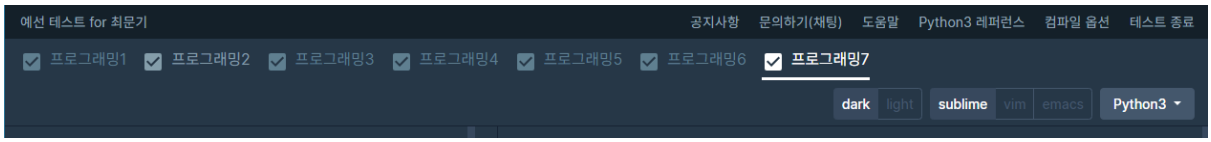


5시간 30분 동안 모두 수고 많으셨습니다.

작년엔 복잡한 구현이 많이 나온데 비하여 올해는 알고리즘 비중이 높다고 생각합니다.

<알고리즘 요약>

1번 : 단순 구현, 문자열

2번 : 조합

3번 : DP, 이분탐색

4번 : 플로이드-워셜

5번 : 라인 스위핑

6번 : 순열, bfs

7번 : 트리 DP

1번 : 단순 구현, 문자열

문제에서 제시하는 7단계를 순서대로 잘 구현하면 됩니다. 문자열은 파이썬을 쓰면 편합니다.

소스코드(파이썬)

```
def is_valid(c):
    if ord('a') <= ord(c) <= ord('z'): return True
    if ord('0') <= ord(c) <= ord('9'): return True
    if c in {'-', '_', '.'}: return True
    return False

def solution(new_id):
    answer = []
    # 1
    new_id = new_id.lower()
    # 2
    for c in new_id:
        if is_valid(c):
            answer.append(c)
    # 3, 4, 5
    nxt = []
    for i in range(len(answer)):
        if i == len(answer) - 1: nxt.append(answer[i])
        elif answer[i] == answer[i + 1] == '.': continue
        else: nxt.append(answer[i])

    if nxt and nxt[0] == '.': nxt = nxt[1:]
    if nxt and nxt[-1] == '.': nxt.pop()

    if not nxt: answer = ['a']
    else: answer = nxt

    # 6
    if len(answer) >= 16:
        answer = answer[:15]
        if answer[-1] == '.':
            answer.pop()

    # 7
    elif len(answer) <= 2:
        while len(answer) != 3:
            answer.append(answer[-1])

    return ''.join(answer)

if __name__ == "__main__":
    print(solution(".........!@BaT#*........y.abcdefghijklm............"))
    print(solution("abcdefghijklmn.p"))
```

2번 : 조합

각 손님의 주문에서 단품 메뉴 조합의 개수를 모두 count합니다. 손님의 주문은 최대 10개이므로 조합이 시간 안에 됩니다. 그리고 각 코스 요리 개수를 만족하는 조합에 대해서 count가 가장 많은 조합을 선택하고 정렬해서 return합니다. 조합, count의 더 빠른 구현을 위하여 파이썬 내장 라이브러리에서 itertools.combinations와 collections.Counter를 사용했습니다.

(알파벳 26개에서 1개, 2개, … , 10개를 선택하는 식으로 구현하면 문제에서 시간 제한이 적혀있지 않지만 시간초과가 뜹니다. 26개에서 10개를 선택하는 방법은 5311735가지 입니다.)

소스코드(파이썬)

```
from itertools import combinations
from collections import Counter

def solution(orders, course):
    answer = []
    C = [Counter() for _ in range(11)]
    for order in orders:
        for n in course:
            if n > len(order): continue
            for sub in combinations(order, n):
                s = ''.join(sorted(sub))
                C[len(s)][s] += 1
    for n in course:
        max_count = -1
        ret = []
        for key in C[n].keys():
            if C[n][key] < 2: continue
            if max_count == C[n][key]: ret.append(key)
            elif max_count < C[n][key]:
                max_count = C[n][key]
                ret = [key]
        answer += ret
    return sorted(answer)

if __name__ == "__main__":
    print(solution(["XYZ", "XWY", "WXA"], [2,3,4]))
```

3번 : DP, 이분탐색

query : [조건]을 만족하는 사람 중 코딩테스트 점수를 X점 이상 받은 사람은 모두 몇 명인가?

쿼리가 들어올 때, 쿼리에 만족하는 자원자들의 수를 출력합니다.

쿼리로 들어올 수 있는 모든 상태의 수가 어떻게 될까요?

개발언어 : “cpp”, “java”, “python”, “-“

직군 : “backend”, “frontend”, “-“

경력 : “junior”, “senior”, “-“

소울푸드 : “chicken”, “pizza”, “-“

쿼리로 들어올 수 있는 모든 상태의 수는 4 * 3 * 3 * 3 = 108가지 입니다.

따라서 풀이는 다음과 같습니다.

1. 108가지 배열에 지원자들의 점수를 채웁니다.

2. 정렬합니다.

3. 쿼리가 들어오면 쿼리를 만족하는 배열을 108가지 배열에서 찾고 이분탐색을 통해서 답을 구합니다.

소스코드(C++)

```cpp
#include <bits/stdc++.h>

using namespace std;

int mapping(string s) {
    if (s == "-") return 0;
    if (s == "and") return 0;

    if (s == "cpp") return 1;
    if (s == "java") return 2;
    if (s == "python") return 3;

    if (s == "backend") return 10;
    if (s == "frontend") return 20;

    if (s == "junior") return 100;
    if (s == "senior") return 200;

    if (s == "chicken") return 1000;
    if (s == "pizza") return 2000;

    return -1;
}
vector<string> split(string str) {
    vector<string> ret;
    stringstream stream(str);
```

```
    string tmp;
    while (getline(stream, tmp, ' '))
        ret.push_back(tmp);
    return ret;
}

vector<int> solution(vector<string> info, vector<string> query) {
    vector<int> answer;
    vector<int> dp[2300];
    int is_sorted[2300] = {0};
    // fill dp
    for (auto s : info) {
        int curr = 0, point = 0;
        for (auto sub_s : split(s)) {
            int tmp = mapping(sub_s);
            if (tmp < 0) {
                point = stoi(sub_s);
            } else {
                curr += tmp;
            }
        }
        int diff[] = {1, 10, 100, 1000};
        for (int i = 0; i < (1 << 4); i++) {
            vector<int> diffs;
            for (int j = 0; j < 4; j++) {
                if (i & (1 << j)) diffs.push_back(diff[j]);
            }
            int total = curr;
            for (auto k : diffs) {
                total -= ((total/k) % 10) * k;
            }
            dp[total].push_back(point);
        }
    }

    // do query
    for (auto q : query) {
        int curr = 0, point = 0;
        for (auto sub_s : split(q)) {
            int tmp = mapping(sub_s);
            if (tmp < 0) {
                point = stoi(sub_s);
            } else {
                curr += tmp;
            }
        }

        if (is_sorted[curr] == 0) {
            sort(dp[curr].begin(), dp[curr].end());
            is_sorted[curr] = 1;
        }

        auto lower = lower_bound(dp[curr].begin(), dp[curr].end(), point);
        answer.push_back(dp[curr].end() - lower);
    }
    return answer;
}

int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios::sync_with_stdio(false);
    vector<string> info = {"java backend junior pizza 150","python frontend senior chicken
210","python frontend senior chicken 150","cpp backend senior pizza 260","java backend junior chicken
80","python backend senior chicken 50"};
    vector<string> query = {"java and backend and junior and pizza 100","python and frontend and
senior and chicken 200","cpp and - and senior and pizza 250","- and backend and senior and - 150","-
and - and - and chicken 100","- and - and - and - 150"};
    solution(info, query);
}
```

4번 : 플로이드-워셜

플로이드 워셜 알고리즘을 통해서 모든 점과 점 사이의 거리를 저장합니다. O(N^3)

그리고 나서 다음의 최솟값을 구합시다.

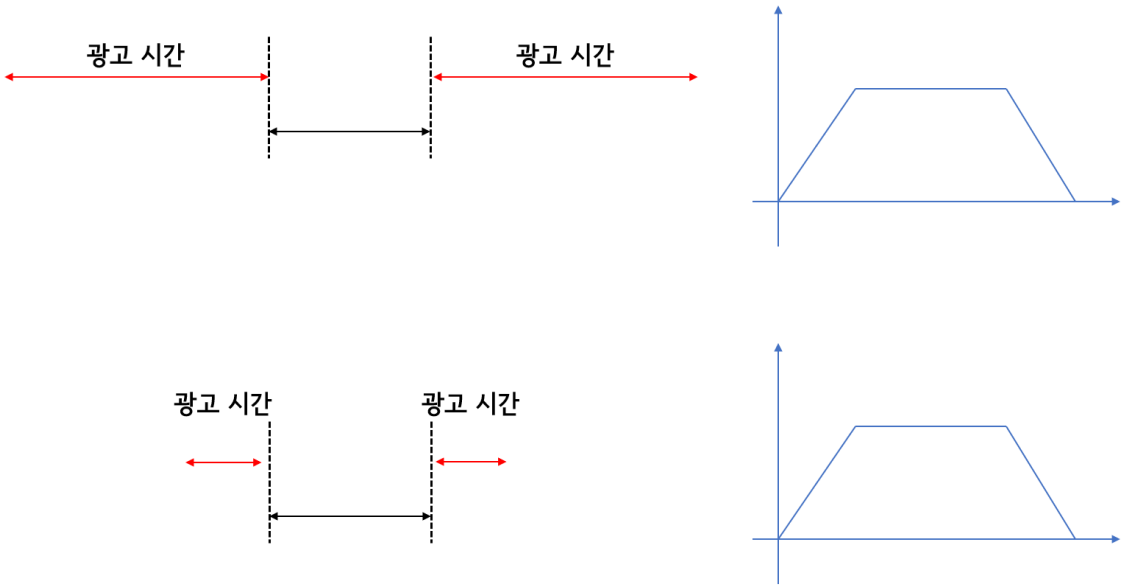
> S에서 i까지 합승하고 i에서 A, i에서 B로 가는 비용
>
> = dist[S][i] + dist[i][A] + dist[i][B]

소스코드(파이썬)

```
INF = 10**18

def solution(n, s, a, b, fares):
    answer = INF
    dp = [[INF] * n for _ in range(n)]
    for i in range(n):
        dp[i][i] = 0

    for i in range(len(fares)):
        c, d, f = fares[i]
        dp[c - 1][d - 1] = f
        dp[d - 1][c - 1] = f

    # Floyd-Warshall
    for k in range(n):
        for i in range(n):
            for j in range(n):
                dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j])

    for mid in range(n):
        ret = dp[s - 1][mid] + dp[mid][a - 1] + dp[mid][b - 1]
        answer = min(answer, ret)
    return answer


if __name__ == "__main__":
    print(solution(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]]))
```

5번 : 라인 스위핑

유사한 문제 : https://www.acmicpc.net/problem/15750



현재 동영상에 대해서 현재 시작시간에 따른 광고할 수 있는 시간을 오른쪽에 그림으로 표현해봤습니다. (x축 시작 시간, y축 광고할 수 있는 시간) 그리고 모든 동영상에 대해서 그려지는 그래프를 모두 더합시다. 더해진 그래프의 최고점을 라인스위핑으로 찾습니다.

그림을 좀 잘못 그렸는데 저 그림에 있는 그래프의 기울기는 1 또는 -1또는 0입니다.

그럼 (시작시간, 그래프의 기울기 변화량)을 스위핑 배열에 넣고 라인 스위핑을 하면 됩니다.

라인 스위핑을 하면서 (현재 x좌표, 현재 y좌표, 현재 기울기)를 관리합시다.

(참고로 세그먼트 트리로도 풀린다고 합니다.)

소스코드 (파이썬)

```
def sec(time):
    h, m, s = time.split(":")
    return int(h) * 3600 + int(m) * 60 + int(s)

def inv_sec(sec):
    h = sec // 3600
    sec %= 3600
```

```python
    m = sec // 60
    sec %= 60
    return f'{h:02d}:{m:02d}:{sec:02d}'

def sweeping(play_time, L, logs):
    periods = []
    for s, e in logs:
        if e - s < L:
            periods.append((s - L, 1))  # up
            periods.append((e - L, -1))  # maintain
            periods.append((s, -1))  # down
            periods.append((e, 1))  # maintain
        else:
            periods.append((s - L, 1))  # up
            periods.append((s, -1))  # maintain
            periods.append((e - L, -1))  # down
            periods.append((e, 1))  # maintain

    periods.append((0, 0))
    periods.append((play_time - L, 0))

    # do sweeping
    max_time = -1
    ret = -1
    y = t = 0
    before_time = -10**18
    for start_time, dt in sorted(periods):
        y += t * (start_time - before_time)
        if 0 <= start_time and start_time + L <= play_time and max_time < y:
            max_time = y
            ret = start_time
        t += dt
        before_time = start_time
    return ret

def solution(play_time, adv_time, logs):
    sec_logs = []
    for log in logs:
        s, e = log.split('-')
        sec_logs.append((sec(s), sec(e)))
    answer_time = sweeping(sec(play_time), sec(adv_time), sec_logs)
    return inv_sec(answer_time)

if __name__ == "__main__":
    print(solution("02:03:55", "00:14:15", ["01:20:15-01:45:14", "00:40:31-
01:00:00", "00:25:50-00:48:29", "01:30:59-01:53:29", "01:37:44-02:02:30"]))
    print(solution("50:00:00", "50:00:00", ["15:36:51-38:21:49", "10:14:18-
15:36:51", "38:21:49-42:51:45"]))
```

6번 : 순열, BFS

카드의 종류는 6가지 입니다.

카드를 없애는 순서는 총 6! = 720가지 밖에 안됩니다.

따라서 모든 순열에 대해서 순서대로 카드쌍을 없애는 답을 구하고 그 중 최솟값을 return합니다.

한 쌍의 카드쌍을 없애는 방법은 총 4가지 입니다.

가능한 현재 위치 : a 또는 b

가능한 다음 위치 : c 또는 d

그렇다면 (a, c, d), (a, d, c), (b, c, d), (b, d, c) 중에 한가지 방법으로 이동합니다.

현재 위치로 가능한 곳은 2개를 넘지 못하지요. 정해진 순열을 따라서 지우면 매번 지운 카드 위에서 끝날 것이고 카드는 최대 2개이기 때문입니다. 그리고 한번 이동할 때마다 최소 이동횟수는 BFS로 구했습니다.

소스코드 (파이썬)

```
from itertools import permutations
from copy import deepcopy

dx = [1, -1, 0, 0]
dy = [0, 0, 1, -1]
INF = float('inf')

def in_board(x, y):
    return (0 <= x < 4 and 0 <= y < 4)

def get_distance(A, _from, _to):
    x0, y0 = _from
    tx, ty = _to
    # BFS
    q = [(x0, y0, 0)]
    vst = [[False] * 4 for _ in range(4)]
    vst[y0][x0] = True
    for x, y, count in q:
        if x == tx and y == ty: return count
        # not Ctrl
        for i in range(4):
            x1, y1 = x + dx[i], y + dy[i]
            if not in_board(x1, y1): continue
            if vst[y1][x1]: continue
            vst[y1][x1] = True
            q.append((x1, y1, count + 1))
        # Ctrl
        for i in range(4):
            x1, y1 = x, y
```

```
            while True:
                if not in_board(x1 + dx[i], y1 + dy[i]): break
                x1 += dx[i]
                y1 += dy[i]
                if A[y1][x1] > 0: break
            if vst[y1][x1]: continue
            vst[y1][x1] = True
            q.append((x1, y1, count + 1))

def find_card(A, card):
    ret = []
    for y in range(4):
        for x in range(4):
            if A[y][x] == card:
                ret.append((x, y))
    return ret

def sub_solution(A, r, c, P):
    queue = [(c, r, 0)]
    for card in P:
        to = find_card(A, card)
        total_dist = [INF, INF]
        for x, y, dist in queue:
            for i in range(2):
                ret = dist
                mx, my = to[1 - i]
                ret += get_distance(A, (x, y), (mx, my))
                ret += get_distance(A, (mx, my), to[i])
                total_dist[i] = min(total_dist[i], ret)
        queue = [(to[0][0], to[0][1], total_dist[0]), (to[1][0], to[1][1],
total_dist[1])]
        for x, y in to:
            A[y][x] = 0
    return min(queue[0][2], queue[1][2])

def solution(board, r, c):
    cards = set()
    for row in board:
        cards |= set(row)
    cards -= {0}
    answer = min(sub_solution(deepcopy(board), r, c, P) for P in
permutations(cards))
    answer += len(cards) * 2
    return answer

if __name__ == "__main__":
    print(solution([[1, 0, 0, 3], [2, 0, 0, 0], [0, 0, 0, 2], [3, 0, 1, 0]], 1,
0))
```

7번 : 트리 DP

트리 DP를 이용해서 풉시다.

dfs(u)는 다음을 구하는 함수입니다.

u가 회의에 참석했을 때 최솟값 u가 회의에 참석하지 않을 때 최솟값

이 때, u가 회의에 참석하지 않으면 적어도 하나의 자식노드(팀원)가 회의에 참석해야합니다.

그래서 마지막에 diff를 더하는데 적어도 한명의 팀원이 참가하기 위한 최소비용입니다.

소스코드 (파이썬)

```
from sys import setrecursionlimit
setrecursionlimit(10**6)

def dfs(u, sales, childs):
    in_leader = sales[u]
    not_in_leader = 0
    diff = float('inf')
    for child in childs[u]:
        a, b = dfs(child, sales, childs)
        not_in_leader += min(a, b)
        in_leader += min(a, b)
        diff = min(diff, a - b)
    if childs[u]:
        not_in_leader += max(0, diff)
    return in_leader, not_in_leader

def solution(sales, links):
    n = len(sales)
    childs = [list() for _ in range(n)]
    for p, c in links:
        childs[p - 1].append(c - 1)
    a, b = dfs(0, sales, childs)
    return min(a, b)

if __name__ == "__main__":
    print(solution([14, 17, 15, 18, 19, 14, 13, 16, 28, 17],[[10, 8], [1, 9], [9,
7], [5, 4], [1, 5], [5, 10], [10, 6], [1, 3], [10, 2]]))
```