Algorithm 4월 3주차

BFS, Hash, two-pointer, Dijkstra, Combinations, Counter

Table of content

BFS

[프로그래머스] 단어변환

- 단어 변환에서 한 단계에 글자 하나만 달라야 변환에 성공한다.
- Words에 변환 단계에 징검다리로 사용할 수 있는 단어들이 주어짐
- Begin 단어에서 target 단어로 갈 때까지 필요한 단어 변환 횟수를 반환하는 문제

두 개의 단어 begin, target과 단어의 집합 words가 있습니다. 아래와 같은 규칙을 이용하여 begin에서 target으로 변환하는 가장 짧은 변환 과정을 찾으려고 합니다.

1. 한 번에 한 개의 알파벳만 바꿀 수 있습니다.
2. words에 있는 단어로만 변환할 수 있습니다.

예를 들어 begin이 "hit", target가 "cog", words가 ["hot","dot","dog","lot","log","cog"]라면 "hit" -> "hot" -> "dot" -> "dog" -> "cog"와 같이 4단계를 거쳐 변환할 수 있습니다.

두 개의 단어 begin, target과 단어의 집합 words가 매개변수로 주어질 때, 최소 몇 단계의 과정을 거쳐 begin을 target으로 변환할 수 있는지 return 하도록 solution 함수를 작성해주세요.

begin	target	words	return
"hit"	"cog"	["hot", "dot", "dog", "lot", "log", "cog"]	4
"hit"	"cog"	["hot", "dot", "dog", "lot", "log"]	0

풀이법 - BFS

- DFS로 시도했으나 실패.
- 최소 횟수를 구하는 것이므로 BFS(큐 자료구조)를 사용하는 것이 효율적
- 이미 사용한 단어는 사용하지 못하도록 visited를 사용한다.
- 매칭은 반복문을 활용해서 손쉽게 카운트하여 비교

```
visited = [ 0 for i in range(len(words))]
q = deque()
q.append([begin, 0])

while q:
    last, cnt = q.popleft()
    if last == target:
        return cnt
    for j in range(len(words)):
        diff = 0
        if visited[j]:
            continue
        now = list(words[j])
        prev = list(last)
        for i in range(len(now)):
            if prev[i] != now[i]:
                diff += 1
        if diff == 1:
            visited[j] = 1
            q.append([words[j], cnt + 1])
```

Hash, Two pointer

[프로그래머스] 보석쇼핑

- 보석쇼핑을 하는데 모든 종류의 보석을 담아야 한다.
- 시작 인덱스에서 끝 인덱스까지 모든 보석을 담는다. (중간에 빼먹을 수 없음)
- 가장 짧은 길이로 모든 보석을 담아야 한다.
- 그 때의 시작점과 끝점을 반환

1	2	3	4	5	6	7	8
DIA	RUBY	**RUBY**	**DIA**	**DIA**	**EMERALD**	**SAPPHIRE**	DIA

gems	result
`["DIA", "RUBY", "RUBY", "DIA", "DIA", "EMERALD", "SAPPHIRE", "DIA"]`	[3, 7]
`["AA", "AB", "AC", "AA", "AC"]`	[1, 3]
`["XYZ", "XYZ", "XYZ"]`	[1, 1]
`["ZZZ", "YYY", "NNNN", "YYY", "BBB"]`	[1, 5]

Sorting을 사용하면 시간초과

- Sort를 하면 시간초과가 발생한다.

Sorting을 사용하면 시간초과

- Sort를 쓰지 않기 위해 딕셔너리를 사용
- Answer를 양쪽 끝지점으로 지정하고 right – left와 비교하여 계속 업데이트 한다.

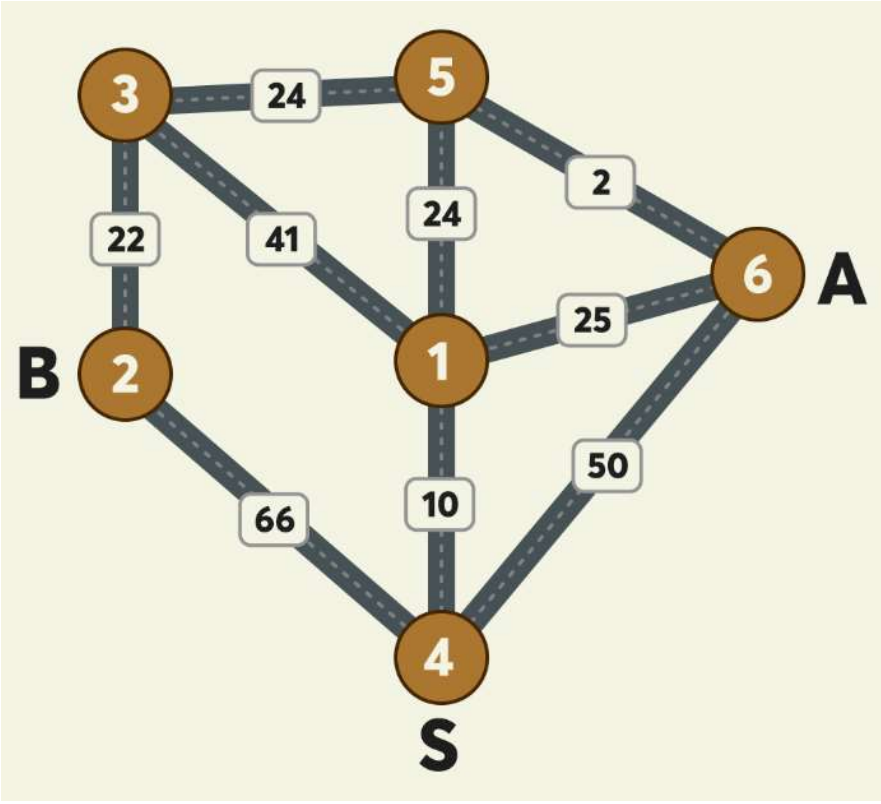
```
def solution(gems):
    answer = [0, len(gems)]
    kind = set(gems)
    dic = {gems[0]:1}
    left, right = 0, 0

    while right < len(gems):
        if len(dic) < len(kind):
            right += 1
            if right == len(gems):
                break
            if gems[right] in dic:
                dic[gems[right]] += 1
            else:
                dic[gems[right]] = 1
        else:
            if (right - left) < answer[1] - answer[0]:
                answer = [left, right]
            if dic[gems[left]] == 1:
                del dic[gems[left]]
            else:
                dic[gems[left]] -= 1
            left += 1
```

Dijkstra

[프로그래머스] 합승 택시 요금

- 출발 노드S 와 도착 노드 A, B가 각각 정해져 있다.
- 간선별로 비용이 기록되어있다.
- 택시비를 절약하기 위해 두사람은 같이 탔다가 중간에 내려서 각자 집으로 다시 택시를 타고 귀가한다.
- 가장 절약할 수 있는 루트에서의 비용을 반환해라.



n	s	a	b	fares	result
6	4	6	2	[[4, 1, 10], [3, 5, 24], [5, 6, 2], [3, 1, 41], [5, 1, 24], [4, 6, 50], [2, 4, 66], [2, 3, 22], [1, 6, 25]]	82
7	3	4	1	[[5, 7, 9], [4, 6, 4], [3, 6, 1], [3, 2, 3], [2, 1, 6]]	14
6	4	5	6	[[2,6,6], [6,3,7], [4,6,7], [6,5,11], [2,5,12], [5,3,20], [2,4,8], [4,3,9]]	18

다익스트라 구현 방법

- 딕셔너리로 그래프 구조 셋팅
- 우선순위큐(힙)에 시작노드 삽입
- Distance 배열 초기화
- 반복문 내 heappop() 사용 -> 거리와 노드 뽑아내기
- 뽑아낸 거리와 노드에서 각 목적지까지의 거리를 합산.
- 합산한 거리가 distance 배열에 기록된 거리보다 더 짧다면 갱신해준다. 그리고 heap에 추가한다.

```
graph = defaultdict(list)
    for node1, node2, fare in fares:
        graph[node1].append([node2, fare])
        graph[node2].append([node1, fare])

def dijkstra(start):
        distance = [inf] * (n + 1)
        distance[start] = 0
        heap = [(start, 0)]
        while heap:
            node, cost = heapq.heappop(heap)
            for arrival, weight in graph[node]:
                new_cost = cost + weight
                if new_cost < distance[arrival]:
                    distance[arrival] = new_cost
                    heapq.heappush(heap, (arrival, new_cost))
        return distance

dijkstra(1)
```

풀이법 – 다익스트라 + 모든 노드 시작점 반복

- 기본적으로 다익스트라는 시작점이 주어질 때 사용가능한 알고리즘
- 이 문제에서는 모든 노드가 시작점이 되어서 최소 비용을 계산해야한다. (그래야 중간에 분산되는 노드 포인트에서 도착지까지의 비용을 계산할 수 있음)

```
from collections import defaultdict
from math import inf
import heapq
def solution(n, s, a, b, fares):
    answer = inf
    graph = defaultdict(list)
    for node1, node2, weight in fares:
        graph[node1].append((node2, weight))
        graph[node2].append((node1, weight))

    def dijkstra(start):
        distance = [inf] * (n + 1)
        distance[start] = 0
        heap = [(start, 0)]
        while heap:
            node, cost = heapq.heappop(heap)
            for arrival, weight in graph[node]:
                new_cost = cost + weight
                if new_cost < distance[arrival]:
                    distance[arrival] = new_cost
                    heapq.heappush(heap, (arrival, new_cost))
        return distance
```

```
route = [[]]
for i in range(1, n + 1):
    route.append(dijkstra(i))

for i in range(1, n + 1):
    answer = min(answer, route[s][i] + route[i][a] + route[i][b])

return answer
```

Counter, Combination

[프로그래머스] 메뉴 리뉴얼

- 손님들이 가장 많이 주문한 단품 메뉴들을 조합해서 세트메뉴를 구성한다.
- 최소 두 사람이상이 주문한 메뉴만 세트 구성이 들어갈 수 있음
- 세트 메뉴에 들어가는 단품 개수가 주어짐
- 가장 많이 등장한 단품 조합만 넣는다.
- 각 단품 개수로 구성된 세트메뉴를 구성하여 반환해라.

코스 종류	메뉴 구성	설명
요리 2개 코스	A, C	1번, 2번, 4번, 6번 손님으로부터 총 4번 주문됐습니다.
요리 3개 코스	C, D, E	3번, 4번, 6번 손님으로부터 총 3번 주문됐습니다.
요리 4개 코스	B, C, F, G	1번, 5번 손님으로부터 총 2번 주문됐습니다.
요리 4개 코스	A, C, D, E	4번, 6번 손님으로부터 총 2번 주문됐습니다.

손님 번호	주문한 단품메뉴 조합
1번 손님	A, B, C, F, G
2번 손님	A, C
3번 손님	C, D, E
4번 손님	A, C, D, E
5번 손님	B, C, F, G
6번 손님	A, C, D, E, H

orders	course	result
`["ABCFG", "AC", "CDE", "ACDE", "BCFG", "ACDEH"]`	[2,3,4]	`["AC", "ACDE", "BCFG", "CDE"]`
`["ABCDE", "AB", "CD", "ADE", "XYZ", "XYZ", "ACD"]`	[2,3,5]	`["ACD", "AD", "ADE", "CD", "XYZ"]`
`["XYZ", "XWY", "WXA"]`	[2,3,4]	`["WX", "XY"]`

[파이썬 특화] combinations, Counter

- 파이썬의 Combinations는 배열의 조합을 내뱉어 준다.
- 자매품으로 permutations도 있음 (순열을 뱉어주는 것)

```
From itertools import combinations

arr = ['A', 'B', 'C']
nCr = combinations(arr, 2)
print(list(nCr))

결과: [('A', 'B'), ('A', 'C'), ('B', 'C')]
```

- 파이썬에는 Counter라는 함수가 있는데, 배열에서 각 원소 등장 빈도를 카운트 해준다.
- Counter에는 most_common() 이라는 메서드가 있다. 가장 숫자가 높은 것부터 내림차순으로 정렬해준다.

```
from collections import Counter

Counter('hello world').most_common()

결과: [('l', 3), ('o', 2), ('h', 1), ('e', 1), (' ', 1), ('w', 1), ('r', 1), ('d', 1)]
```

[프로그래머스] 풀이 – Counter, Combinations

- Itertools의 combinations
- Collections의 Counter
- 조합을 구성해서 후보자 배열에 모두 넣고
- 후보자 배열을 Counter 사용해서 등장 빈도를 센다.
- Most_common()을 사용, 등장 빈도가 많은 순으로 정렬
- 가장 많이 등장하는 메뉴 조합만 answer에 등록시킨다.

```python
from itertools import combinations
from collections import Counter
def solution(orders, course):
    answer = []
    for count in course:
        candidates = []
        for order in orders:
            for li in combinations(order, count):
                candidates.append("".join(sorted(li)))
        counter = Counter(candidates).most_common()
        if not counter:
            continue
        most_freq = counter[0][1]
        if most_freq == 1:
            continue
        for i in range(len(counter)):
            if counter[i][1] == most_freq:
                answer.append(counter[i][0])

    return sorted(answer)
```