**이니로 스터디 2주차 보고서**

**팀명: I am Brute**



|  |  |  |  |
| --- | --- | --- | --- |
| **회차** | 2회차 | **날짜** | 2021/03/29(월) |
| **스터디 주제** | Chapter3, 4 코드 리뷰 및 발표 | | |
| **스터디 장소/시간** | 학술 정보관 / 18:00~19:00 | | |
| **스터디 내용** | 참여인원( 팀원 전원 )  **Greedy Algorithm**   * 현재 상황에서 가장 좋은 것만 선택하는 방법   ex) 거스름돈   * 거슬러줘야하는 금액이 n원일 때, 가장 적은 동전으로 거슬러 주고 싶다. 이때 동전의 갯수는?   In [4]:  n = 1260  coins = [500, 100, 50, 10]  cnt = 0  **for** coin **in** coins:  cnt += n // coin  n %= coin  print(cnt)  **greedy algorithm의 정당성**  가지고 있는 동전 중에서 큰 단위가 항상 작은 단위의 배수이므로 작은 단위의 동전들을 종합해 다른 해가 나올 수 없다.   * 만약 동전이 500, 400, 100원 이었다면 얘기가 달라진다. greedy로 푼다면 800원을 거슬러 준다면 500, 100, 100, 100 이 되지만 실제 답은 400, 400 이기 때문이다.   **큰 수의 법칙**   * integers 배열이 주어지고, M, K가 주어진다. 해당 배열의 값들을 M번 더하는데 각 숫자마다 **연속하여** K번을 넘게 더할 순 없다. 같은 숫자라도 인덱스가 다르다면 다른 숫자로 생각한다.   N, M, K = map(int, input().split())  nums = list(map(int, input().split()))  nums.sort(reverse=**True**)  to\_add = [nums[0], nums[0], nums[0], nums[1]]  cnt = 0  **for** i **in** range(M):  cnt += to\_add[i % (K + 1)]  print(cnt)  **숫자 카드 게임**   * 각 row의 가장 작은 숫자 카드 중 가장 큰 카드를 뽑는 게임 * input: N, M cards   N, M = map(int, input().split())  cards = [list(map(int, input().split())) **for** \_ **in** range(N)]  cards.sort(key=**lambda** x: min(x), reverse=**True**)  print(min(cards[0]))  **1이 될 때까지**  **import** **math**  N, K = map(int, input().split())  cnt = 0  **while** N > 1:  N //= K  cnt += 1 + N % K  **if** N == 0:  cnt -= 1  print(cnt)  **Chapter 4**  **상하좌우**  input: N directions  N = int(input())  dirs = list(input().split())  **class** **Position**:  **def** \_\_init\_\_(self, x, y):  self.x = x  self.y = y  **def** \_\_add\_\_(self, other):  **return** Position(self.x + other.x, self.y + other.y)  **def** \_\_repr\_\_(self):  **return** f'**{**self.x**}** **{**self.y**}**'  **class** **Grid**:  **def** \_\_init\_\_(self, N):  self.N = N  **def** \_\_contains\_\_(self, item):  **return** 0 < item.x <= self.N **and** 0 < item.y <= self.N  directions = {'R': Position(0, 1), 'U': Position(-1, 0),  'L': Position(0, -1), 'D': Position(1, 0)}  cur\_position = Position(1, 1)  grid = Grid(N)  **for** dir **in** dirs:  next\_position = cur\_position + directions[dir]  **if** next\_position **in** grid:  cur\_position = next\_position  print(cur\_position)  **시각**   * 정수 N이 입력되면 00시 00분 00초부터 N시 59분 59초까지의 모든 시각 중에서 3이 하나라도 포함되는 모든 경우의 수를 구하는 프로그램을 작성하시오. 예를 들어 1을 입력했을 때 다음은 3이 하나라도 포함되어 있으므로 세어야 하는 시각이다.   1. 00시02분03초   2. 00시13분30초   input : N  **from** **datetime** **import** time  **def** add\_a\_sec(\_time):  hour, minute, second = \_time.hour, \_time.minute, \_time.second  second += 1  **if** second == 60:  minute += 1  second = 0  **if** minute == 60:  hour += 1  minute = 0  **if** minute == 0 **and** second == 0:  print(time(hour=hour, minute=minute, second=second).\_\_str\_\_())  **return** time(hour=hour, minute=minute, second=second)  N = int(input())  cur\_time = time(hour=0, minute=0, second=0)  cnt = 0  **while**(cur\_time < time(hour=N+1, minute=0, second=0)):  *# print(cur\_time)*  cnt += 1 **if** '3' **in** cur\_time.\_\_str\_\_() **else** 0  cur\_time = add\_a\_sec(cur\_time)  print(cnt)  **왕실의 나이트**  In [11]:  **def** char2int(alphabet):  **return** ord(alphabet) - ord('a') + 1  pos = list(input())  pos = (char2int(pos[0]), int(pos[1]))  steps = [(2, 1), (2, -1), (-2, 1), (-2, -1),  (1, 2), (-1, 2), (1, -2), (-1, -2)]  cnt = 0  **for** step **in** steps:  **if** 0 < pos[0] + step[0] < 9 **and** 0 < pos[1] + step[1] < 9:  cnt += 1  print(cnt)  **게임 개발**   * 현민이는 게임 캐릭터가 맵 안에서 움직이는 시스템을 개발 중이다. 케릭터가 있는 장소는 1x1 크기의 정사각형으로 이뤄진 NxM 크기의 직사각형으로, 각각의 칸은 육지 또는 바다이다. 캐릭터는 동서남북 중 한 곳을 바라본다.   **def** find\_way():  **global** x, y, dir, grid  **for** i **in** range(4):  dir = (dir + 1) % 4  sight = (x + directions[dir][0], y + directions[dir][1])  **if** grid[sight[0]][sight[1]] != MARKED:  x = sight[0]  y = sight[1]  **return** 0  **else**:  **return** -1  N, M = map(int, input().split())  x, y, dir = map(int, input().split())  grid = []  MARKED = 1  directions = [(-1, 0), (0, -1), (1, 0), (0, 1)]  **for** i **in** range(N):  grid.append(list(map(int, input().split())))  cnt = 0  **while**(**True**):  grid[x][y] = MARKED  result = find\_way()  **if** result == -1:  x, y = (x + directions[(dir + 2) % 4][0], y + directions[(dir + 2) % 4][1])  **if** grid[x][y] == MARKED:  **break** 아이디어를 코드로 바꾸는 구현피지컬로 승부하기 구현하기 어려운 문제?   * 알고리즘은 간단한데 코드가 지나치게 길어지는 문제 * 특정 소수점 자리까지 출력해야 하는 문제 * 문자열이 입력으로 주어졌을 때 한 문자 단위로 끊어서 리스트에 넣어야 하는(파싱 해야하는)문제 등..   이런 문제를 만나면   * 언어 문법을 잘 이해해서 -> python itertools같은 표준 라이브러리로 코드 구현 하거나 * 무작정 기능 전부 작성하거나   이 책에서는 완전탐색, 시뮬레이션 유형 모두 '구현'유형으로 묶어 다룸 완전 탐색 모든 경우의 수를 주저 없이 다 계산하는 해결 방법 시뮬레이션 문제에서 제시한 알고리즘을 한 단계씩 차례대로 직접 수행해야 하는 문제 유형 파이썬 리스트 크기  * 파이썬에서 여러 개 변수 사용할 때 리스트 이용함. * 코테에서 메모리 제한 128~512MB * 메모리 제한 염두해두고 코딩해야.  | **데이터 개수(리스트 길이)** | **메모리 사용량** | | --- | --- | | 1,000 | 약 4KB | | 1,000,000 | 약 4MB | | 10,000,000 | 약 40MB |  채점 환경 대체로 다음과 같은 채점 시스템의 시간 제한 및 메모리 제한 정보   * 시간 제한 : 1초 * 메모리 제한 : 128MB   일반적으로 시간 제한 1초이고, 데이터 개수 100만개인 문제 있으면 시간복잡도는 O(NlogN)이내의 알고리즘 사용해서 문제 풀어야 한다. **예제 4-1 :: 상하좌우** 여행가 A가 N x N 정사각형 공간에 있고 무조건 (1,1)에서 시작. 상,하,좌,우로 이동 가능. 정사각형을 벗어나는 움직임은 무시됨. 계회것 주어졌을 때 A가 최종적으로 도착할 지점의 좌표 구하기   * 요구사항대로 구현 시 연산 횟수는 이동 횟수에 비례. 따라서 이동횟수 N번이면 시간복잡도 O(N). 넉넉한 편   # n받기  n = int(input())  x, y = 1, 1  plans = input().split()  # L, R, U, D에 따른 이동 방향  dx = [0, 0, -1, -1]  dy = [-1, 1, 0, 0]  move\_types = ['L', 'R', 'U', 'D']  # 이동 계획 확인  for plan in plans:  # 이동 후 좌표 구하기  for i in range(len(move\_types)):  if plan == move\_types[i]:  nx = x + dx[i]  ny = y + dy[i]  # 공간을 무시하는 경우 무시  if nx < 1 or nx> n or ny > n:  continue  # 이동 수행  x, y = nx, ny  print(x, y) **예제 4-2 :: 시각** 정수 N 입력되면 00:00:00부터 N:59:59 까지의 모든 시각 중 3이 하나라도 포함되는 모든 경우 수 구하는 프로그램 작성.   * 모든 시각의 경우를 하나 씩 세서 쉽게 풀 수 있는 문제. * **완전탐색(Brute Forcing)** 문제. 가능한 경우의 수를 모두 검사해보는 탐색 방법. * 완전탐색 문제는 일반적으로 확인(탐색)할 데이터 수가 100만개 이하일 때 사용함.   # H 입력받기  h = int(input())  count = 0  for i in range(h+1):  for j in range(60):  for k in range(60):  # 매 시각 3이 포함되어있으면 카운트 증가:  if '3' in str(i)+str(j)+str(k):  count += 1  print(count) 그리디(Greedy) 당장 좋은 것만 선택하는 그리디 알고리즘(탐욕법)은 **현재 상황에서 지금 당장 좋은 것만 고르는 방법** 이다. 특징  * 코테에서 만날 그리디 알고리즘 : 사전에 외우고 있지 않아도 풀 수 있을 가능성이 높은 문제 유형 * 암기한다고 잘 푸는 알고리즘 아님. * **창의력** 요구. * 특정 문제를 만났을 때 '단순히 현재 상황에서 가장 좋아 보이는 것만 선택해도 문제 풀 수 있는지' 파악해야 함.  **예제 3-1 :: 거스름돈** 카운터에서 거스름돈으로 사용할 500원, 100원, 50원, 10원 동전 무한히 존재. 손님에게 거슬러줘야 할 돈 N원일 때 거슬러줘야 할 **동전의 최소 개수** 구하기(N은 10의배수).   * **가장 큰 화폐**부터 돈 거슬러주기.   n = 1260  count = 0  # 큰 단위부터 확인  coin\_types = [500, 100, 50, 10]  for coin in coin\_types:  count += n//coin # 해당 화폐로 거슬러 줄 수 있는 동전 개수 세기  n %= coin  print(count)   * 시간복잡도 O(K) : K는 화폐 종류  그리디 알고리즘의 정당성  * 대부분의 문제는 그리디 알고리즘 사용해서 '최적의 해' 찾을 수 없음. * 그리디로 풀 때는 -> 그 해법이 정당한지 검토해야... * 문제 풀이를 위한 최소한의 아이디어를 떠올리고, 이것이 정당한지 검토할 수 있어야 답을 도출할 수 있다. | | |
| **기타** | 작성자 : 김준호 | | |