

공유기 설치: https://www.acmicpc.net/problem/2110

문제 풀이 핵심 아이디어

- 반복적으로 Gap을 설정하며, C개 이상의 공유기를 설치할 수 있는 경우를 찾습니다. (N = 5, C = 3)
  - 최대 Gap = 8 결과 = 0
  - 최소 Gap = 1

공유기 설치: https://www.acmicpc.net/problem/2110

문제 풀이 핵심 아이디어

- 반복적으로 Gap을 설정하며, C개 이상의 공유기를 설치할 수 있는 경우를 찾습니다. (N = 5, C = 3)
  - 최대 Gap = 8 결과 = 0
  - 최소 Gap = 1 Gap = 4

공유기: 2개

⇨ 설치 가능한 공유기의 개수가 C보다 작으므로, Gap을 감소시킵니다.

공유기 설치: https://www.acmicpc.net/problem/2110

문제 풀이 핵심 아이디어

- 반복적으로 Gap을 설정하며, C개 이상의 공유기를 설치할 수 있는 경우를 찾습니다. (N = 5, C = 3)
  - 최대 Gap = 3 결과 = 0
  - 최소 Gap = 1

공유기 설치: https://www.acmicpc.net/problem/2110

I 문제 풀이 핵심 아이디어

- 반복적으로 Gap을 설정하며, C개 이상의 공유기를 설치할 수 있는 경우를 찾습니다. (N = 5, C = 3)
  - 최대 Gap = 3 결과 = 2
  - 최소 Gap = 1 Gap = 2

공유기: 3개

⇨ 설치 가능한 공유기의 개수가 C 이상이므로, **현재의 Gap을 저장**한 뒤에 Gap을 증가시킵니다.

공유기 설치: https://www.acmicpc.net/problem/2110

I 문제 풀이 핵심 아이디어

- 반복적으로 Gap을 설정하며, C개 이상의 공유기를 설치할 수 있는 경우를 찾습니다. (N = 5, C = 3)
  - 최대 Gap = 3 결과 = 2
  - 최소 Gap = 3

공유기 설치: https://www.acmicpc.net/problem/2110

문제 풀이 핵심 아이디어

- 반복적으로 Gap을 설정하며, C개 이상의 공유기를 설치할 수 있는 경우를 찾습니다. (N = 5, C = 3)
  - 최대 Gap = 3 결과 = 3
  - 최소 Gap = 3 Gap = 3

공유기: 3개

⇨ 더 이상 Gap을 증가시킬 수 없으므로, 최적의 경우입니다.

공유기 설치: https://www.acmicpc.net/problem/2110

| 소스코드

```python
n, c = list(map(int, input().split(' ')))

array = []
for _ in range(n):
    array.append(int(input()))
array = sorted(array)

start = array[1] - array[0]
end = array[-1] - array[0]
result = 0

while(start <= end):
    mid = (start + end) // 2 # mid는 Gap을 의미합니다.
    value = array[0]
    count = 1
    for i in range(1, len(array)):
        if array[i] >= value + mid:
            value = array[i]
            count += 1
    if count >= c: # C개 이상의 공유기를 설치할 수 있는 경우
        start = mid + 1
        result = mid
    else: # C개 이상의 공유기를 설치할 수 없는 경우
        end = mid - 1

print(result)
```

I 혼자 힘으로 풀어 보기

중량제한: https://www.acmicpc.net/problem/1939

문제 제목: 중량제한

문제 난이도: 중상(Hard)

문제 유형: 이진 탐색

추천 풀이 시간: 1시간

중량제한: https://www.acmicpc.net/problem/1939

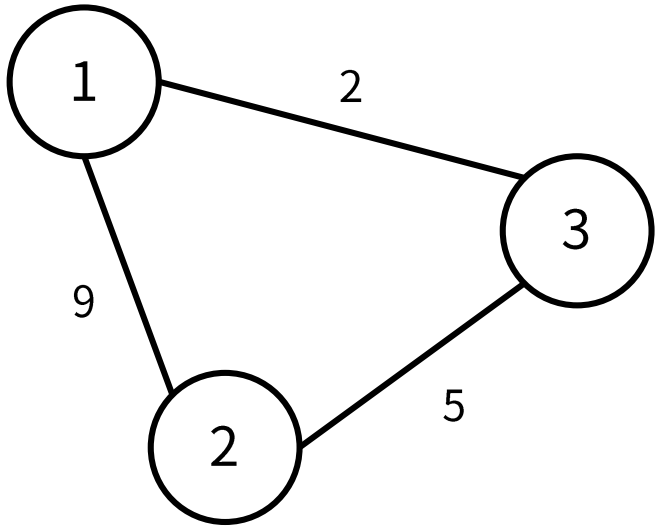
문제 풀이 핵심 아이디어

- 다리의 개수 M은 최대 100,000이며, 중량 제한 C는 최대 1,000,000,000입니다.
- 이진 탐색을 이용하여 $O(M * logC)$에 문제를 해결할 수 있습니다.
- 한 번의 이동에서 옮길 수 있는 물품들의 중량의 최댓값을 이진 탐색으로 찾습니다.

중량제한: https://www.acmicpc.net/problem/1939

문제 풀이 핵심 아이디어

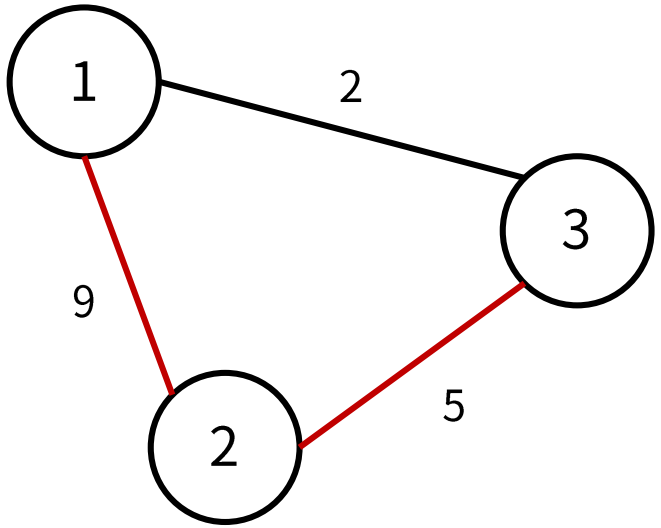
- 반복적으로 중량을 설정하며, 이동이 가능한 경우를 찾습니다. (시작 노드: 1, 도착 노드: 3)
  - 최대 중량 = 9 결과 = 2
  - 최소 중량 = 2



중량제한: https://www.acmicpc.net/problem/1939

문제 풀이 핵심 아이디어

- 반복적으로 중량을 설정하며, 이동이 가능한 경우를 찾습니다. (시작 노드: 1, 도착 노드: 3)
  - 최대 중량 = 9 결과 = 5
  - 최소 중량 = 2 중량 = 5

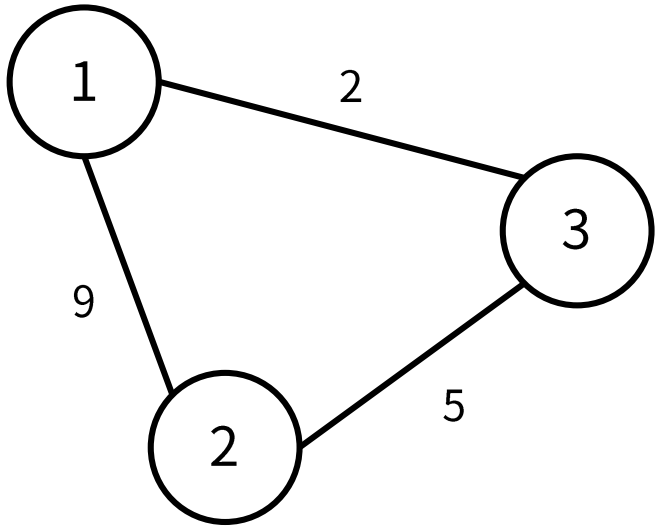


⇨ 이동이 가능하므로, 중량을 증가시킵니다.

중량제한: https://www.acmicpc.net/problem/1939

문제 풀이 핵심 아이디어

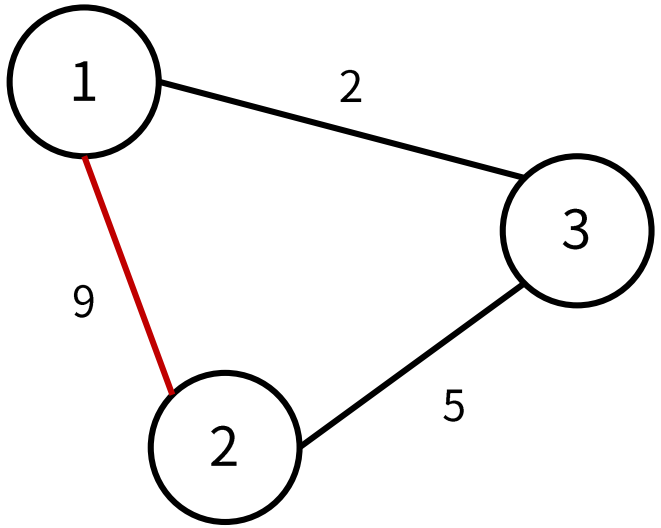
- 반복적으로 중량을 설정하며, 이동이 가능한 경우를 찾습니다. (시작 노드: 1, 도착 노드: 3)
  - 최대 중량 = 9 결과 = 5
  - 최소 중량 = 6



중량제한: https://www.acmicpc.net/problem/1939

문제 풀이 핵심 아이디어

- 반복적으로 중량을 설정하며, 이동이 가능한 경우를 찾습니다. (시작 노드: 1, 도착 노드: 3)
  - 최대 중량 = 9 결과 = 5
  - 최소 중량 = 6 중량 = 7

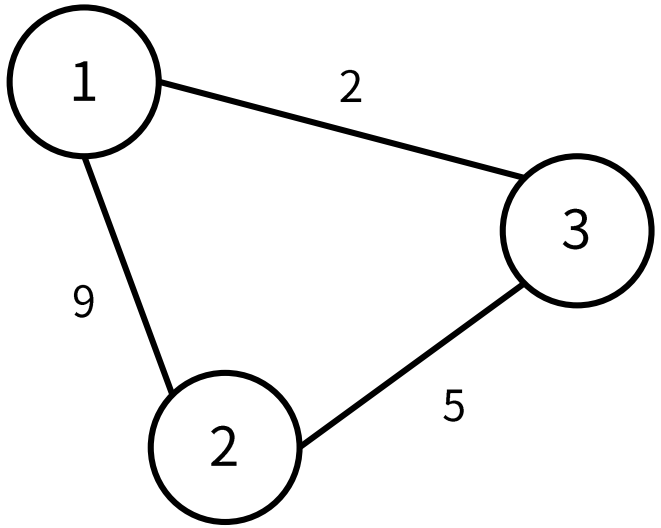


⇨ 이동이 불가능하므로, 중량을 감소시킵니다.

중량제한: https://www.acmicpc.net/problem/1939

문제 풀이 핵심 아이디어

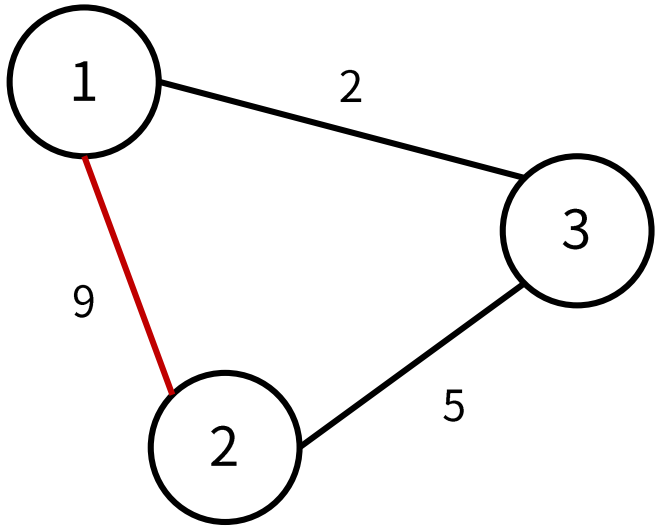
- 반복적으로 중량을 설정하며, 이동이 가능한 경우를 찾습니다. (시작 노드: 1, 도착 노드: 3)
  - 최대 중량 = 6 결과 = 5
  - 최소 중량 = 6



중량제한: https://www.acmicpc.net/problem/1939

I 문제 풀이 핵심 아이디어

- 반복적으로 중량을 설정하며, 이동이 가능한 경우를 찾습니다. (시작 노드: 1, 도착 노드: 3)
  - 최대 중량 = 6 결과 = 5
  - 최소 중량 = 6 중량 = 6



⇨ 이동이 불가능하며, 더 이상 중량을 감소시킬 수 없습니다.

중량제한: https://www.acmicpc.net/problem/1939

소스코드

```python
from collections import deque

n, m = map(int, input().split())
adj = [[] for _ in range(n + 1)]

def bfs(c):
    queue = deque([start_node])
    visited = [False] * (n + 1)
    visited[start_node] = True
    while queue:
        x = queue.popleft()
        for y, weight in adj[x]:
            if not visited[y] and weight >= c:
                visited[y] = True
                queue.append(y)
    return visited[end_node]
```

```python
start = 1000000000
end = 1

for _ in range(m):
    x, y, weight = map(int, input().split())
    adj[x].append((y, weight))
    adj[y].append((x, weight))
    start = min(start, weight)
    end = max(end, weight)

start_node, end_node = map(int, input().split())

result = start
while(start <= end):
    mid = (start + end) // 2 # mid는 현재의 중량을 의미합니다.
    if bfs(mid): # 이동이 가능하므로, 중량을 증가시킵니다.
        result = mid
        start = mid + 1
    else: # 이동이 불가능하므로, 중량을 감소시킵니다.
        end = mid - 1

print(result)
```