1.

출제 의도

Custom 정렬을 구현할 수 있는가?

문자열 처리를 할 수 있는가?

의도한 난이도

실버 2 ~ 실버 1

풀이

먼저 이 문제를 해결하기 위해서는 3가지가 필요합니다. 파일명과 확장자를 잘라내는 것, os에 확장자가 인식할 수 있는지 있는지 판단하는 것, 기준에 맞게 정렬하는 것. 먼저 파일명과 확장자를 잘라내려면 . 을 기준으로 자르면 됩니다. c에서는 strchr로, java나 python에서는 split 등으로 하면 됩니다. 두 번째는 인식할 수 있는 확장자를 set이나 map에 저장합니다. 그런데, 인식할 수 있는 확장자는 삽입, 삭제가 되지 않으니, 모아 놓았다가 정렬 후에 binary search를 이용하셔도 됩니다. 세 번째는 문자열 비교를 하면 됩니다. c에서는 strcmp, c++의 string에서는 compareTo가 이 역할을 합니다. java는 compareTo를 쓰시면 됩니다.

여기서 정렬할 때, 문자열을 비교하는 것은 생각보다 가벼운 연산이 아닙니다. 문자열을 정수 하나로 압축할 수 없을까요? 64^10 = 2^60 < 2^64이므로 long long 형으로 떨굴 수 있습니다. 그러면, string을 비교하는 복잡도가 사라지게 됩니다.

필요한 스킬

문자열 처리, 정렬, 자료 구조, 이진 탐색, 구현

2.

출제 의도

기본적인 문자열 처리를 할 수 있는가?
시간 복잡도와 상수 복잡도에 대한 이해가 있는가?
효율적인 자료구조나 접근 방법을 생각할 수 있는가?

의도한 난이도

실버 2 ~ 골드 5

풀이

메모장에 있는 키워드들은 삭제만 되는 것들입니다. set이나 map을 이용해도 되지만, 삭제만 된다는 것을 생각하면 닭 잡는 데 소 잡는 것을 쓰는 격입니다. 심지어 느리므로, 경우에 따라서는 시간 초과가 될 수도 있습니다. 업데이트가 삭제밖에 없다는 점을 이용하면 좋은데요. 메모장에 있는 키워드들을 정렬 한다면, binary search (이진 탐색)등으로 좌표 압축을 시도할 수 있습니다. 문자열이 x번에 대응된다면, x번을 썼다면 visit[x] = 1로 체크해 두고, 남은 키워드 개수를 하나 감소시키면 됩니다.

그런데 문제에서, 가희가 블로그에 글을 최대 M개를 쓰게 됩니다. 글 하나 당, 키워드가 10이고, 여기에 M을 곱하면 10M이 됩니다. N개의 키워드와 10M개의 키워드에 대해서, 모두 자료 구조에 넣으면 메모리가 얼마나 될까요? (N+10M)인데, N = 20만, M = 20만이라 하면 이 값은 2.2M이 됩니다. 여기에 11을 곱하면 25M가 되게 됩니다. 메모리는 둘째 치더라도, 각 키워드 마다 log(N+10M)의 복잡도로 찾게 되는데, log(220만)과 log(10만)은 약 1.3배 입니다.

시간에 생각보다 빡빡하므로, 상수가 중요한데, 1.3배의 차이는 생각보다 무시 못합니다. 물론, fast io를 이용하면 절약 되겠지만. 심지어, string과 같은 문자열은 메모리를 동적 할당하기에, locality가 좋지 않습니다. 심지어, binary search 복잡도에 문자열 길이까지 곱해지면 생각보다 효율이 좋지 않습니다. 키워드 길이가 10 이하이고, 나올 수 있는 문자 종류는 64개 이하이므로 64 진법으로 떨굽니다. 그러면 64^10 = 2^60 < 2^63이므로, 충분히 long long의 범위 (8byte) 에 들어올 수 있습니다. 이제 binary search 하면서 문자열 비교하는 복잡도는 뺄 수 있습니다.

필요한 스킬

이진 탐색, 문자열 처리, 구현

3.

출제 의도

Round robin 알고리즘을 구현할 수 있는가?
상황에 맞는 자료구조를 이용할 수 있는가?

의도한 난이도

골드 5

풀이

1회 구현 문제 중에, event 관련 처리에 대한 언급을 한 적이 있습니다. 이 문제에서는 event를 t초일 때 특정 고객이 들어온 것이라고 정의하겠습니다. 그러면, 아래 알고리즘을 설계할 수 있습니다.

(1) event가 있다면 event의 대상인 고객을 대기 큐의 맨 뒤로 보낸다.
(2) 만약에 대기 큐의 맨 앞에 있는 고객이 뒤로 가야 한다면, 대기 큐의 맨 뒤로 보낸다.

이는 문제 조건에서, 대기 큐의 맨 앞에 있는 고객이 뒤로 갈 때, 은행에 들어온 손님이 있다면, 그 손님 뒤로 보낸다는 조건 때문입니다. 이는, 손님이 대기 큐의 맨 뒤에 선 다음, 대기 큐의 맨 앞에 있는 고객이 뒤로 가야 하는 경우에, 대기 큐의 맨 뒤로 이동한 것과 상황이 같다는 것을 알 수 있습니다.

event 배열에는 아래와 같이 저장하면 될 겁니다.

event[t] : t초에 온 고객에 대한 정보

이건 당연한 부분입니다. 이제, 대기 큐를 어떻게 구현하는 지가 문제입니다. 상황을 생각해 보면, 맨 앞에 있는 유저에 대한 것이 중요하고, 맨 뒤에는 줄을 서는 것만이 중요하므로, 큐나 deque 같이 앞에서 빼고, 뒤에 추가하는 구조가 적합합니다.

필요한 스킬

event driven 처리, 자료구조, 구현

4.

출제 의도

벡터 처리를 할 수 있는가?

상황에 맞는 자료구조를 선택할 수 있는가?

의도한 난이도

골드 3 ~ 골드 2

풀이

먼저, 공격 방향과 원점으로부터 풍선이 있는 방향은 벡터이므로, 이를 **normalize** 해야 합니다. 2가지 경우로 나눠서 처리하면 됩니다.

(1) 둘 중 하나의 좌표가 **0**인 경우
0이 아닌 쪽의 부호를 보면 됩니다. 만약에 **0**이 아닌 정수가 음수라면 해당 값을 **-1**로 바꾸고, 그렇지 않으면 **1**로 바꾸면 됩니다.
ex. **(-53, 0)**은 **(-1, 0)**으로, **(0, -77)**은 **(0, -1)**로, **(83, 0)**은 **(1, 0)**으로 정규화 됩니다.

(2) 그렇지 않은 경우
a의 절대값을 **A**, **b**의 절대값을 **B**라 했을 때 **A**와 **B**의 최대 공약수를 **G**라 하면 **(a, b)**를 **(a/G, b/G)**로 정규화 합니다.

이제, 관통력이 무한인 것을 생각해 봅시다. 이는, 개틀링 거너가 바라보고 있는 방향에 있는 적 모두를 공격한다는 의미입니다. 공격 후에는 적이 사라질 수도 있습니다. 그런데, 모두의 **hp**가 **D**씩 낮아지는 상황을 구현하기가 쉽지는 않아 보입니다.

대신, 해당 방향으로 누적 데미지를 **nj_D**만큼 주었다고 해 봅시다. **nj_D**보다 적의 **hp**가 작거나 같다면, 해당 적은 없어져야 할 겁니다. 즉, 각 방향마다 최소 힙을 넣고, 최소 힙에 있는 **hp**가 **nj_D**보다 크도록 관리하면 됩니다. 추가까지 되는 경우는 조금 더 생각해야 하는데요. 누적 데미지가 **nj_D**인 상황에서 **hp**가 **h**인 적이 추가되었다면, 누적 데미지를 **nj_D+h**만큼 줘야, 해당 적을 제거할 수 있을 겁니다. 따라서, 누적 데미지가 **nj_D**일 때, **hp**가 **h**인 적이 추가되었다면, 최소 힙에 **nj_D + h**를 넣어주면 됩니다.

필요한 스킬

자료구조, 기본 물리학, 수학

5.

출제 의도

시각 처리를 잘 할 수 있는가?

그리디 알고리즘에 대해서 이해하고 이를 응용할 수 있는가?

의도한 난이도

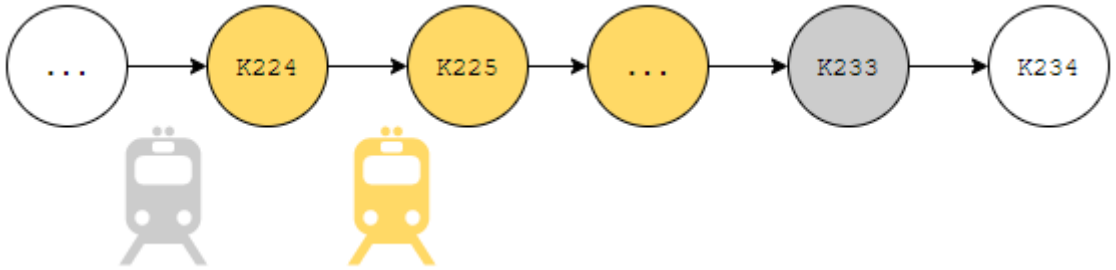
골드 5 ~ 골드 4

풀이

먼저, 이 알고리즘을 생각해 봅시다.

> (1) 모란역에서 죽전역으로 가는 가장 빠른 열차를 탑니다.
> (2) 죽전역에서 고색역으로 가는 가장 빠른 열차를 탑니다.
> (3) 고색역에서 오이도역으로 가는 가장 빠른 열차를 탑니다.
> (4) 오이도역에서 인천역으로 가는 가장 빠른 열차를 탑니다.

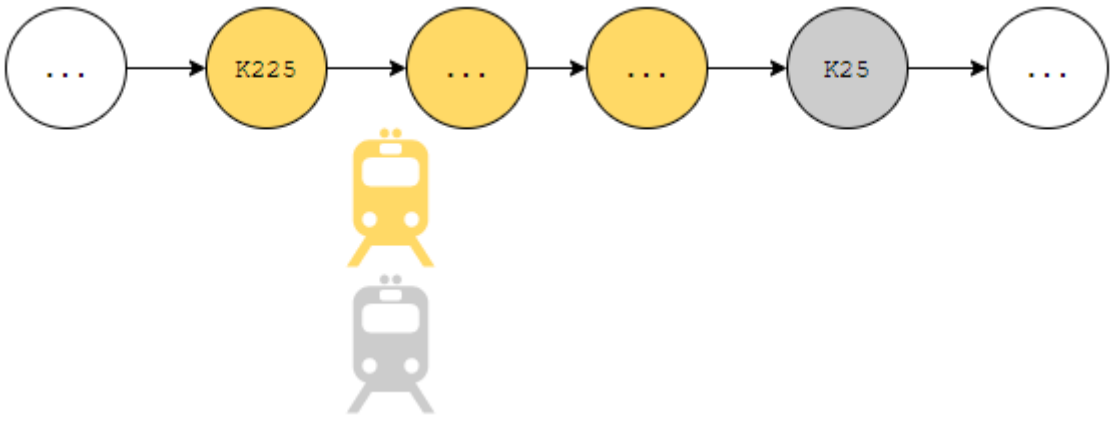
일종의 그리디 알고리즘입니다. 이렇게 하면 제대로 답을 구할 수 있을까요?



[그림 **1]** 급행이 없는 경우

노란색으로 칠한 열차가 고색행이고, 회색으로 칠한 열차가 인천행이라고 하겠습니다. 노란색 열차를 탄 경우에는, 특정한 역에 내려서 회색 열차로 갈아탈 수 있습니다. 반대의 경우, 그렇지 못합니다. 회색 열차를 탄 경우에는 노란색 열차를 타지 못합니다. 이는 노란색 열차가 회색보다 먼저 왔기 때문입니다.

물론, 회색 열차가 급행 열차이기 때문에 노란색 열차를 특정한 역에서 추월하는 경우에는, 회색을 타는 것이 이득일 수도 있습니다.



[그림 **2]** 급행이 있는 경우

그런데, 이러한 상황은 주어지지 않습니다. 왜냐하면, 급행이 있다는 조건이 주어지지 않았기 때문입니다. 즉, 앞에 있는 열차는 종착역까지 가더라도, 계속 앞에 있으므로, 가장 빨리 오는 것을 타고, 죽전, 고색, 오이도 역에서 갈아타는 것이 이득입니다. 질문. 굳이 죽전역에서 내렸다가 탈 필요가 있을까요? 사실 없습니다만, 구현의 단순화를 위해서입니다. 왕십리발 인천행 열차를 타고 쭉 가는 거나, 중간에 죽전에서 내렸다가 다시 같은 열차에 타는 것이나 상황은 똑같기 때문입니다.

시각 처리와 소요시간 처리만 남았습니다. hh:mm은 hh:mm:00으로 처리한 다음에 hh시 mm분 00초가 하루의 몇 초인지 변환하면 됩니다. hh:mm:ss 꼴로 주어진다면, hh시 mm분 ss초가 하루의 몇 초인지 변환합니다. 소요 시간도 마찬가지로 변환하면 되는데요. 문제는 특정 구간입니다.

K210 K211 3 K220 K221 4

이걸 그대로 드래그 해서, 메모장에 복사 붙여넣기 하면 위와 같이 붙여넣기가 될 겁니다. tab이나 공백을 기준으로 split 한 다음에, 1번째, 3번째, 4번째, 6번째 원소만 가지고 오면 됩니다. 이것은 각각 구간, 시간, 구간, 시간으로 써먹을 수 있기 때문입니다. 이제 그리디 알고리즘을 적용해서, 죽전, 고색, 오이도, 인천 순서대로 방문하면 됩니다.

필요한 스킬
시각 파싱, 그리디 알고리즘

6.

출제 의도

문제를 모델링 할 수 있는가?

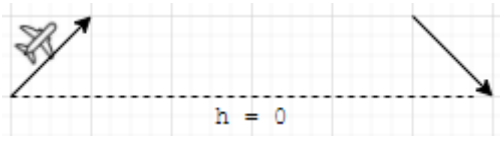
다이나믹 프로그래밍을 아는가?

의도한 난이도

골드 3

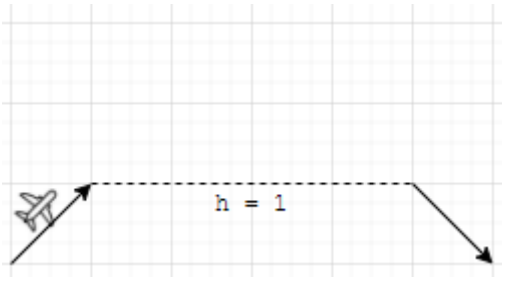
풀이

이 문제는 2가지 풀이가 있습니다. 먼저 카탈란 수를 이용하는 방법입니다.



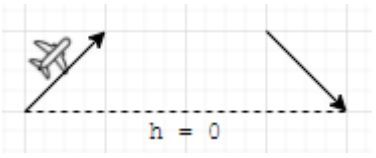
[그림 **3]** 카탈란 수의 개략적인 도식화

카탈란 수는 간단합니다. 거리가 있고, 수평선, 즉 y = 0 밑으로 떨어지지 않는 경우의 수를 계산 합니다. 그런데, 문제에서 요구하는 바는 여기서 조금 더 발전이 되었습니다. 김포 공항에서 김해 공항까지 가는데 중간에 착륙하지 않아야 합니다. 이 상황을 그림으로 도식화 시키면 아래 그림이 나옵니다.



[그림 **4]** 문제의 조건을 만족하는 비행을 도식화 시킨 그림

잘 보면, 수평 방향으로 2만큼이 빠지고, 높이가 1만큼 올라갔음을 알 수 있습니다. y축으로 -1 만큼 평행이동 시켜 봅시다. 사실, 양 옆 한 칸은 의미가 없기 때문입니다. 이미 상승, 하강해야 한다는 것이 정해졌기 때문입니다.



[그림 **5] y**축으로 **-1**만큼 평행 이동 후

그림을 보면, d-2만큼 이동할 동안 고도가 0 밑으로 떨어지지 않는 경우의 수와 같습니다. 다른 방법은 아래 상태 정의를 이용합니다.

dp[w][h] : 수평 거리 w만큼 이동했을 때, 고도가 h인 경우의 수

이 때, dp[w][h] = dp[w-1][h-1] + dp[w-1][h+1]임을 알 수 있습니다. 단 dp[0][0] = 1입니다. 이걸 보면, 별로 효율이 없는 게 아닌가? 라고 생각할 수도 있습니다. 그런데, 이 dp 식은 벡터 연산을 할 수 있게 해 줍니다.

dp[w-1][0]	dp[w-1][1]	...	...	dp[w-1][h-2]	dp[w-1][h-1]
dp[w-1][2]	dp[w-1][3]	...	...	dp[w-1][h]	dp[w-1][h+1]
dp[w][0]	dp[w][1]	...	...	dp[w][h-1]	dp[w][h]

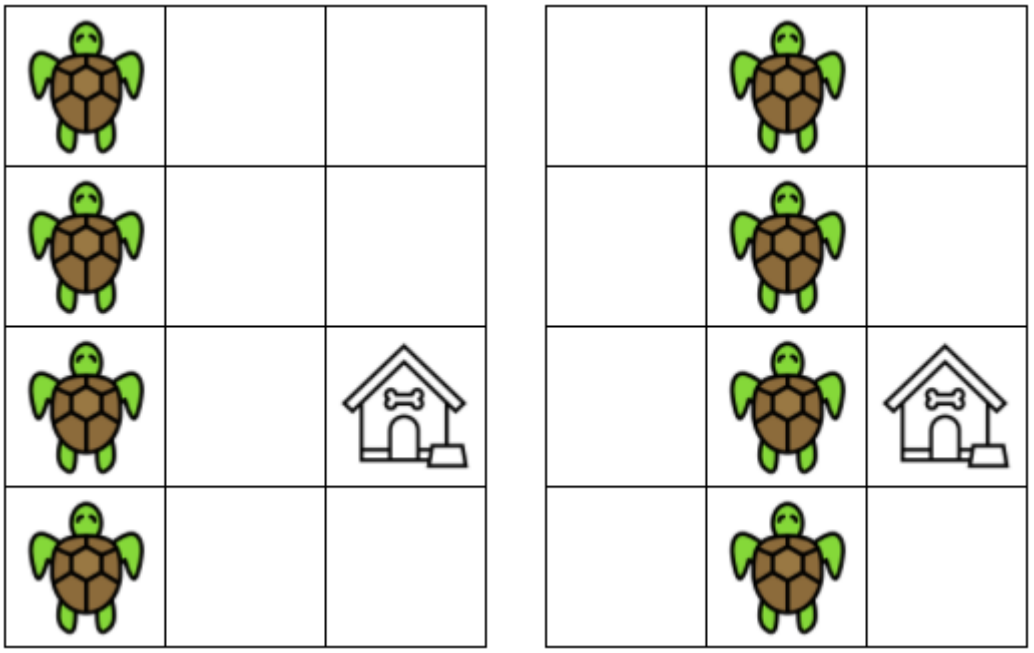
회색 부분이 같은 값인 걸 알 수 있어요. 이 둘을 sum 하면 dp[w]가 나옵니다. 위에 1행을 L벡터, 2행을 R벡터라고 하면, L벡터에 R벡터를 더한 것이 그 다음 dp[w+1]의 상태임을 알 수 있어요. 이것을 구하면 dp[w][2 ... h+2]까지의 상태 또한 구할 수 있습니다. indexing을 하는 대신, 벡터 화를 시켜서 빠르게 처리할 수도 있습니다. numpy를 쓰는 목적 중 하나가 될 수 있겠습니다.

필요한 스킬
다이나믹 프로그래밍, (python 한정) numpy

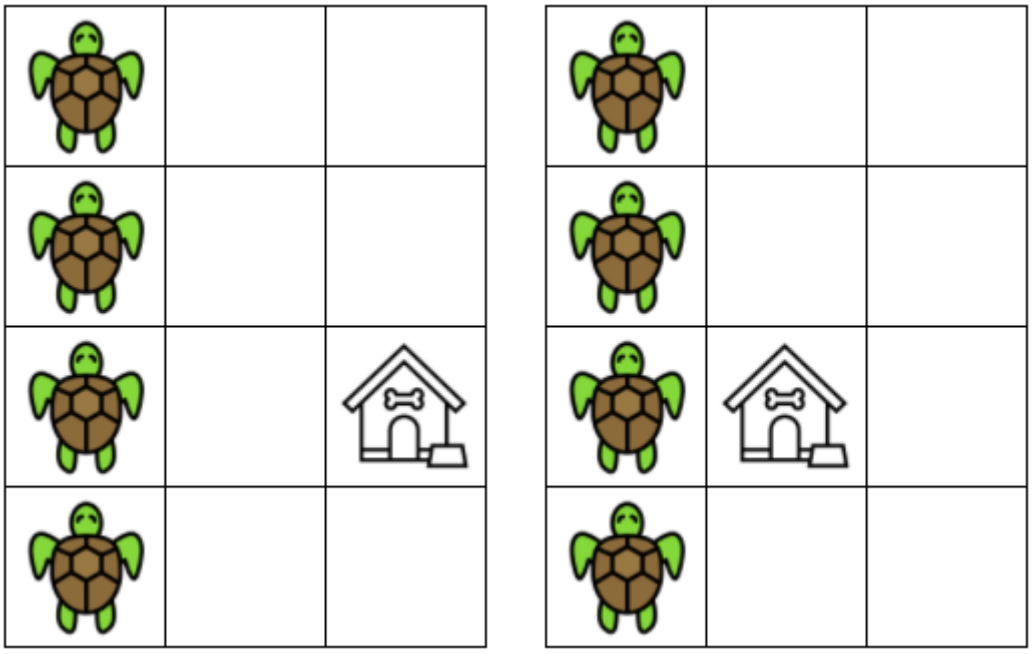
7.
출제 의도
상대 속도의 개념을 이해할 수 있는가?
충돌 처리를 이해할 수 있는가?

의도한 난이도
골드 1 ~ 플레 5+

풀이
먼저, 그림 6과 그림 7의 상황이 같다는 것을 직관적으로 이해하셔야 합니다.



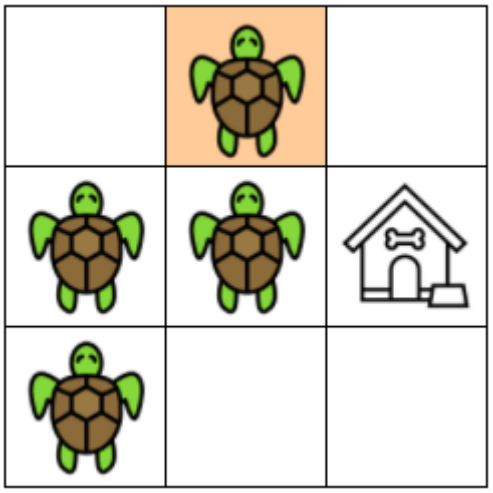
[그림 **6]** 거북이 컴포넌트가 우측으로 한 칸 이동



[그림 **7]** 거북이 컴포넌트 말고 집이 좌측으로 한 칸 이동

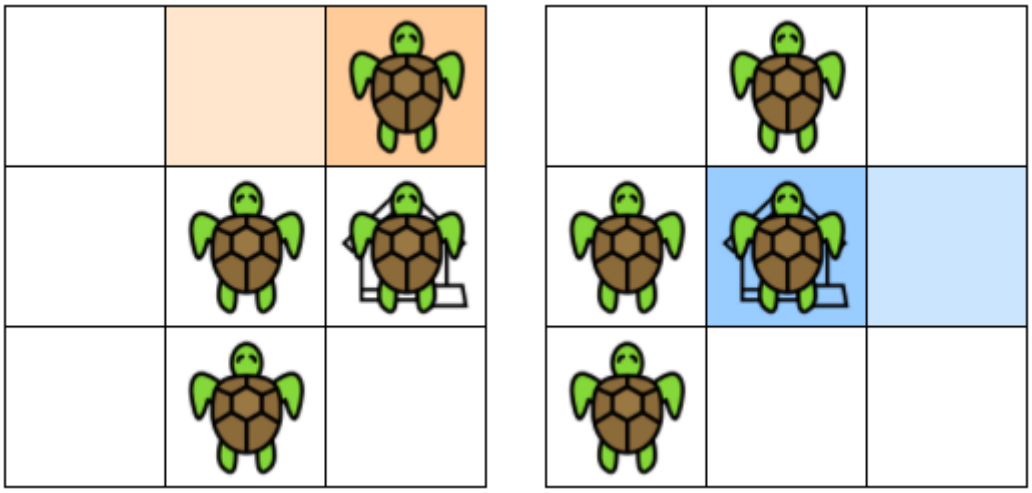
이유는 간단합니다. 거북이 집 입장에서 보았을 때, 그림 6도 그림 7도 서쪽으로 1칸만 가면 위에서 3번째 위치에 도달하기 때문입니다. 즉, 우리는 거북이 컴포넌트가 (dx, dy)만큼 이동하고 장애물이나 집이 가만히 있었다면, 거북이 컴포넌트들이 가만히 있고, 장애물이나, 집이 (-dx, -dy)만큼 이동한 것과 같습니다.

상식적으로 생각해 보면 간단합니다. 1차원 좌표계에서 제가 위치 0에 있고, 가희가 위치 10에 있습니다. 제가 1만큼 동쪽으로 이동했습니다. 그러면 가희 입장에서는 서쪽으로 9만큼 이동 하면, 저와 만날 겁니다. 반대로, 제가 위치 0에 있고 가희가 위치 10에 있었다고 합시다. 가희가 서쪽으로 1칸 이동했습니다. 그 후에, 가희가 서쪽으로 9칸 이동해도 저와 만납니다. 가희가 서쪽으로 1칸 이동한 상황과, 제가 동쪽으로 1칸 이동한 상황과 같음을 알 수 있어요. 상대적인 위치 때문입니다. 이제 거북이 컴포넌트를 어떻게 처리할 지 생각해 봅시다.



[그림 **8]** 거북 컴포넌트의 기준 위치

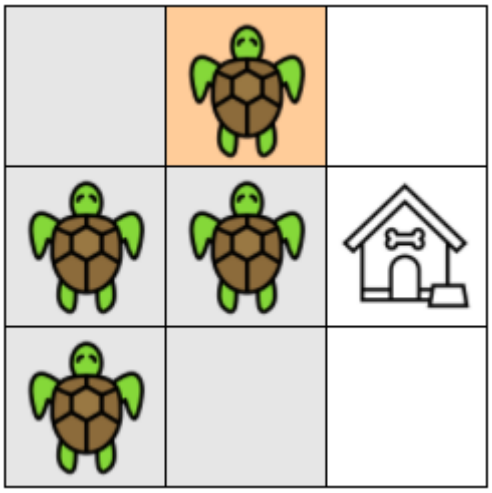
거북 컴포넌트의 위쪽 맨 좌측 위치를 기준점으로 잡습니다. 다음에, 주황색으로 칠한 것이, (x, y)에 있다면 집에 있는지, 장애물에 있는지, 맵 바깥에 있는지 판단합니다. 집에 있는 경우, 장애물에 있는 경우, 맵 바깥에 있는 순서 대로 맵을 채워나가면 됩니다. 집이던 장애물이던, 알고리즘은 똑같습니다.



[그림 **9]** 거북이 기준점이 **1**행 **1**열에서 **1**행 **2**열로 이동한 경우
집이 2행 3열에서 2행 2열로 이동한 것과 같다.

거북이의 기준점이 기준점으로부터 (dx, dy)만큼 변했다면, 집과 장애물들의 좌표가 (-dx, -dy) 만큼 변한 것과 같습니다. 이 좌표에 T가 있는지 판단해 주면 됩니다. 만약에 장애물이 있다면, 해당 위치에 '#'을 표시해 주면 됩니다. 단, 범위에 벗어난 경우는 처리하지 않고 무시하시면 됩니다.

그런데, 이 처리까지만 하면 끝날까요? 당연하게도 아닙니다.



[그림 10] bounding box

bounding box 처리를 해야 합니다. bounding box는 모든 점이 놓이는 가장 작은 넓이를 갖는 직사각형을 의미합니다. 위 예에서는 1행 1열, 1행 2열, 2행 1열, 2행 2열, 3행 1열, 3행 2열. 이렇게 2 by 3 직사각형이 bounding box가 됩니다. 거북이 기준점이 (gx, gy)일 때 bounding box의 범위가 sx ~ ex, sy ~ ey였다면, 기준점이 (x, y)라면 sx + (x-gx) ~ ex + (x-gx), sy + (y-gy) ~ ey + (y-gy)가 될 겁니다. 이 범위가 valid한 좌표 범위에 속하는지 검사하시면 됩니다. 실수하지 않는 방법입니다.

Bounding Box 처리까지 처리가 끝났다면, 이제 흔한 (쉬운) bfs 문제로 바뀝니다. 역추적은 어디로부터 왔는가를 wif 배열에 저장하면 됩니다. U 방향이냐, R 방향이냐, D 방향이냐, L 방향이냐는 좌표 변화량을 가지고 switch 문을 돌리면 됩니다.

필요한 스킬
bfs, 그래프 이론, 기본 물리학, 그래픽 이론

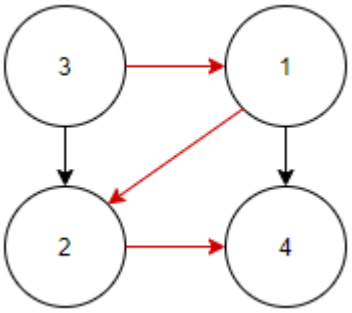
8.
출제 의도
상황 도식화를 시킬 수 있는가?
위상 정렬에 대해서 이해하고 있는가?

의도한 난이도
골드 1 ~ 플레 5+

풀이
먼저 게임의 결과가 아래와 같이 나왔다고 해 보겠습니다.

1211 1231 1215 1217

이 말은 1211 다음에 1231이 오고, 다음에 1215가 오고 1217이 온다는 의미입니다. 이것을 순서 1이라고 하겠습니다. 그 다음에, 각 사람마다 카드를 내는 순서가 있습니다. 예를 들어, 1이 1211이 적혀있는 3번, 1215가 적혀있는 2번을 순서대로 낸다는 정보가 주어지고, 2가 1231이 적혀있는 1번, 1217이 적혀있는 4번을 순서대로 낸다는 정보가 주어졌다고 해 봅시다. 이러한 정보들을 순서 2라고 합시다.



[그림 **11]** 순서 **1**과 순서 **2**로 구축된 그래프

순서 1과 순서 2를 가지고 그래프를 구축한 뒤에, 위상 정렬을 시도해 보면 됩니다. 만약에 모두 방문할 수 있으면 방문한 순서가 답이 됩니다. 그렇지 않으면 사이클이 있으므로 불가능합니다. 문제는 순서 1과 순서 2가 주어져 있을 때, 답이 되는 경우는 한 가지만 나오는가 입니다.

결과 리스트의 길이가 C이고 카드가 C개 주어졌고, valid한 결과라면, 1가지만 나오게 됩니다. 이미 C개에 대한 선후 관계가 순서 1에서 주어졌기 때문입니다. 문제는 리스트에 두 번 이상 나온 수가 존재하는 경우입니다. 이 경우, 많아야 7개가 주어지기 때문에, 두 번 이상 나온 것만 모아두고 permutation을 돌리면 됩니다. 많아봐야 7개이기 때문에 7!에 위상 정렬 복잡도인 2만을 곱하면 1억이 약간 안 되게 됩니다. permutation을 구축하는 방법은 가희와 읽기 쓰기 놀이 1 문제를 참고하시면 됩니다.

그런데 이 문제, 사실 위상 정렬을 안 쓰고 단순 dfs만 이용해서 풀어도 됩니다. 어떻게 풀어야 하는지는 과제로 남겨두겠습니다.

필요한 스킬

위상 정렬, permutation, brute force