Computer Algorithm (CSE 301) HW4

202211167 장지원

1. 달구의 카드게임

   알고리즘 구상 아이디어: Greedy algorithm

   details: 카드를 뽑을 때 '현재 가능한 가장 큰 숫자 +1'을 선택하는 것이 optimal을 배제하지 않는다.

```
for card in cards:
        if (possible_num >= goal_num):
            break
        if (card <= possible_num+1):
            possible_num += card
            continue
        if (card >= possible_num+2):
            while (card >= possible_num+2):
                additional_pick += 1
                possible_num += possible_num+1 # 계속 빈칸 채우면서 뽑기
(card <= possible_num+1) 이 조건 만족할 때 까지
                if (possible_num >= goal_num):
                    break # 엣지 case: 1 1 / 3
            possible_num += card
    card = possible_num+1 # 카드를 효율적으로 뽑기
    while (possible_num < goal_num):
        if (possible_num >= goal_num):
            break
        additional_pick += 1
        possible_num += card
        card = possible_num+1 # 카드를 효율적으로 뽑기
```

2. 달구의 음식점

   알고리즘 구상 아이디어: Kruskal algorithm

   details: input(table info)를 distance 기준으로 sorting한 후, distance가 작은 것부터 MST에 추가. 이 때 cycle이 생기지 않게 주의한다. 미리 sorting을 진행하고, Kruskal을 돌리면, union-find DS(data structure)를 사용하지 않아도 시간 초과가 생기지 않게 된다.

```
table_info.sort(key=lambda x: x[2])  # 거리순 정렬

    MST = set([0])  # 주방부터 시작해서 연결된 노드들
    result = 0  # 총 거리

    # 모든 테이블이 연결될 때까지 반복
    while len(MST) < table:
        for a, b, distance in table_info:
```

```
            if (a in MST) != (b in MST): # 사이클 조건
                MST.add(a)
                MST.add(b)
                result += distance
                break
```

3. 부족 전쟁

알고리즘 구상 아이디어: min-cut, max-flow

details: '사용 불가능한 폭탄 기준' 이상인 edge의 capacity를 inf로 설정하고, min-cut, max-flow를 구한다. 이 때 min-cut과 max-flow는 Edmonds karps 알고리즘을 사용해서 구한다.

```
node, edge, bomb_threshold = map(int, input().split()) # 거점, 다리, 최소 폭탄 머시기
table_info = [tuple(map(int, input().split())) for _ in range(edge)]

ajacency_matrix = [[0] * node for _ in range(node)] # ajacency matrix 만들기
for u, v, w in table_info:
    ajacency_matrix[u][v] = w
    ajacency_matrix[v][u] = w

for i in range(node):
    for j in range(node):
        if ajacency_matrix[i][j] >= bomb_threshold:
            ajacency_matrix[i][j] = float('inf') # 핵심!!!!: 유량 inf 로 두고 하기

print(EdmondsKarp(node, ajacency_matrix, 0, node-1)) # 노드 개수, 그래프, s, t
```

4. 달구의 장애물

알고리즘 구상 아이디어: min-cut, max-flow

details: 모든 격자점을 연결하되, in-out edge를 각 node마다 '하나씩' 주고, 이 edge에만 1-capacity를 준다. 나머지 edge의 capacity는 inf로 설정하고, min-cut, max-flow를 구한다. 이 때 min-cut과 max-flow는 Edmonds karps 알고리즘을 사용해서 구한다. 이 문제에서는 메모리 초과가 날 수 있기 때문에 Edmons karps 알고리즘을 '경로를 추적'하는 방식으로 구현한다.

```
if (x, y) != (x1, y1) and (x, y) != (x2, y2):
                in_node = get_in_node(x, y)
                out_node = get_out_node(x, y)
                adjacency_matrix[in_node].append(out_node)
                adjacency_matrix[out_node].append(in_node)
                edge_dict[(in_node, out_node)] = 1
                edge_dict[(out_node, in_node)] = 0 # in node / out node 만들어 두기 -> 여기에만 weight
```

```
...
(중략)
for nx, ny in [(x-1,y), (x+1,y), (x,y-1), (x,y+1)]:
                if 0 <= nx < N and 0 <= ny < N and (nx, ny) not in
obstacles:
                    in_node = get_in_node(nx, ny)
                    adjacency_matrix[out_node].append(in_node)
                    adjacency_matrix[in_node].append(out_node)
                    edge_dict[(out_node, in_node)] = float('inf')
                    edge_dict[(in_node, out_node)] = 0
...
(중략)
while v != source:
            u = parent[v]
            if flow <= edge_dict[(u,v)]: # flow 구하기
                flow = flow
            else:
                flow = edge_dict[(u,v)]
            v = parent[v]
```

Reference

Chat-GPT:

- 각 문제에 대한 test case 요청 (edge case를 찾기 위해 다양한 test case를 제공받았다)
- 메모리 사용 줄이는 방법 질문 (4. 문제에서 3.과 같이 Edmonds/BFS 코드를 구현하면 시간 제한 초과가 생긴다. 이를 해결하기 위한 방법을 물어보았다)