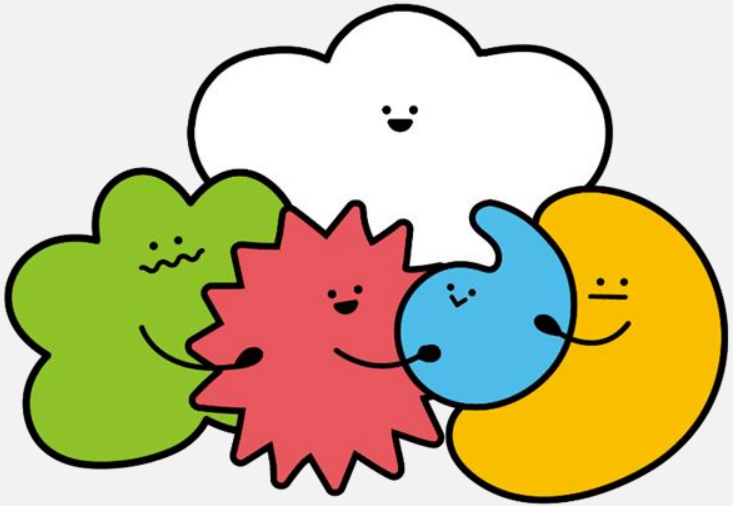


goorm

세미나

Contents.

1번 빵야

시뮬레이션 문제입니다.

오늘도 구름이는 사랑과 평화를 위해 권총을 들고 적들 앞에 섰다.

적들은 총 N명이며, 1번부터 N번까지 차례대로 번호가 부여되어 있다. 그리고 i번 적은 H_i 만큼의 체력을 가지고 있다. 또한, 적들은 구름이를 향해 1번부터 차례대로 일렬로 서있다.

구름이는 자신과 가장 가까운 순서대로 적을 쓰러뜨리기 위해 권총을 쏠 것이다. 즉, 1번 적부터 쓰러뜨릴 것이다. 구름이가 i번째 발사로 적을 맞췄을 때, 적은 $((i-1) \bmod 4) + 1$만큼의 체력이 깎이게 된다. 적들은 체력이 0 이하가 되었을 때 쓰러지게 된다.

N명의 적들의 체력이 주어질 때, 구름이가 **권총을 발사해야 하는 횟수**를 알아보자.

1번 빵야

시뮬레이션 문제입니다.

- i번 적은 H_i 만큼의 체력을 가지고 있으며, 체력이 0 이하가 된다면 쓰러지게 됩니다.
- 구름이는 1번 적부터 차례대로 쓰러질 때까지 권총을 발사합니다.
- 권총은 i번째 발사일 때, $((i-1) \bmod 4) + 1$만큼의 피해를 줍니다.
- 모든 적을 쓰러뜨리기 위해 발사해야 하는 횟수를 구해야 합니다.

1번 빵야

최적화 시뮬레이션 문제입니다.

문제는 요구 사항을 그대로 잘 구현하기만 하면 어렵지 않다고 생각을 합니다.

다만, 최적화가 필요하죠. 문제의 규칙을 살펴보면 실제로 총이 입힐 수 있는 데미지는
1, 2, 3, 4가 반복됩니다.

이제 최악,, 아주 최악의 경우를 생각해봅시다.

모든 적의 체력이 10^9 이고, 이런 적인 100,000 명 있다고 생각을 해봅시다.
단순하게 한 번씩 총을 쏘면 총 몇 번 총을 쏴야 할까요?

1번 빵야

40,000,000,000,000 번 총을 발사해야 한다.

자 쉽게 이야기하면 단순 반복문이 40조 번 반복되어야 이 값을 계산할 수 있다는 의미입니다.

당연히, 개선이 필요합니다. 시뮬레이션 문제일수록 이 입/출력에 집중해야 합니다. 더불어, 최적화해서 계산을 해도 40조 이상의 값이기 때문에 자료형에 주의해야 합니다.

총알을 발사했을 때, **적의 체력이 10이상이라면 굳이 직접 계산을 하지 않고, 10으로 나눈 나머지와 몫**을 찾습니다.
그리고 총을 발사한 횟수에 **몫 * 4**를 추가하면 됩니다.

나머지로는 이제, 총의 발사 횟수에 따라서 남은 체력을 0으로 만드는 횟수를 찾아서 더하면 됩니다.
어차피 총 발사 횟수와 상관없이 이것도 1, 2, 3, 4를 반복하는 부분만 챙기면 됩니다.

2번 친구 사이

세상에는 **친구 사이**라는 것이 있다. 만약, A와 B가 직접적으로 친분이 있다면 그 두 사람은 친구 사이가 된다. 또한, B와 C가 직접적으로 친분이 있다면, A는 B를 통해 C를 만날 수 있으므로, A, B, C 모두 친구 사이가 된다. 즉, **직접적으로 친분이 있는 사람들을 거쳐 만날 수 있는 사람들 모두 친구 사이**가 된다.

구름이는 친구 사이인 사람이 총 N명이 있다. 이때, N명의 사람마다 1번부터 N번까지 차례대로 번호가 부여되어 있다. 구름이를 포함한 $N+1$명은 친구 사이이기 때문에 자주 만나지만, **구름이는 자기를 제외한 나머지가 전부 친구 사이를 이뤘으면 좋겠다**고 생각했다. 그래서 직접적으로 친분이 없는 두 사람이 친분을 쌓게끔 노력하려고 한다. 하지만 잘 맞지 않는 두 사람도 있어서 친분을 쌓는 데에 오래 걸리는 사이도 있고 아예 친분을 쌓을 수 없는 사이도 있다. 그래서 구름이는 친분을 쌓을 수 있는 사이를 조사해서, **친분을 쌓는 시간의 합이 최소**가 되게끔 친분을 쌓게 도와줘서 **모두가 친구 사이**가 되게 하려고 한다.

구름이와 친구 사이인 사람들 중에서 이미 친분이 있는 사이와 친분을 쌓을 수 있는 사이의 정보가 주어질 때, **친분을 쌓는 시간의 합이 최소가 되게끔 친분을 쌓게 도와줘서 모두가 친구 사이가 될 때의 친분을 쌓는 시간의 합**을 알아보자.

2번 친구 사이

요구 사항 정리

- 직접적으로 친분이 있는 사람들을 거쳐서 만날 수 있는 사람들 모두 친구 사이가 됩니다.
- N명의 사람들 중 친분이 있는 사이는 총 M쌍이 있습니다.
- 친분을 쌓을 수 있는 사이는 총 K쌍이 있습니다. 친분을 쌓는 데에 임의의 시간이 걸립니다.
- N명이 서로 친구 사이가 되기 위해 친분을 쌓는 데에 걸리는 시간의 총합의 최솟값을 구해야 합니다.

알고리즘 정리

- 최소 스패닝 트리를 구하는 문제입니다.
- 시간복잡도는 $O(M + K \log K)$ 또는 $O((M + K) \log N)$입니다.

2번 친구 사이

이런 문장이 나온다면, MST입니다.

모든 노드들이 연결되어 있는 그래프를 만들고 싶다.
이때 간선을 연결할 때 **비용이 발생하는데 그 비용이 최소**가 되도록 하고 싶다.

MST는 어느 정도 알고리즘, 코딩테스트 공부를 했다면 익숙하면서도 불편한 파트입니다.
기본적으로 최소 스패닝 트리라고 불리는데,
이 트리의 특징은 **가중치가 있는 그래프에서, 모든 정점을 연결하는 가중치의 합이 최소가 되는 트리**를 찾는 알고리즘입니다.

그렇다면 이 문장을 각각 뜯어서 개념을 좀 잡아보려고 합니다.

2번 친구 사이

그래프와 트리

자 그래프의 개념과 트리의 개념 중에서 뚜렷한 차이를 보이는 건 무엇이죠?

그래프는 모든 노드가 연결될 필요가 없고, N개의 노드와 M개의 간선으로 노드 사이의 관계(양방향)나 종속성(단방향)을 표현한다고 했습니다. 때문에 사이클도 존재할 수 있으며, 노드 사이의 경로는 여러 개가 나올 수 있습니다.

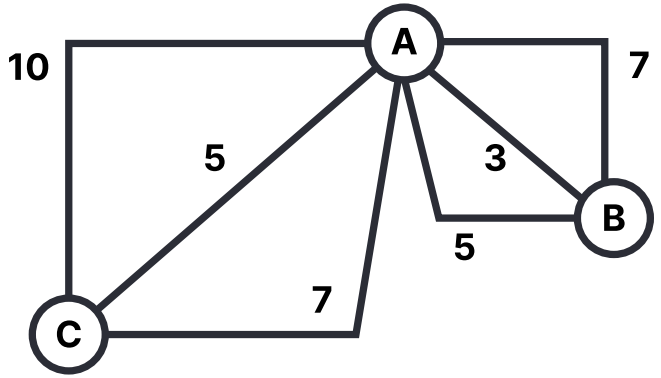
트리는 그래프 중에서, N개의 노드와 N-1개의 노드로 모든 노드를 연결한 상태라고 했습니다. 때문에 사이클이 존재할 수 없고, 노드 사이의 경로는 유일하다고 했습니다.

이때 스패닝 트리 개념과 트리의 개념이 같습니다.
그리고 복잡한 구조의 그래프에서는, 꽤 많은 개수의 스패닝 트리가 있을 수 있습니다.

2번 친구 사이

스패닝 트리와 MST(최소 스패닝 트리)

아래와 같은 그래프가 있다고 합니다. 정말 단순한 구조이지만 이 구조에서 **9개의 스패닝 트리**를 찾을 수 있습니다.
그래고 9개의 스패닝 트리에서, 우리는 간선의 비용이 8인 스패닝 트리를 찾을 수 있습니다.
이 트리가 바로 최소 스패닝 트리입니다.
MST는, 그래프에서 구성할 수 있는 스패닝 트리 중에서 간선의 비용 총 합이 최소가 되는 스패닝 트리입니다.



2번 친구 사이

MST를 찾을 수 있는 2가지 알고리즘

MST를 찾는 방법은 2가지가 있습니다. 첫번째는 **간선의 비용**을 중심으로 찾는 방법이고, 다른 하나는 **정점을 중심**으로 찾는 방법입니다. 다만, 둘 다 그리디한 선택을 강요하는 알고리즘 이라는 점도 참고해주시면 좋을 듯 합니다.

간선의 비용 중심인 알고리즘은 **칼루스칼 알고리즘**이고, 정점이 중심인 알고리즘은 **프림 알고리즘**입니다.
명확한 코드는 제가 오늘 참고 자료를 2개 추가해서 올려두었어요. 해당 내용을 읽고 코드로 바꿔보면 될 듯 합니다.

중요한 건, 이 MST 유형이 은근하게 나오는 국밥 유형입니다. 왜-일까요?

MST가 그래프 구조로 낼 수 있는 사실상 최상위 개념입니다.
이보다 더 상위 개념으로 올라가면 단순한 코딩테스트의 목적이 아닌, 경진대회 문제에 가까워집니다.
경진대회용 문제를 내는 회사가 몇 군데 있기는 하지만, 아마 일반적인 입사 시험에서는 보기 어렵습니다.
물론, 최근에 삼성전자에서 경진대회용 문제가 나오기는 했지만, 이는 전력-외로 판단하기로,,

중요한 건 상위 알고리즘 개념이 나올수록 구현 부분은 조금 빠지고 알고리즘을 적용할 수 있는 유형의 문제가 많습니다.
딱 MST가 그런 문제로 내기 좋아서 국밥같이 어려운 유형이 필요할 때 나옵니다.

3번 Run and Fly

일단 그래프 탐색 문제 맞습니다. 근데,, 이분 탐색을 곁들인..

천사 마을이 악마들에게 1차 침공을 당했다. 그리고 T초 후에 2차 침공이 있을 예정이어서 천사 마을에 있던 모든 천사들이 대피하느라 바빴고, 천사 구름이 또한 마찬가지였다.

천사 마을은 N개의 행과 M개의 열로 이루어진 격자 모양이며, **맨 왼쪽 위 칸은 구름이의 집**이고 **맨 오른쪽 아래 칸은 탈출구**이다. 또한, 각 칸은 땅, 하늘, 쑥대밭 중 하나이며, 이때 반드시 **구름이의 집과 탈출구가 있는 칸은 땅**이다.

각 칸에서 **상하좌우로 인접한 칸으로 이동할 때 1초가 걸린다.** 물론, 쑥대밭인 칸으로는 갈 수 없다. 그리고 땅에서 땅으로 갈 때에는 날지 않아도 된다. 하지만 **땅에서 하늘로 갈 때, 하늘에서 하늘로 갈 때, 하늘에서 땅으로 갈 때에는 반드시 날아가야 한다.** 그런데 여기서 문제가 있다. 천사는 자유롭게 날 수 있지만, 구름이는 아직 어리기 때문에 **에너지**라는 요소를 사용해 날 수 있다. 구름이는 처음에 K만큼의 에너지를 갖고 출발한다. 에너지는 **1초 날 때마다 1씩 소모**되며, **땅에서 움직이지 않고 1초 쉴 때마다 1씩 충전**된다. 이때, **에너지가 0 미만으로 내려가면 구름이가 기절**한다. 그리고 **에너지가 K를 넘어가도록 충전하는 것은 불가능**하다.

구름이의 집에서 출발해서 탈출구까지 기절하지 않고 T초 이내로 도달하기 위한 **K의 최솟값**을 알아보자.

3번 Run and Fly

요구 사항을 정리하면,

적기 귀찮아서 가져왔습니다.

- 천사 마을은 N개의 행과 M개의 열로 이루어진 격자 모양입니다.
- 구름이의 집은 맨 왼쪽 위 칸이며, 탈출구는 맨 오른쪽 아래 칸입니다.
- 각 칸은 땅, 하늘, 쑥대밭 중 하나이며, 구름이의 집과 탈출구가 있는 칸은 항상 땅입니다.
- 각 칸에서 상하좌우로 인접한 칸으로 이동할 수 있으며, 한 칸 이동할 때 1초가 걸립니다.
- 땅에서 하늘, 하늘에서 하늘, 하늘에서 땅으로 갈 때에는 1만큼의 에너지를 소모합니다.
- 땅에서 움직이지 않고 1초 쉰다면 에너지가 1만큼 충전됩니다.
- 이때, 에너지는 0 미만이 되게 소모하거나 K 초과가 되게 충전이 불가능합니다.
- 구름이가 집에서 탈출구까지 T초 이내로 도달하기 위한 K의 최솟값을 구해야 합니다.

3번 Run and Fly

1번 예시를 봤을 때, 아래 2가지 경로가 있습니다.

3번 Run and Fly

그래프 탐색을 하기 전에, 우리가 먼저 정의해야 하는 게 있어요.

우리가 찾으려고 하는 값이 무엇인가요? 이전의 그래프 탐색 문제와는 확실히 차이가 있습니다.

거리나, 비용을 찾는 문제가 아니라 필요한 에너지의 최소값을 찾는 문제예요.
일단 문제의 요구 사항이 독특하기 때문에 문제를 풀어낼 전략을 짤 필요가 있습니다.

모든 칸이 하늘이라고 가정했을 때, 필요한 최대 에너지는 **N x M - 1**이 됩니다. 반대로 모든 칸이 땅이라고 가정했을 때 필요한 에너지는 **0**이겠죠?

즉, K는 항상 0 이상, N x M – 1 사이의 어떤 값 입니다.

3번 Run and Fly

배열에서 조건에 맞는 작거나/큰 값을 찾는 알고리즘은 무엇일까요?

기본적으로 우리는 완전탐색을 떠올릴 수 있습니다. 이 경우에는 배열의 길이가 N x M 이기 때문에, O(NM)의 시간복잡도가 발생하게 됩니다.

생각보다 큰 값이고, 따지고 보면 N^2이 됩니다. 이 값에서 그래프 탐색의 시간 복잡도까지 고려한다면 상황에 따라 N^4까지 커질 수 있습니다.

즉, 우리는 가능성 있는 K중에서, 최소값을 찾는 시간 복잡도를 줄일 필요가 있습니다.
그 대표적인 방법이 바로 이분 탐색입니다.

3번 Run and Fly

이분 탐색으로 에너지의 최소값 찾기

2	0	2	2	2
2	1	1	1	1
2	0	0	0	2
2	0	2	2	2
2	0	2	0	2
2	2	2	0	2

예시로 우리가 상상을 했을 때, 에너지가 K가 있다고 합시다.
우선 경로상 T는 최소 9초가 필요합니다.
이 문제에서 **T의 값이 9보다 작다면 탐색을 하지 않으면** 됩니다. 이게 첫번째 탐색 조건이 됩니다.

그리고 그 다음 필요한 에너지를 구할 때, 이분 탐색이 필요합니다.
만약에 예시에서 필요한 에너지는 최소 5입니다. 이것보다 부족하면, 에너지가 0이 되었을 때, **하늘을 나는 구간을 통과**하지 못하기 때문입니다.

우선 각각 **T와 K에 최소값**에 대한 이해가 되었나요?

3번 Run and Fly

이분 탐색으로 에너지의 최소값 찾기

2	0	2	2	2
2	1	1	2	1
2	0	0	0	2
2	0	2	2	2
2	0	2	0	2
2	2	2	0	2

자 그럼 다음과 같은 상황으로 한 번 따져보면 조금 다른 결과가 나옵니다.
동일하게 T가 9라고 합시다.
이때, K의 최소값은 몇이 필요할까요?

3번 Run and Fly

이분 탐색으로 에너지의 최소값 찾기

2	0	2	2	2
2	1	1	2	1
2	0	0	0	2
2	0	2	2	2
2	0	2	0	2
2	2	2	0	2

자 그럼 다음과 같은 상황으로 한 번 따져보면 조금 다른 결과가 나옵니다.
동일하게 T가 9라고 합시다.
이때, K의 최소값은 몇이 필요할까요?

순간적으로 가장 긴 "2 – 1 – 1 – 2 " 구간을 기준으로 3이라고 생각할 수 있지만, 사실은 동일하게 에너지의 최소값은 5입니다.

바로 T와 K가 관계가 있기 때문입니다. 즉, 최소 에너지 K를 특정하기 위해서 완전 탐색으로 그 값을 구했더라도 실제 T보다 크다면 배제해야 합니다. 그리고 그 중에서 최소의 K값을 찾는 방법으로 즉, 완전 탐색의 방법으로 K를 찾기에는 너무 복잡합니다.

3번 Run and Fly

이분 탐색으로 에너지의 최소값 찾기

2	0	2	2	2
2	1	1	2	1
2	0	0	0	2
2	0	2	2	2
2	0	2	0	2
2	2	2	0	2

이를 단순하게 해보려고 합니다.

최대치 K로 T초안에 통과할 수 있다고 합시다. 만약에 통과할 수 없다면 최소치 T를 만족하지 못하기 때문에 -1을 출력할 수 없겠죠?

통과할 수 있다면, K를 반으로 줄입니다.

그리고 (K + 0)//2로 T초안에 통과할 수 있는지 확인합니다.
통과할 수 있다..? -> K//2를 또 2로 나누면 되겠죠?

통과할 수 없다면, (K + 최대치K) // 2로 K의 값을 올립니다.

이 개념을 완전탐색보다 더 빠르게 최소치 K를 찾을 수 있습니다.

3번 Run and Fly

그래프 탐색 자체도 조금 복잡해요.

단, 풀이 방법을 찾는 것도 복잡하지만, 실제로는 그래프 탐색 코드 역시 복잡합니다. BFS/DFS 는 스택 메모리 이슈가 어디에나 존재하기 때문에, 편하게 어떤 걸 사용해도 됩니다.

단, 에너지라는 요소가 있기 때문에 조금 더 복잡한 방문 관리 변수가 필요합니다.

Visited[i][j][k] : (0, 0) 위치에서, (i, j) 위치로 이동할 때 사용한 k에너지의 남은 값
첫 탐색 값 : (0, 0, 최대치 K)로 하여, 에너지가 달 때 마다 K-1을 한다.

그리고 실제 탐색을 할 때, 다음의 탐색 조건을 걸면 되겠죠?

1. 다음 탐색하는 위치가 유효한지?
2. 탐색하는 위치의 값이 현재 2이고, 다음 위치가 2인가?
   즉, 땅에서 땅으로 이동하는 경우 탐색 후보에 (i, j, k) 로 추가
3. 2번에 해당하지 않는가?
   즉, 땅에서 하늘, 하늘에서 땅, 하늘에서 하늘로 이동하는 경우 탐색 후보에 (i, j, k – 1) 로 추가하여 에너지를 소비한다.
4. 마지막에, 탈출구에 K를 남기고, T초안에 도착한 경우가 있는지를 확인하여 return True or False 를 한다.

3번 Run and Fly

맞추지 말라고 낸 문제입니다.

사실 시험 현장에서 완전 탐색까지는 충분히 떠올릴 수 있다고 생각을 합니다.

종이도 없고 또 앞에 2문제 풀었으니 대략 1시간 남았을 시점에 이 문제를 마주하면
대부분 응시자는 물론 저도 역시 이분 탐색까지 생각하기는 어려웠을 꺼 같아요.

올해 꽤 많은 기업(OO전자는 대부분)들이 마지막 문제에 이렇게 킬러 문항을 넣고 있습니다.
저는 개인적으로 킬러 문항은 완전 탐색으로도 도전해볼 시간을 만들고 도전해보는 게 좋은 거 같아요.

이거 완전 탐색으로 풀어도 60점 이상 가져갈 수 있게 설계된 문제거든요.
당연히 정답을 맞출 수 있으면 좋겠지만, 부분 점수도 꽤 큽니다.
시험마다 다르지만, 어려운 문제의 경우 배점이 다른 경우도 있어요.
만약에 이 문제의 배점이 400점이라면..? 아마, 정답 하나 맞춘 것과 거의 비슷한 효과 입니다.

물론 풀어낸다면, 아마 거의 우수한 지원자로 보이기는 하겠죠..?

Contents.

주로 어떤 알고리즘부터 공략하면 되나요?

구현이 아닌, 알고리즘으로 접근을 했을 때..

1. 탐색에 대한 이해가 있으면 좋습니다.
   제 생각에 완전 탐색과 이분 탐색에 대해서는 공부 초기에 잡아 두는 게 정말 좋습니다.
   지금 문제의 방향이 그래프 탐색 + 이분/완탐 이거나 누적합 + 이분/완탐 같은 유형이 많아진 거 같아요.
2. 그래프 탐색은 기본입니다. 그래프안에서만 BFS/DFS를 하는게 아닌 배열에서도 그래프 탐색을 할 수 있을 정도로 공부하면 좋아요.
3. 마지막은 DP입니다. 기본적으로 DP는 수학이나 경우의 수, 확률, 가능성 등을 많이 공부할 필요가 있기 때문에 오래 걸려서 나중으로 잡았습니다만, DP까지는 필수입니다.

이외, 필수는 아닌데, 하면 좋은 알고리즘 그리디의 기본 개념과 적용 가능한 상태에 대한 이해, 전반적인 자료 구조를 적용할 수 있는 상태가 어떠한 상태인지, 트리의 상태와 트리에서의 구조적 탐색

마지막으로는 어떤 알고리즘이던지 구현하고, 응용할 수 있어야 공략했다고 할 수 있겠죠?

어디서부터 공부하면 좋을까요?

자주 답변하고 있는 어떻게 답변하면 좋을까 고민했습니다.

사실 알고리즘의 시작점은 당연히 **기초 프로그래밍**이겠죠? 응시하려는 언어에 대한 이해는 필수라고 했습니다.
다음은 프로그래밍 언어를 잘 응용하고 활용할 수 있는지 연습을 조금 했으면 좋겠습니다.

그에 대한 방법으로 제가 **구현 문제를 자주 풀어**보라는 겁니다. 감각적인 부분과 더불어 코드를 작성하는 속도가 다릅니다.
복붙개발자들이 코테에서 무너지는 이유가 코드를 엄청 느리게 치고, 오탈자도 너무 많더라구요. 이게 의외로 발목을 잡습니다.

다음을 알고리즘을 공부해야겠죠.. 이 방법에 대해서 순서는 아까 알려드렸습니다.
그렇다면 어떻게 공부하느냐? 에 대해서 이거는 오늘 해드릴 이야기에서 답변 드리겠습니다.

자주 출제되는, 올해 유형, 방향성 etc..

9월말부터, 11월까지 어마어마하게 많은 코테가 예정되어 있습니다.

초반 부 리뷰 역시 조금 있다가 하겠습니다.

엣지케이스 고민하기

사실 유형마다 다른데, 저만의 TIP입니다.

구현 문제는 주어진 입/출력으로 만들 수 있는 최악의 경우를 생각합니다. 그리고 그 크기를 줄인 상태로 테스트를 해보는 겁니다. 오늘 1번 문제에서 10^9 를 계산하는 과정을 10번하면 얼추 40,000,000이 나오고 이거에 10^5를 곱하면, 얼추 값을 예상할 수 있죠? 입/출력으로 극단적인 사례를 조금 작게 계산해보는 게 좋아요.

동적프로그래밍 문제의 경우에는 항상 소수로 테스트를 해보는 게 좋습니다. 1차원 배열의 DP라면 2, 3, 7, 9, 11 까지는 반드시..! 11이 진짜 킥입니다.
2차원 배열이라면 여유가 된다면 7*7 까지 극단적인 값으로 확인하는 편이 좋습니다. 탐색 제한이 있다면 탐색 제한이 있는 칸으로 가득 채우거나, 모두 다 열어 두거나, 혹은 0이 되는 경우라던가, 문제 상황마다 다르지만 꼭 이 범위까지는 확인해야 합니다.

그래프라면 본인이 직접 구조를 짜되, 노드는 7개로 잡는게 좋습니다. 보통 7개 노드로 순환하는 원구조, 비순환 원구조, 사이클이 2개 생기는 경우, 컴포넌트로 잘라지는 경우, 2개의 컴포넌트와 단독 노드 형태를 모두 구성할 수 있고 예외 처리도 많이 넣어볼 수 있습니다. 예외 처리는 문제 상황에 맞게 본인이 요구 사항에서 취약점을 찾아야 합니다.

문제 선택에 대한 고민들..

문제 많이 풀기, 적당히 풀기, 어떤 문제를 선택해야, 하루 몇 개,,

이 절충안은 사실 좋은 문제를 많이 풀기가 되는 거 같더라구요? 제가 다른 사이트 홍보는 하지 않겠다는 그 약속을 어기고 전달합니다.

백준에서, 다들 알고리즘 분류로 가시죠?
대회 탭을 가보세요. 거기 있는 대회 문제 세트가 진짜 진국입니다.
구현부터 고난이도 유형은 물론, 실제 기업 문제를 출제하는 제작자들도 많이 있습니다.

2시간 정도 잡고, 몇 문제 푸는 지 계속 확인하면서 공부하면 도움이 많이 됩니다. 사실상 모의고사예요. 물론 문제가 좀 어렵습니다만, 어렵게 공부하면 당연히 본 시험이 쉽습니다.

근데 이게 단점이 풀이를 구하기 어렵습니다. 이것도 역시 조금 있다가 설명해드릴께요.

하지만, 단순한 공부와 준비가 아니라, 본격적으로 코딩 테스트를 준비한다면 해당 플랫폼에서 제공하는 공간에서 연습하세요.
구름의 경우 구름 레벨에서 이미 다양한 문제를 제공하고 있습니다. 곧 리뉴얼 계획도 있으니, 참고해주세요!

언어 선택

개발을 하는 언어를 선택하되, 서브 언어로는 Python3

개발하는 언어로 준비하는 게 제일 좋습니다만, 하나를 더 준비한다면 반드시 python3 로 준비하세요.

병행은 없습니다. 병행하면 더 어렵고 문법만 헷갈릴 수 있어요

이외 팁

문제를 몰라도 일단 요구 사항을 구현하는 연습이 필요하다.

수시채용 언어제한은 회사로 문의하면 의외로 언어 제한은 잘 풀어준다.

백준에서 공부해도 충분한데, 실행환경 준비는 해당 플랫폼에서 한다.

정확도가 틀린다면, 본인이 자주 실수하던 부분부터 확인하세요.

문제를 풀 때는 항상 요구 사항을 정리하는 연습을 꾸준히 하세요.

코딩테스트까지 시간이 많다면 한 문제를 깊게, 적다면 요구 사항에 따른 알고리즘 선택을 이해하고 구현 방법을 암기하고 변형이 되는 위치만 이해해야 한다.

Contents.

하반기 공채가 시작되었습니다.

토스는 고개를 들어주세요.

현재까지 제가 모든 코딩테스트를 응시하고 있지는 않고, 후기, 정보들을 미뤄봤을 때 올해 시험 전체적으로 어려웠습니다.
생각보다 많이 어려웠어요. 일단 아래의 부분들이 핵심인 거 같습니다.

1. 시간안에 풀 수 있는 문제와 풀 수 없는 문제를 구별해내기
   어려운 주제인 거 같아요. 킬링 문제를 제외하고, 나머지 문제들에 난이도 수준이 비슷해요.
   엄청 빡빡한 구현 문제와 다음에 이어지는 알고리즘 문제의 난이도가 엇비슷합니다.

2. 보통은 3문제 2시간이 스탠다드이지만, 토스는 10문제(서/논술형 6문항 + 알고리즘 4문항)에 2시간이 나왔습니다.
   난이도 자체도 매우 어려웠습니다. 같은 시간에 여러 개의 시험이 있었는데, 토스 보신 분들이 제일 손해봤다고 하는 후기가 지배적이긴 합니다.

3. 원래는 코딩테스트를 사용하지 않았던 기업 중에서 코딩 테스트를 채택한 기업들이 많이 생겼어요.

4. 정말 폭-넓은 알고리즘들이 출제되었습니다. 기출, 기존의 그래프라는 유형이 반드시 나온다.. 이런 규칙은 사라진 거 같아요.

하반기 공채가 시작되었습니다.

다양한 알고리즘과 알고리즘의 조합

4. 에 대해서 조금 더 이야기하자면, 이전에는 분명 그래프나 DP의 응용이 나왔습니다. 응용이라는 건 기본적인 그래프 탐색의 구현과 응용, DP의 구현이나 점화식을 활용한, 문제를 이해하고 적용하는 이런 경향이었습니다.

다만 올해는 어떤 느낌이냐? 했을 때 **2개 이상의 알고리즘을 조합한 문제**가 정말 많았습니다.
기본적으로 그래프 탐색을 진행하면서, 그 결과에 따른 추가적인 탐색이나, 누적합을 결과를 바탕으로 또 다시 이분 탐색이 필요한 방향의 문제들도 나왔습니다.

다양한 알고리즘은 이전에 잘 등장하지 않던 **K-냅색, 분리집합, 세그먼트 트리** 등 다양한 알고리즘이 나왔습니다.
물론 회사마다 경향 자체는 달랐지만 기존에 잘 안 나오던 자료구조와 알고리즘이 나온 것은 주목할만한 부분입니다.

하반기 공채가 시작되었습니다.

어떻게 공부해야 할까?

저는 문제의 기조가 코딩테스트에 맞는 기조라고 생각은 합니다. 하지만 방향은 조금 안 좋은 거 같아요.
문제가 어렵지는 않았어요. 어렵지 않다는 게, 해당 알고리즘을 이해하고 적용만 할 수 있으면 대부분 쉽게 풀리기는 했어요.
(몇몇 기업을 제외하고 제가 받은 자료만 봤을 때는)

그 문제가 바로 어떤 알고리즘을 적용하는가? 에서 오는 문제가 성적의 차이를 만들어낸 거 같습니다.
대량으로 문제 푸는 것도 중요하지만 조금 스마트하게 풀 필요도 생긴 거 같아요.

공부 방법에 왕도는 없다고 하지만, 조금 더 스마트하게 하기 위해서는 제가 강조하는 문제의 요구 사항을 정리하고
해당 요구 사항 중에서 어떤 표현과 어떤 상황이 어떤 알고리즘으로 나아가는 지 외우고 공부할 필요가 있는 거 같아요.
요구 사항에 따른 알고리즘이 하나만 나오지 않을 가능성이 높기 때문에, 각각의 알고리즘의 핵심이 무엇인지 이해도 필요할 듯 합니다.

제가 최근에 소통방에 답글을 단 것처럼 최악은 **수능처럼 가는 겁니다.** 저는 이제 좋은 방향은 아니라고 생각은 해요.
하지만, 정량적인 측정 방법이 필요해진 상황에서는 어쩔 수 없는 선택이라고 생각이 되기는 합니다.

Contents.

LLM으로 알고리즘 공부하기

제가 문제 유형 다음으로 가장 많이 받은 질문입니다.

LLM으로 문제 푸는 거 어떻게 생각하세요?

이 질문에 저는 사실 부정적이었답니다. 당연히 손으로 작성한 그 코드(모르면 손코딩하면 이해할 수 있어..)가 무조건 도움이 된다 생각하는 꼰대적 마인드 이제는 여러분들 위해서 버리겠습니다.

이 아젠다를 팀 안에서 조금 해결해보기로 했어요. 그래서 실제로 LLM이 얼마나 문제를 잘 풀고, 이해하는 지와 더불어 어떻게 하면 공부에 도움이 될 지 연구를 조금해봤습니다.

사실 저는 LLM 파인 튜닝 경험도 있다는 점..

LLM으로 알고리즘 공부하기

어떻게 하면 공부가 될까?

제가 다양하게 시도해봤어요.

문제가 사실 쉬우면 문제를 캡처해서 전달만 해도 정말 잘 문제를 풀더라구요?
하지만 이 방법은 공부하는 데 전혀 도움이 되지 않는다고 생각을 했습니다.

우선 문제의 요구 사항을 정리해서, 전달을 해봤습니다.
요구 사항을 직접 명확하게 작성하는 연습을 해본다고 생각을 했어요.

요구 사항은 아래의 구조로 정리해서 줬습니다.

사용하는 자료구조에 대한 정의
무엇을 어떻게 처리해야 하는 지에 대한 정의
어떤 출력값을 찾아야 하는지에 대한 정의
입/출력에 대한 내용 서술

LLM으로 공부하기

금융권 2차원 DP, 경우의 수 문제

구름이는 구름카페를 운영하는 사장님이다. 구름카페에는 테이블이 $N \times 3$ 격자 형태로 배열되어 있다. 한 줄에는 3개의 테이블이 있고, 이러한 줄이 총 N줄 있다.

기존에는 모든 테이블에 동시에 사람들이 앉을 수 있었다. 하지만 사람들 사이에서 소음으로 자주 다툼이 발생하자, 구름이는 **테이블 간 거리두기**를 시행해서 일부 테이블에서만 앉아서 커피를 마시거나 작업을 할 수 있도록 하려고 한다. **테이블 간 거리두기**를 시행하게 되면 어떤 사람이 앉아있는 자리에서 **앞뒤**와 **양옆**으로 **인접한 테이블**에는 동시에 사람들이 앉을 수 없게 된다.

구름이는 거리두기 방식에 따라, 사람들이 앉을 수 있는 테이블에 스티커를 붙이기로 했다. 구름이는 충분히 많은 스티커를 가지고 있기 때문에 붙일 수 있는 스티커의 개수에는 제한이 없으며, 스티커를 하나도 붙이지 않을 수도 있다.

구름카페에 있는 테이블의 줄 수 N이 주어졌을 때, 구름이가 테이블에 스티커를 붙일 수 있는 경우의 수를 구해보자. 단, 수가 너무 커질 수 있으므로 경우의 수를 $100\ 000\ 007$로 나눈 나머지를 출력한다.

지금부터 문제의 상황과 조건을 제공하면, 이를 해결할 수 있는 알고리즘을 설명하고 풀이 코드를 제공해줘

N * 3 배열이 N줄로 이루어진 격자가 있습니다.
이 격자에서, 사람들을 배치하려고 합니다.
다만, **배치하는 기준은 사람들의 상/하/좌/우에 다른 사람이 없는 방식**으로 배치한다고 했을 때, 이 격자에 사람을 배치할 수 있는 모든 **경우의 수**를 찾는 문제입니다.

첫째 줄에 N이 주어집니다.

LLM으로 알고리즘 공부하기

알고리즘 확인하기

세상에, 경우의 수를 찾는 문제이기 때문에 동적 프로그래밍을 제안하는 게 인상적이었습니다.
만약 DP의 개념을 모른다면 한 번 읽기전에 어떤 알고리즘인지 공부하고 오는 게 좋겠죠?

이 문제는 **동적 계획법(Dynamic Programming)**을 활용해 해결할 수 있습니다. 문제의 핵심은 한 줄당 3개의 칸이 있는 여러 줄의 격자에서 인접한 칸에 사람이 배치되지 않도록 하는 경우의 수를 구하는 것입니다.

문제를 작은 부분으로 나눠보겠습니다.

문제 접근:

1. 각 줄의 칸은 3개이며, 앞뒤로 인접하지 않도록 배치해야 합니다.
2. 각 칸은 사람이 배치되거나 비어있을 수 있습니다.
3. **동적 계획법**을 사용하여, 줄마다 사람 배치를 고려한 경우의 수를 누적해서 계산합니다.
4. 이전 줄의 배치에 따라 현재 줄의 배치가 영향을 받기 때문에, **상태를 메모이제이션**하며 처리합니다.

LLM으로 알고리즘 공부하기

문제에서 틀리는 부분을 빠르게 찾기

하지만 구현 부분에서 틀렸습니다. 이게 중요한 거 같아요. LLM이 작성한 내용 중에서, 틀린 건 본인이 찾아야 합니다. 이게 진짜 도움이 많이 되는 거 같아요.

아이디어:

- 각 줄에서 3칸을 채울 때의 상태는 다음과 같은 **8가지 조합**이 가능합니다. 각 칸에 사람이 있거나 없으므로:
  - `000` (모두 비어있음)
  - `001` (맨 오른쪽에 사람)
  - `010` (가운데에 사람)
  - `011` (가운데와 오른쪽에 사람)
  - `100` (맨 왼쪽에 사람)
  - `101` (왼쪽과 오른쪽에 사람)
  - `110` (왼쪽과 가운데에 사람)
  - `111` (모두 사람)

LLM으로 문제 풀기

틀린 부분을 개선할 때, 정확히 무엇을 틀렸는지..

이때, 틀린 부분을 수정하여, 다시 코드를 작성 요청해야 합니다. 틀린 부분은 아래처럼 명확하고 개선 방향까지 제안을 해야 합니다. 이게 정말 공부하는 데 도움이 많이 되어요.

설명한 부분 중 틀린 부분, 그리고 어떻게 수정하면 되는지, 그리고 어떤 요구 사항 때문에 그렇게 되는지..!

너가 설명해준 내용 중에서 8가지 조합은 틀린 내용입니다.
실제 조합은
000 100 010 001 101 5가지만 됩니다.

왜냐하면, 양/옆/위/아래 에 사람이 존재하면 안되기 때문입니다.

LLM으로 문제 풀기

100%가 아닌, 90%를 찾기

그리고 LLM에 의존하면 안됩니다. 다시 질문을 하기 전에 제공한 글을 모두 읽어보세요. 설명을 조금 잘못하는 경우가 많아요. 꼭 다시 읽어보면서 이게 정말 맞는지, 또 풀이는 어떤 지 이해하고 넘어가야 합니다.

분명 LLM이 완벽한 정답 코드를 작성할 수 있는 문제도 많습니다. 다만, 잘못된 코드를 정상 코드로 수정하는 과정이 길어야 합니다. 말 그래도 90%코드를 100%로 만드는 과정에서 꽤 공부가 많이 됩니다.
그리고 생각보다 자주 틀리는 부분 적어두었어요.

실제로 작성했을 때, 자주 틀리는 부분

1. 자료형에 대한 오류를 자주 합니다.
2. 입/출력 처리를 잘 못하는 코드가 많습니다. 이거는 직접 수정할 필요가 있어요.
3. 알고리즘과 실제 코드가 다른 경우가 있습니다. 이때는 코드만 복사한 후 새로운 대화를 열어 코드에 대한 설명을 다시 해달라고 해야 합니다.
4. 못 푸는 경우..? 요구 사항을 여러 번 확인하세요 여러분들 푸시는 문제 중에서 못 푸는 거 많이 없습니다. (단, 너무 구현이 필요한 문제는 실수를 많이 합니다. 알고리즘만 이해하고 직접 구현하는 게 더 도움 많이 되어요)

LLM으로 문제 풀기

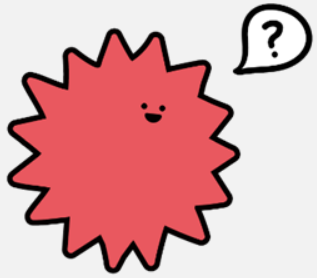
내 것/내 코드로 만들기

그리고 마지막은 문제를 풀고 난 후, 진짜 그 코드를 이해했는가? 는 다른 질문이죠?

보통 LLM은 함수형으로 코드를 작성해줍니다. 이렇게 작성된 코드를 모두 풀어서 **일반형**으로 작성해보세요.
그리고 작성한 **코드를 다시 함수**로 묶어보세요.

진짜 코드 이해는 물론 코드 암기까지 문제 없이 갑니다.
문제 풀이 시간도 많이 줄어들고, 코드 센스나 객체화에 대한 이해도 되더라구요.

저 이걸로 강의 찍어볼까 고민도 해보고 있습니다.



Q&A