**<Greedy Algorithm – E.doc 4주차>**

#1541번 – 잃어버린 괄호

양수와 +,-,그리고 괄호를 가지고 만든 식의 괄호를 모두 지우고 괄호를 적절히 쳐서 이 식의 값을 최소로 만들고자 한다. 수는 0으로 시작할 수 있으며, 식은 ‘0~9’, ‘+’, 그리고 ‘-‘만으로 이루어져 있으며 가장 처음이자 마지막 문자는 숫자이다.

이 문제는 탐욕법을 적용하여 ‘-‘를 무조건 나중으로 미루는 것이 포인트이다. 우선 작은 수에서 큰 수를 뺄수록 최소이기 때문에 전부 더할 수 있는 것들은 더하고 빼기 연산을 나중에 적용한다.

|  |
| --- |
| file = input().split(**'-'**) ans = 0 **for** i **in** file[0].split(**'+'**):  ans += int(i) **for** i **in** file[1:]:  **for** j **in** i.split(**'+'**):  ans -= int(j) print(ans) |

#1931번 – 회의실 배정

한 개의 회의실이 있는데 이를 사용하고자 하는 n개의 회의에 대하여 회의실 사용표를 만들려고 한다. 각 회의 i에 대해 시작, 끝 시간이 주어져 있으며 각 회의가 겹치지 않게 하면서 회의실을 사용할 수 있는 회의의 최대 개수를 찾아보자.

이 문제도 마찬가지로 탐욕법을 적용한다면 끝나는 시간이 사실상 중요한 것이기 때문에 lambda x 를 이용하여 sort을 끝 시간 기준으로 한 뒤에 연속으로 찾아내면 된다.

|  |
| --- |
| **import** sys N = int(input()) time = [] **for** x **in** range(N):  time.append(list(map(int, sys.stdin.readline().split()))) time = sorted(time, key = **lambda** x: (x[1], x[0])) start = 0 count = 0 **for** term **in** time:  **if** term[0] >= start:  start = term[1]  count += 1 print(count) |

#1946번 – 신입 사원

다른 모든 지원자와 비교할 때 서류와 면접 둘 중 적어도 하나가 다른 지원자보다 떨어지지 않는 자만 선발한다. 즉, 어떤 지원자 A의 성적이 다른 어떤 지원자 B의 성적에 비해 서류 심사 결과와 면접 성적이 모두 떨어진다면 A는 결코 선발 될 수 없다.

모든 지원자의 성적을 두 성적 중 하나만에 대해서 정렬을 오름차순으로 한다. 즉 위에 있는 사람이 더 그 해당 종목은 성적이 낮게 되는 것이다. 이후 greedy함수를 설계하여 둘 중 한 종목이라도 1등이면 count += 1을 하고 그렇지 않은 경우 최솟값을 계속 갱신해 준다.

|  |  |  |
| --- | --- | --- |
| **import** sys **def** change(case\_num):  **for** i **in** range(case\_num):  n = int(sys.stdin.readline())  num = []  **for** j **in** range(n):  fax, interview = map(int, sys.stdin.readline().split())  num.append((fax, interview))  num = sorted(num, key = **lambda** x: x[0])  **yield** num | **def** greedy(num):  count = 0  min\_num = 99999999999999999  **for** idx, num **in** enumerate(num):  **if** num[1] < min\_num:  min\_num = num[1]  **if** num[0] == 1 **or** num[1] == 1:  count += 1  **continue  if** num[1] > min\_num:  **continue** count += 1  **return** count | **if** \_\_name\_\_ == **"\_\_main\_\_"**:  case\_num = int(input())  **for** num **in** change(case\_num):  print(greedy(num)) |

#2352번 – 반도체 설계

n개의 포트를 n개의 다른 포트와 연결하되 서로 꼬이지 않게 하기 위한 최대 연결 개수를 구하여라.

이 문제를 풀 때에는 최대 증가 수열을 이용해서 구하면 되는데, 차례로 1번부터 n번까지 연결해야 하는 포트의 번호가 주어질 때 그 리스트 안에서 제일 긴 증가하는 수열의 길이를 찾으면 된다.

|  |  |
| --- | --- |
| n = int(input()) nums = list(map(int, input().split())) dp = [1]\*n **for** i **in** range(1,n):  **for** j **in** range(i):  **if** nums[i] > nums[j]:  dp[i] = max(dp[i], dp[j]+1) print(max(dp)) | 그리디 알고리즘을 여기서 어떻게 이용해야 좋을지 생각하던 와중에 dp로 풀었는데, 그렇게 했더니 python3로는 TLE가 발생하여서 pypy3로 구현했더니 통과 되었다.  그러나 생각해보면 이것도 일종의 ‘가장 긴 증가 수열’을 찾는 것이기 떄문에 탐욕법을 사용한 것으로 봐도 되는 것 같다. |

#2138번 – 전구와 스위치

n개의 전구들의 현재 상태와 우리가 만들어야 하는 상태가 주어질 때 그 상태를 만들기 위해 스위치를 최소 몇 번 눌러야 하는지 출력하는 프로그램을 작성하는 문제이다.

처음에는 현재 index에 해당하는 문자가 도달하고자 하는 목표와 다를 때를 생각해야 하는 줄 알았는데 그게 아니라 그 이전의 것을 고려를 해 주어야 했으며, 처음에 버튼을 누르는 것이 허용되는 경우, 그리고 누르는 것이 허용되지 않는 경우로 나누어서 문제를 풀어야 했다. 처음에는 함수를 두개 작성해야 하는 줄 알았으나 생각해 보니 그냥 before리스트만 deepcopy를 이용해서 복제해 두면 되는 것이었고, 하나의 함수 안에서 전부 다 해결했다. 그러나 의문은 함수를 벗어난 이후에 왜 before리스트와 new\_before이 바뀐 상태로 유지되지 않는지 였는데, 생각해 보니 global before, new\_before를 하지 않았기 때문인 것 같다.

|  |  |
| --- | --- |
| **def** check():  **global** count, new\_count  **for** i **in** range(n):  **if** i == 0:  before[i], before[i+1] = str(abs(int(before[i])-1)), str(abs(int(before[i+1])-1))  count += 1  **elif** i == n-1:  **if** before[i-1] != after[i-1]:  before[i-1], before[i] = str(abs(int(before[i-1])-1)), str(abs(int(before[i])-1))  count += 1  **if** new[i-1] != after[i-1]:  new[i - 1],new[i] = str(abs(int(new[i - 1]) - 1)), str(abs(int(new[i]) - 1))  new\_count += 1  **else**:  **if** before[i-1] != after[i-1]:  before[i-1], before[i], before[i+1] = str(abs(int(before[i-1])-1)), str(abs(int(before[i])-1)), str(abs(int(before[i+1])-1))  count += 1  **if** new[i-1] != after[i-1]:  new[i-1], new[i], new[i+1] = str(abs(int(new[i-1])-1)), str(abs(int(new[i])-1)), str(abs(int(new[i+1])-1))  new\_count += 1  **if** before == after:  **return** count  **else**:  **if** new == after:  **return** new\_count  **else**:  **return** -1 | before, after= [],[] **for** i **in** range(n):  before.append(b[i])  after.append(a[i]) new = copy.deepcopy(before) count, new\_count = 0,0  **if** before == after:  print(0) **else**:  print(check()) |

#2812번 – 크게 만들기

n자리의 숫자가 주어졌을 때 여기서 숫자 k개를 지워서 얻을 수 있는 가장 큰 수를 구하는 프로그램을 작성하시오.

|  |  |
| --- | --- |
| **import** sys n,k = map(int, sys.stdin.readline().split()) num = str(sys.stdin.readline().strip()) file = [] **for** i **in** range(n):  file.append(num[i]) board = [] idx, left = 1, k **for** i **in** range(n):  **while** left > 0 **and** board **and** board[-1] < file[i]:  **del** board[-1]  left -= 1  board.append(file[i]) print(**''**.join(board[:n-k])) | 처음에는 온갖 방법을 다 시도해 보았고, 특히나 주어진 문자열의 index까지 저장한 리스트를 만들어서 정렬을 한 뒤에 기존의 배열이 흐트러지지 않는 선에서 답을 구한다던가 총 남겨야 하는 길이를 만족하도록 제일 큰 수를 구해 그 이후에 작은값 부터 순서대로 제거하는 등의 방법을 생각했었다.  그러나 전부 6%에서 ‘틀렸습니다’가 출력되었고, 그렇기 때문에 방법을 바꾸어서 풀었다.  스택에 넣는 방법을 이용했는데, **스택의 top에 있는 수보다 새롭게 입력할 문자열의 다음 수가 더 작으며 남은 제거해야 할 수가 남아있으며 여전히 스택에 수가 남아 있을 때 까지** while 문으로 반복하며 조건을 만족한다면 스택의 마지막 원소를 제거한다. 그리고 while 을 탈출한다면 해당 수를 스택에 넣는다.  총 자리수에서 없애야 하는 수를 뺀 만큼까지의 스택을 slicing한 값을 ‘’.join으로 문자열로 바꾼 값이 정답이 된다. |

#2212번 – 센서

첫 줄에 센서의 개수 n과 집중국의 개수 k가 주어질 때 최대 k개의 집중국의 수신 가능 영역의 길이의 합의 최솟값을 출력한다.

|  |
| --- |
| **import** sys n = int(sys.stdin.readline()) k = int(sys.stdin.readline()) num = list(map(int, sys.stdin.readline().split())) new = sorted(num) cost = [new[i+1]-new[i] **for** i **in** range(len(new)-1)] cost.sort(reverse = **True**) ans = 0 **for** i **in** range(k-1, len(cost)):  ans += cost[i] print(ans) |

#19539번 – 사과나무

높이 0부터 1씩 자라게 하는 물뿌리개와 2씩 자라게 하는 물뿌리개가 각각 하나씩 존재한다. 이를 각각 다른 나무에 사용해도 되고 합쳐서 한 나무에 3이 자랄 수 있도록 할 수 있는데, 두 물뿌리개는 반드시 동시에 사용해야 한다. 이때 주 물뿌리개를 이용해 주어진 사과나무의 배지를 만들 수 있는지 알아보는 프로그램을 작성해 보자.

|  |  |
| --- | --- |
| **import** sys n = int(sys.stdin.readline()) num = list(map(int, sys.stdin.readline().split())) board = [] num\_a, num\_b = 0,0 **for** i **in** range(n):  **if** num[i]%2 != 0:  num\_a += 1  num\_b += num[i]//2 **if** num\_b < num\_a:  print(**'NO'**) **else**:  **if** (num\_b-num\_a)%3 == 0:  print(**'YES'**)  **else**:  print(**'NO'**) | 처음에 이 문제를 접했을 때는 각각의 나무 높이의 리스트를 오름차순으로 정렬한 뒤에 두 나무 높이의 차이를 구하는 방법으로 1과 2씩 자라는 것을 해결한 이후에 3으로 나누어 봐서 확인을 해야 하는 것이라고 생각했었다.  그러나 그게 아니라 1씩 자라게 하는 물뿌리개의 최소 이용횟수는 홀수의 개수, 2씩 자라게 하는 뭉뿌리개의 최대 이용 횟수는 각 높이를 2로 나눈 몫의 합인데 전자가 후자보다 작으면 무조건 NO이고 아니라면 두 값의 차가 3으로 커버가 되는지, 즉 나누어 떨어지는지 봐야 한다. 나누어 떨어진다면 YES이고 아니라면 NO가 출력되는 것으로 프로그램을 마무리 할 수 있다. |

#1041번 – 주사위

|  |  |
| --- | --- |
| **import** sys n = int(sys.stdin.readline()) num = list(map(int, sys.stdin.readline().split())) file = [0 **for** \_ **in** range(3)] file[2] = min(num) file[0] = min(num[0], num[-1]) + min(num[1]+num[2], num[2]+num[4], num[4]+num[3], num[3]+num[1]) small = min(num[1]+num[2], num[2]+num[4], num[4]+num[3], num[3]+num[1]) small = min(small, (min(num[0], num[-1])+min(num[1], num[3], num[2], num[4]))) file[1] = small ans = 0 ans += (file[0]\*4+file[1]\*(8\*n-12)+file[2]\*((n-2)\*(5\*n-6))) **if** n ==1:  print(sum(num)-max(num)) **else**:  print(ans) | 생각보다 정답률에 비해 매우 간단한 문제였고, 어찌 보면 기하적인 측면을 조금 응용해야만 ‘탐욕법’적으로 해결이 가능한 문제라고도 할 수 있을 것이다.  결국 총 1,2,3면이 노출되는 각각의 블록에 따라서 다르게 가중치를 정해주고 직접 각각의 개수의 면에 따라 적혀진 숫자의 합의 최댓값을 구하여서 곱해 계산하면 되는 것이다.  이렇게 계산한 이후에는 잠시 틀렸다고 나왔는데, n이 1일때를 고려하지 못했기 때문이어서 n==1일떄를 따로 계산해 주니 맞았다고 통과 했다.(아직도 왜 1일떄는 따로 해야 하는지 모르겠지만 앞으로는 주어지는 입력값의 범위를 잘 살피자.) |