알튜비튜

구현 & 코너케이스

최근 대다수의 코딩테스트에서 구현 능력을 보기 위한 문제들이 출제되고 있습니다. 또한 대다수의 기업에서 코딩테스트 시 히든 케이스를 제공하지 않습니다.
따라서 이런 문제를 만났을 때 어떻게 접근하면 좋을지, 어떤 코너 케이스가 있을지 함께 생각해보면서 풀어보는 시간을 가져보겠습니다.

지금까지 배운 것들…

#알튜비튜

이분 탐색

다익스트라

벨만-포드

정렬

덱

스택

브루트 포스

플로이드-워셜

set

투 포인터

우선순위 큐

동적 계획법

DFS

map

큐

백트래킹

BFS

유클리드 호제법

그리디

구현?

이분 탐색
다익스트라
정렬
덱
스택
벨만-포드
플로이드-워셜
set
우선순위 큐
투 포인터
→ 구현!
동적 계획법
DFS
브루트 포스
map
큐
백트래킹
BFS
유클리드 호제법
그리디

코딩 테스트에선?

삼성 공채 / SW 역량 테스트 A형

- 3시간 동안 2문항 출제. 2문항 모두 구현 문제
- 히든 테스트 케이스 존재. 주어진 테스트 케이스 맞췄다고 합격인 것이 아님!

카카오 블라인드 채용

- 구현 문제가 아니더라도 전체적으로 기본적인 문자열 처리 및 설계 능력이 필요함.
- 구현 문제의 난이도가 높은 편. (정답률 5~10% 미만!)

네이버, 라인, 쿠팡…

- 짧은 제한 시간 내에 문제를 해결해야 함.
- 제공되는 테스트 케이스 1개. 이외 모든 케이스는 전부 히든 케이스!

어떻게 풀지?

#알튜비튜

문제 분석

- 문제에서 요구하는 바를 읽고, 어떤 절차를 통해 문제를 풀어야 하는지 설계.
- 구현 문제는 정보량이 많은 문제가 상당하니 주의 깊게 읽고 설계해야 함.
- 주어진 예제가 내가 설계한 로직에 따라 잘 나오는지 확인하는 과정도 필수!

코드 구조 설계

- 설계한 풀이를 실제로 코드로 옮기기 위해 필요한 구현 방식, 자료구조 등을 설계함.
- 실수를 줄이기 위해, 함수화를 많이 하는 게 좋음!

모듈별 구현 및 테스트

- 설계 내용을 바탕으로 모듈마다 구현
- 모듈 별로 원하는 답 나오는지 출력하면서 코딩하는 것을 추천!

지난 과제 문제를 먼저 봅시다!

#알튜비튜

- 20055번 : 컨베이어 벨트 위의 로봇 – Gold 5
- 7662번 : 이중 우선순위 큐 – Gold 5
- 17281번 : ⚾ – Gold 4
- 16234번 : 인구 이동 – Gold 5
- 11559번 : Puyo Puyo – Gold 4

지난 과제 문제를 먼저 봅시다!

#알튜비튜

- 20055번 : 컨베이어 벨트 위의 로봇 – Gold 5 → 자료구조, 구현
- 7662번 : 이중 우선순위 큐 – Gold 5 → 우선순위 큐, 구현, 방문 체크 관리!
- 17281번 :  – Gold 4 → 백트래킹, 순열, 구현
- 16234번 : 인구 이동 – Gold 5 → BFS/DFS, 구현
- 11559번 : Puyo Puyo – Gold 4 → BFS/DFS, 구현

20055번 : 컨베이어 벨트 위의 로봇 - Gold 5

#알튜비튜

문제 분석

벨트가 한 칸 회전하면 1번부터 2N-1번까지의 칸은 다음 번호의 칸이 있는 위치로 이동하고, 2N번 칸은 1번 칸의 위치로 이동한다. i번 칸의 내구도는 Ai이다. 위의 그림에서 1번 칸이 있는 위치를 "올리는 위치", N번 칸이 있는 위치를 "내리는 위치"라고 한다.

→ 오른쪽으로 회전
→ 올리는 위치, 내리는 위치 체크
→ 올리는 위치와 내리는 위치를 바꾸는걸로 회전을 구현할 수 있지 않을까?

컨베이어 벨트에 박스 모양 로봇을 하나씩 올리려고 한다. 로봇은 올리는 위치에만 올릴 수 있다. 언제든지 로봇이 내리는 위치에 도달하면 그 즉시 내린다. 로봇은 컨베이어 벨트 위에서 스스로 이동할 수 있다.
로봇을 올리는 위치에 올리거나 로봇이 어떤 칸으로 이동하면 그 칸의 내구도는 즉시 1만큼 감소한다.

→ 로봇을 관리하는 컨테이너 필요
→ 내구도를 관리하는 컨테이너 필요

20055번 : 컨베이어 벨트 위의 로봇 - Gold 5

#알튜비튜

문제 분석

컨베이어 벨트를 이용해 로봇들을 건너편으로 옮기려고 한다. 로봇을 옮기는 과정에서는 아래와 같은 일이 순서대로 일어난다.
(생략)
종료되었을 때 몇 번째 단계가 진행 중이었는지 구해보자. 가장 처음 수행되는 단계는 1번째 단계이다.

→ 로봇을 옮기는 과정을 그대로 구현하면 되는구나!

로봇 옮기는 과정

1. 벨트가 각 칸 위의 로봇과 함께 한 칸 회전
2. 가장 먼저 벨트에 올라간 로봇부터, 벨트 회전 방향으로 한 칸 이동할 수 있다면 이동
   (이동가능: 이동하려는 칸에 로봇이 없고, 그 칸의 내구도가 1 이상이어야 함)
3. 올리는 위치에 있는 칸의 내구도가 0이 아니면 올리는 위치에 로봇 올림
4. 내구도가 0인 칸의 개수가 k개 이상이라면 과정 종료. 그렇지 않다면 1로 돌아감

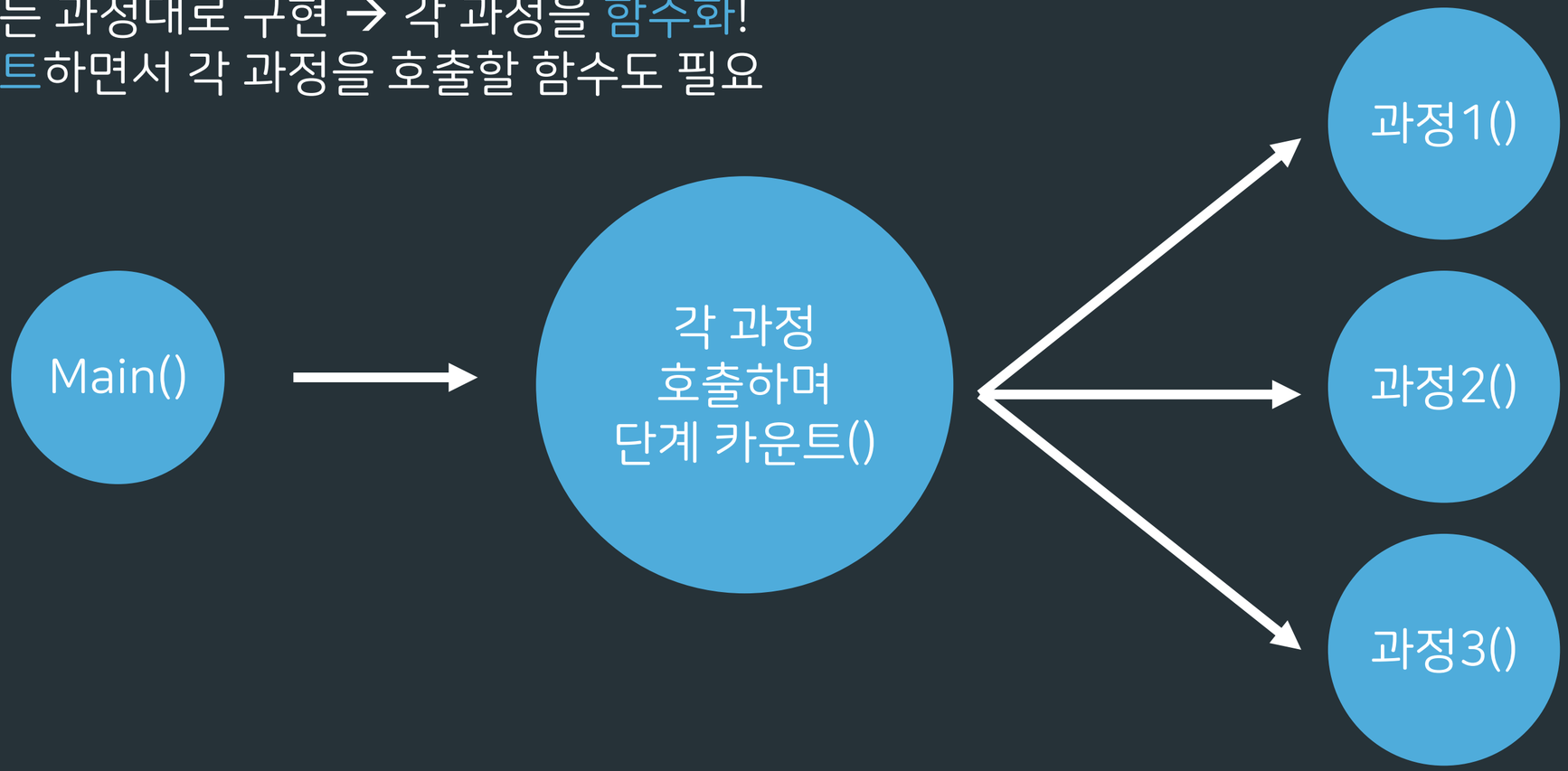
→ 1 ~ 3까지가 1단계

20055번 : 컨베이어 벨트 위의 로봇 - Gold 5

#알튜비튜

코드 구조 및 설계

- 회전은 올리는 위치와 내리는 위치를 한 칸 옮겨서 구현
- 로봇 옮기는 과정대로 구현 → 각 과정을 함수화!
- 단계 카운트하면서 각 과정을 호출할 함수도 필요



20055번 : 컨베이어 벨트 위의 로봇 - Gold 5

#알튜비튜

코드 구조 및 설계

```
//벨트 회전
void first(){
  올리는 위치와 내리는 위치 한 칸 감소
}

//모든 로봇 옮기기
void second(){
  로봇 오른쪽으로 한 칸 움직일 수 있는지 확인 후 옮기기
}

//로봇 올리기
void third(){
  올리는 위치에 로봇 올릴 수 있는지 확인 후 올리기
}
```

```
int simulation(){
  while(){
    if(종료조건){
      break;
    }
    first();
    second();
    third();
    cnt++;
  }
  return cnt;
}

int main(){
  cout << simulation();
}
```

#알튜비튜

3190번 : 뱀 – Gold 5

문제

- NxN 정사각형 보드 위에서 게임이 진행. 뱀이 기어다님
- 몇몇 칸에 사과가 놓여있는데, 뱀은 사과를 먹으면 길이가 늘어남
- 뱀이 벽 또는 자기자신의 몸과 부딪히면 게임이 끝남
- 게임 시작할 때 뱀은 맨위 맨좌측 (1행 1열) 에 위치하고 길이는 1. 방향은 오른쪽
- 뱀은 매 초마다 아래 이동 규칙을 따름
  - 먼저 몸 길이를 늘려 머리를 다음 칸에 위치시킴
  - 이동한 칸에 사과가 있다면, 사과 없어지고 꼬리 그대로
  - 이동한 칸에 사과가 없다면, 꼬리가 위치한 칸 비워줌. 즉, 몸 길이 변하지 않음
- 게임이 몇 초 후에 끝나는지 구하여라

제한 사항

- 보드의 크기 N의 범위는 2 <= N <= 100
- 사과의 개수 K의 범위는 0 <= K <= 100
- 방향 변환 횟수 L의 범위는 1 <= L <= 100
- 방향 변환 정보 X의 범위는 1 <= X <= 10,000

예제

#알튜비튜

예제 입력 1

```
6
3
3 4
2 5
5 3
3
3 D
15 L
17 D
```

예제 출력 1

```
9
```

예제 입력 2

```
10
4
1 2
1 3
1 4
1 5
4
8 D
10 D
11 D
13 L
```

예제 출력 2

```
21
```

예제 입력 3

```
10
5
1 5
1 3
1 2
1 6
1 7
4
8 D
10 D
11 D
13 L
```

예제 출력 3

```
13
```

3190번 : 뱀 - Gold 5

#알튜비튜

문제 분석

- NxN 정사각형 보드 위에서 게임이 진행. 뱀이 기어다님
→ 우선 보드를 나타낼 배열 필요. 뱀은 어떻게 표시할까?

- 몇몇 칸에 사과가 놓여있는데, 뱀은 사과를 먹으면 길이가 늘어남 → 사과도 보드에 같이 표시하자

- 뱀이 벽 또는 자기자신의 몸과 부딪히면 게임이 끝남 → 탐색 종료 조건

- 게임 시작할 때 뱀은 맨위 맨좌측(1행 1열)에 위치하고 길이는 1. 방향은 오른쪽 → 초기화 값

- 뱀은 매 초마다 아래 이동 규칙을 따름
  - 먼저 몸 길이를 늘려 머리를 다음 칸에 위치시킴
  → 머리와 꼬리는 어떻게 구분하지? 다음 탐색을 위해 현재 머리 위치를 항상 저장해두어야 하네
  - 이동한 칸에 사과가 있다면, 사과 없어지고 꼬리 그대로
  - 이동한 칸에 사과가 없다면, 꼬리가 위치한 칸 비워줌. 즉, 몸 길이 변하지 않음

3190번 : 뱀 - Gold 5

#알튜비튜

코드 구조 및 설계

- N의 범위가 100 이하이므로 2차원 배열 사용해서 보드 표현! * c++에선 범위가 10,000을 넘으면 2차원 배열 사용 시 메모리 초과
→ 이때, 구현 편하게 하기 위해 0행 0열부터 시작하는 것으로 하자
- 보드에 뱀과 사과를 표시 (뱀: 1, 사과: 2) * 둘의 값이 다르기만 하면 아무거나 상관 x

- 뱀의 현재 상태를 따로 관리해야 몸을 늘리거나 방향을 바꾸는 연산 원활하게 처리 가능
→ 뱀의 현재 상태를 자료구조에 저장해서 관리!
→ 머리와 꼬리 두 방향에 쉽게 접근하기 위해 양방향 컨테이너인 덱(deque) 사용

- 뱀 매 초마다 이동 → 반복문으로 관리
- 방향 전환을 쉽게 관리하기 위해 방향 배열을 사용해서 구현 (bfs 풀이에서 많이 볼 수 있음)

- 종료 조건 확인! → 뱀 매 초마다 이동하는 반복문 내에서 확인

→ 매 초마다 뱀 이동하는 연산 함수화해서 구현!

3190번 : 뱀 - Gold 5

#알튜비튜

코드 구조 및 설계

```
//매 초마다 뱀 이동시키는 함수. 종료 조건에 걸리면 걸린 시간 리턴
int playGame(){
  뱀 상태 초기화
  while(){
    시간 증가
    이동할 위치 파악
    if(종료 조건){
      break;
    }
    사과 확인
    머리 이동
    방향 변환 확인
  }
  return 시간;
}
```

```
int main(){
  입력 받으면서 보드에 사과 표시
  0행 0열부터 시작하는 것으로 구현하므로 입력 좌표에서 1을 빼줘야 함
  회전 정보 입력
  cout << playGame();
}
```

예제 풀이

#알튜비튜

front: 머리

(0, 1)	(0, 0)	

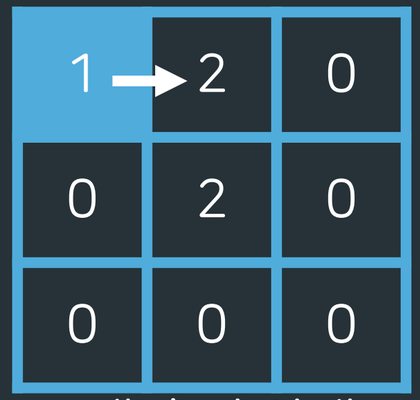
뱀이 위치한 (행, 열)을 저장

입력

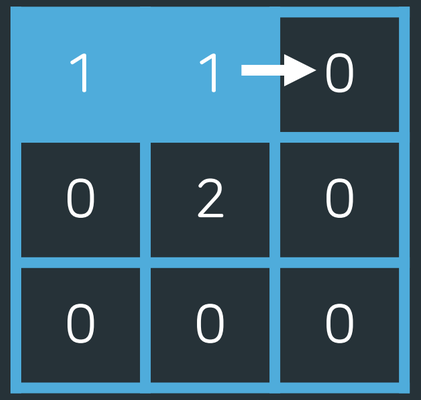
```
3
2
1 2
2 2
2
2 D
4 L
```

출력

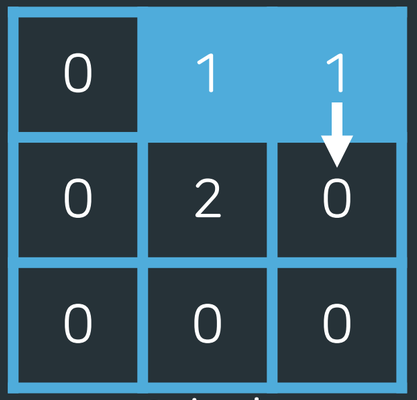
```
5
```



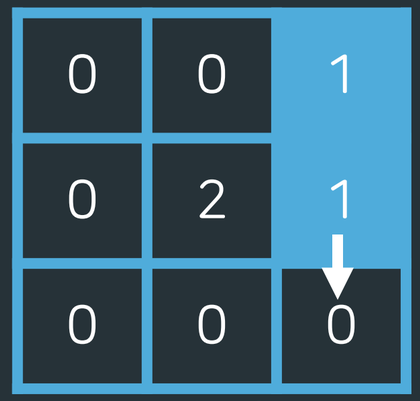
뱀 초기 상태



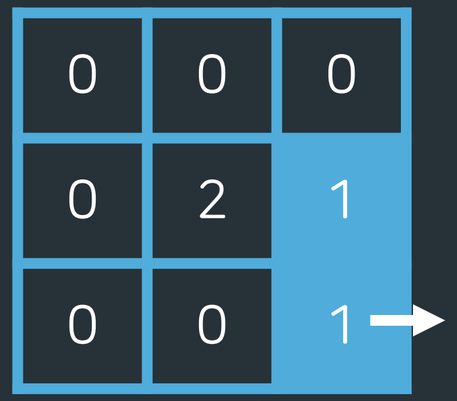
1초 후



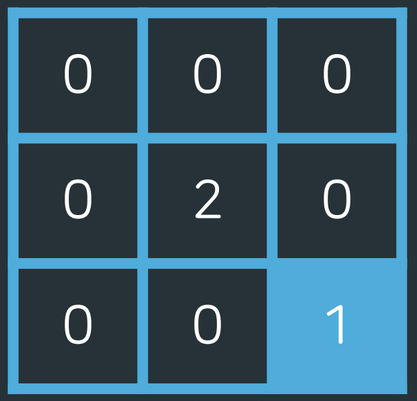
2초 후



3초 후



4초 후



5초 후

충돌 → 게임 종료

코너 케이스란?

#알튜비튜

코너 케이스

- 문제의 예제는 다 맞았는데, 제출하면 틀리는 경우
- 어렵지 않은데 정답률이 낮은 문제 (보통 20-30% 이하!!)
- 알고리즘 유형과 상관 없이 설계한 로직에 대해 코너 케이스가 있는지 고려해야 함

코너 케이스 찾기

#알튜비튜

코너 케이스 찾는 방법

- 입력 범위가 최솟값, 최댓값일 때도 커버를 제대로 하고 있는지 확인
- 자료형을 올바르게 사용하였는지 확인. 특히 int 범위를 넘어가지는 않는지 확인!
- 문제에 빠뜨린 조건이 있는지 확인
- 입력이 간단할 경우 문제의 조건에 따라 나올 수 있는 여러 상황 생각해서 테스트 케이스 만들어보기
- 코너 케이스는 특히 문제를 직접 풀어야 찾는 요령을 터득할 수 있기에 여러 문제를 풀어보는 것이 중요

지난 과제 문제를 먼저 봅시다!

#알튜비튜

- 1205번 : 등수 구하기 – Silver 4
- 5397번 : 키로거 – Silver 3
- 2840번 : 행운의 바퀴 – Silver 4
- 13422번 : 도둑 – Gold 4

1205번 : 등수 구하기 - Silver 4

#알튜비튜

문제

- 랭킹 리스트는 비오름차순으로 저장
- 같은 점수가 있을 때는 가장 작은 등수가 됨
- 예를 들어 랭킹 리스트가 100 90 90 80일 때 각각의 등수는 1 2 2 4
- 새로운 점수가 들어올 때, 기존의 랭킹 리스트에서 몇 등을 하는지 구하여라

제한 사항

- 랭킹 리스트 크기 P의 범위는 10 <= P <= 50
- 리스트에 있는 점수 개수 N의 범위는 0 <= N <= P
- 모든 점수의 범위는 0 <= 점수 <= 2,000,000,000

코너 케이스

- 리스트에 있는 점수 개수가 0일 경우 고려 (입력값이 최소인 상황)
- 동점인 경우 처리 → 구현 방식에 따라 코너 케이스가 될 수도 있고 안될 수도 있음

5397번 : 키로거 - Silver 3

#알튜비튜

문제

- 키로거는 사용자가 키보드를 누른 명령을 모두 기록
- 비밀번호 창에서 입력한 키가 주어졌을 때, 입력한 비밀번호를 구하는 문제
- 키보드로 입력한 키는 알파벳 대문자, 소문자, 숫자, 백스페이스(-), 화살표(<, >)

제한 사항

- 입력한 키의 길이 L의 범위는 1 <= L <= 1,000,000

코너 케이스

- 그냥 배열로 구현할 경우, 글자 삽입 삭제 시에 생길 수 있는 런타임 에러를 주의해야 하는 문제!!
  (존재하지 않는 인덱스에 참조할 가능성 있음)
- 해결하기 위해 덱(deque) 자료구조 사용

2840번 : 행운의 바퀴 - Silver 4

#알튜비튜

문제

- 행운의 바퀴에 알파벳 대문자가 적혀 있음. 바퀴 옆에 화살표 존재. 항상 한 곳 가리킴
- 바퀴에 같은 글자는 두 번 이상 등장하지 않음
- 바퀴는 시계방향으로 돌아감
- 매번 바퀴를 돌릴 때 마다, 화살표가 가리키는 글자가 변하는 횟수와 어떤 글자에서 회전을 멈추었는지에 대한 정보가 주어질 때, 바퀴에 적힌 알파벳 알아내는 문제

제한 사항

- 바퀴 칸의 수 N의 범위는 2 <= N <= 25
- 바퀴를 돌리는 횟수 K의 범위는 1 <= K <= 100

코너 케이스

- 바퀴에 쓰인 알파벳은 중복되지 않는다는 조건 빠뜨리지 않도록 주의
- 회전 방향 주의

13422번 : 도둑 - Gold 4

#알튜비튜

문제

- 도둑은 M개의 연속된 집에서 돈을 훔치되, 각 집에 보관중인 돈을 전부 훔침
- K원 이상의 돈을 훔친다면 자동 방범장치가 작동하여 도둑은 바로 붙잡힘
- 마을을 이루고 있는 집의 개수 N, 도둑이 돈을 훔쳐야 할 연속된 집의 개수 M, 잡히지 않을 최소 돈의 양 K와 각 집에서 보관 중인 돈이 순서대로 주어질 때, 도둑이 들키지 않고 무사히 돈을 훔칠 수 있는 경우의 수 구하는 문제

제한 사항

- N의 범위는 1 <= N <= 100,000
- M의 범위는 1 <= M <= N
- K의 범위는 1 <= K <= 1,000,000,000

코너 케이스

- 마을에 있는 집의 개수와 도둑이 훔칠 집의 개수가 같을 때 (n==m) !

2011번 : 암호코드 - Gold 5

문제

- A를 1, B는 2, … Z는 26 으로 암호화
- 이렇게 하면 암호화한 숫자를 다시 글자로 바꾸는 방법이 여러 가지가 나옴
- 예를 들어 25114를 다시 영어로 바꾸면, “BEAAD”, “YAAD”, “YAN”, “YKD”, “BEKD”, “BEAN” 총 6 가지가 나옴
- 어떤 암호가 주어졌을 때, 나올 수 있는 해석의 가지 수를 구해보자

제한 사항

- 암호 길이의 범위는 1 <= 암호 길이 <= 5000
- 정답이 매우 클 수 있으므로 1000000으로 나눈 나머지를 출력

→ 해당 수는 상수로 선언해서 사용 (실수하기 쉬우므로)

- 암호가 잘못되어 해석할 수 없는 경우에는 0 출력

예제

#알튜비튜

예제 입력 1

```
25114
```

예제 입력 2

```
1111111111
```

예제 출력 1

```
6
```

예제 출력 2

```
89
```

접근법

브루트 포스 접근

- 현재 자리에서 1, 2개 선택해서 우선 수 분리 후 암호 해석 되는지 확인

→ 약 2^5000 만큼 경우의 수를 모두 고려해야 하므로 무조건 시간초과!

→ 그렇다면 이전에 구한 경우를 활용할 수 없을까?

DP 접근

- 자릿수를 돌면서 현재 자릿수까지 만들 수 있는 암호 해석 개수 저장
- 암호는 최대 두 글자를 한 자리의 알파벳으로 해석
- 따라서 현재 자릿수에서 암호 해석 시, 한 글자나 두 글자 전의 암호에서 이어서 해석 가능!

점화식 구하기

#알튜비튜

점화식

- DP[i] = 현재 자릿수까지의 암호 해석 가지수

- 현재 자릿수 하나를 암호로 해석할 경우 [i-1]에서 이어서 해석한 경우고, 현재 자릿수와 그 전 자릿수를 포함해 두 글자를 암호로 해석할 경우 [i-2]에서 이어서 해석한 경우

→ DP[i]는 위의 두 가지 경우 중 암호가 되는 경우의 수를 더해주면 됨!

→ DP[i] = DP[i-1] + DP[i-2] (이때, 각각 암호 해석 가능한 경우만 더하기!)

Index		1	2	3	4	5
dp	암호	2	5	1	1	4
	암호 해석 경우의 수	1	2	2	4	6

* 인덱스 관리 편하게 + 중복 코드 줄이기 위해 1번 인덱스부터 시작!
(최소 현재 인덱스 – 2까지 접근해야 하기 때문)

반례

#알튜비튜

입력 1	입력 2	입력 3	입력 4
10	01	1230	27
오답 1	**오답 2**	**오답 3**	**오답 4**
2	1	3	2
정답 1	**정답 2**	**정답 3**	**정답 4**
1	0	0	1

코너 케이스

- ‘0’ 혼자는 암호 해석 불가

→ 단순히 한 글자는 모두 가능하다고 처리하기 쉬운데, ‘0’은 한 글자로 암호 해석 불가함을 주의!

- 처음 시작이 ‘0’인 경우 주의

→ 처음 시작이 0으로 주어질 경우를 놓치기 쉬우니 주의!

- 현재 수가 ‘0’인데 앞의 수가 ‘1’이나 ‘2’가 아닌 경우 → 암호 해석 불가

→ 0 혼자는 암호 해석이 불가하므로 꼭 두 글자 해석이 이루어져야 하는데 26 이하의 수가 아니므로

- 두 개의 수를 하나의 알파벳으로 고려할 때, 26이하의 수인지 잘 확인!!

→ 그 전 자릿수(십의 자리수)가 1인지 2인지만 고려할 경우 하기 쉬운 실수, 26이하인지 잘 확인하자!

마무리

#알튜비튜

정리

- 구현 & 코너케이스 문제는 많은 문제를 풀어보는 것이 가장 좋음
- 구현 문제는 자료구조를 활용하는 경우 많음. BFS / DFS 탐색도 많이 나옴
- 문제를 분석하고, 코드 구조를 설계하고, 예제가 설계한 구조대로 잘 나오는지 확인하자!
- 함수화를 잘 하자! → 논리가 복잡해질수록 실수를 줄일 수 있음
- 가장 기본적인 반례는 입력 범위의 최솟값, 최댓값으로 주어지는 경우
- 연산 범위를 확인하여 알맞은 자료형 설정
- 조건문이 모든 경우를 커버하는지 확인
- 문제 풀이 시, 바로 코딩하는 것보다 직접 손으로 예제를 수행해보며 문제 이해하고, 구조 설계하는 것 추천

과제

필수

- 16236번 : 아기 상어 – Gold 3
- 2615번 : 오목 – Silver 2

3문제 이상 선택

- 14500번 : 테트로미노 – Gold 5
- 14503번 : 로봇 청소기 – Gold 5
- 16235번 : 나무 재테크 – Gold 4
- 3613번 : Java vs C++ – Silver 3
- 2607번 : 비슷한 단어 – Silver 4

과제 마감일

#알튜비튜

코드리뷰 O 마감 ~ 5월 16일 월요일 낮 12시

코드리뷰 X 마감 ~ 5월 16일 월요일 밤 12시 (16일에서 17일로 넘어가는 자정)

추가제출 마감 ~ 5월 17일 화요일 밤 12시 (17일에서 18일로 넘어가는 자정)