**<Stack – 스택>**

LIFO(Last In-First Out): 마지막에 들어간 것이 먼저 나간다.

|  |  |
| --- | --- |
| 요소 넣기 | push |
| 요소 빼기 | pop |
| 맨 위의 요소 보기 | top(peak) |
| 요소의 개수 확인하기 | size |
| 스택이 비었는지 확인하기 | empty |

위의 표에 있는 연산으로 LIFO를 수행하는 자료 구조가 stack이다.

알고리즘 문제를 풀 때 스택을 쓰는 방법

1. 리스트로 직접 구현

2. from collections import deque

#10828 – 스택

|  |
| --- |
| push X: 정수 X를 스택에 넣는 연산이다.  pop: 스택에서 가장 위에 있는 정수를 빼고, 그 수를 출력한다. 만약 스택에 들어있는 정수가 없는 경우에는 -1을 출력한다.  size: 스택에 들어있는 정수의 개수를 출력한다.  empty: 스택이 비어있으면 1, 아니면 0을 출력한다.  top: 스택의 가장 위에 있는 정수를 출력한다. 만약 스택에 들어있는 정수가 없는 경우에는 -1을 출력한다. |

첫 줄에 주어지는 명령에 맞추어서 n개를 출력하는 프로그램을 작성하자.

|  |  |
| --- | --- |
| n = int(input()) **import** sys **from** collections **import** deque file = deque() **for** \_ **in** range(n):  now = sys.stdin.readline().split()  **if** now[0] == **'push'**:  file.append(now[1])  **if** now[0] == **'top'**:  **if** file:  print(file[-1])  **else**:  print(-1)  **if** now[0] == **'size'**:  print(len(file))  **if** now[0] == **'empty'**:  **if** file:  print(0)  **else**:  print(1)  **if** now[0] == **'pop'**:  **if** file:  print(file.pop())  **else**:  print(-1) | 매우 간단한 문제이다. 잠시 시간 초과가 발생했었으나 그것은 입력 값을 input()이 아니라sys.stdin.readline()으로 구현하면 바로 해결이 되는 문제였다. |

#2504 – 괄호의 값

|  |
| --- |
| 4개의 기호 ‘(’, ‘)’, ‘[’, ‘]’를 이용해서 만들어지는 괄호열 중에서 올바른 괄호열이란 다음과 같이 정의된다.  1. 한 쌍의 괄호로만 이루어진 ‘()’와 ‘[]’는 올바른 괄호열이다.  2. 만일 X가 올바른 괄호열이면 ‘(X)’이나 ‘[X]’도 모두 올바른 괄호열이 된다.  3. X와 Y 모두 올바른 괄호열이라면 이들을 결합한 XY도 올바른 괄호열이 된다.  예를 들어 ‘(()[[]])’나 ‘(())[][]’ 는 올바른 괄호열이지만 ‘([)]’ 나 ‘(()()[]’ 은 모두 올바른 괄호열이 아니다. 우리는 어떤 올바른 괄호열 X에 대하여 그 괄호열의 값(괄호값)을 아래와 같이 정의하고 값(X)로 표시한다.  1. ‘()’ 인 괄호열의 값은 2이다.  2. ‘[]’ 인 괄호열의 값은 3이다.  3. ‘(X)’ 의 괄호값은 2×값(X) 으로 계산된다.  4. ‘[X]’ 의 괄호값은 3×값(X) 으로 계산된다.  5. 올바른 괄호열 X와 Y가 결합된 XY의 괄호값은 값(XY)= 값(X)+값(Y) 로 계산된다.  예를 들어 ‘(()[[]])([])’ 의 괄호값을 구해보자. ‘()[[]]’ 의 괄호값이 2 + 3×3=11 이므로 ‘(()[[ ]])’의 괄호값은 2×11=22 이다. 그리고 ‘([])’의 값은 2×3=6 이므로 전체 괄호열의 값은 22 + 6 = 28 이다. |

|  |  |
| --- | --- |
| **import** sys file = list(map(str, sys.stdin.readline().strip())) **from** collections **import** deque stack = deque() ans = 0 num = [] **for** i **in** file:  **if** i == **'('**:  stack.append(-2)  **elif** i == **'['**:  stack.append(-3)  **elif** i == **']'**:  temp = 0  **while True**:  **if** len(stack)==0:  print(0)  sys.exit()  k = stack.pop()  **if** k > 0:  temp += k  **elif** k == -2:  print(0)  sys.exit()  **elif** k == -3:  **if** temp == 0:  temp = 1  temp \*= 3  stack.append(temp)  **break  elif** i == **')'**:  temp = 0  **while True**:  **if** len(stack) == 0:  print(0)  sys.exit()  k = stack.pop()  **if** k > 0:  temp += k  **elif** k == -2:  **if** temp == 0:  temp = 1  temp \*= 2  stack.append(temp)  **break  elif** k == -3:  print(0)  sys.exit() answer = 0 **while** len(stack)>0:  top = stack.pop()  **if** top < 0:  print(0)  sys.exit()  answer += top print(answer) | Stack 자료구조를 이용하기 위해 from collections import deque를 이용한다. 이때 ‘(‘나 ‘[‘처럼 특정 괄호를 시작하면 stack에 각각 -2와 -3을 넣는다.  반대로 ‘)’나 ‘]’처럼 매듭 짓는 괄호가 있다면, 각각 ‘(‘와 ‘[‘가 등장할 떄 까지, 즉 -2와 -3이 등장할 때 까지의 양수가 나왔다면 임의로 temp라는 변수를 설정해서 곱해주고 드디어 -2나-3이 나오면 최종적으로 temp에 2또는 3을 곱해서 다시 stack에 넣어준다.  도중에 음수가 나오거나 아직 시작하는게 없는데 stack이 비어버렸다면 0을 출력하고 sys.exit()을 통해 완전히 프로그램을 멈춘다.  마지막에는 stack에 남아있는 수들을 처리해주고, 음수가 나오면 0을 출력하고 sys.exit()을 해준 뒤에 양수는 모두 더하여 answer을 출력한다. |

#2841 – 외계인의 기타 연주

|  |
| --- |
| 이렇게 손가락으로 프렛을 한 번 누르거나 떼는 것을 손가락을 한 번 움직였다고 한다. 어떤 멜로디가 주어졌을 때, 손가락의 가장 적게 움직이는 회수를 구하는 프로그램을 작성하시오.  다음 N개 줄에는 멜로디의 한 음을 나타내는 두 정수가 주어진다. 첫 번째 정수는 줄의 번호이고 두 번째 정수는 그 줄에서 눌러야 하는 프렛의 번호이다. 입력으로 주어진 음의 순서대로 기타를 연주해야 한다. |

|  |  |
| --- | --- |
| **import** sys count = 0 stack = [[] **for** \_ **in** range(6)] *#총 6개의 열이(줄이) 존재한다.* n,p = map(int, sys.stdin.readline().split())  **for** \_ **in** range(n):  a,b = map(int, input().split())  a -= 1  **if** len(stack[a]) == 0:  stack[a].append(b)  count += 1  **else**:  **while** stack[a][-1] > b:  stack[a].pop()  count += 1  **if** len(stack[a]) == 0:  **break  if** len(stack[a]) == 0:  stack[a].append(b)  count += 1  **elif** stack[a][-1] != b:  stack[a].append(b)  count += 1 print(count) | stack문제를 풀고자 하면 무조건 from collections import deque를 이용했던 나 자신을 반성하게 되는 문제였다. 파이썬으로는 리스트로 충분히 구현이 가능한 것이 stack 자료구조이다.  이 문제의 경우 ‘최소’이기 때문에 보이는 족족 손가락을 떼는 것이 아니라 같은 줄의 더 낮거나 같은 음이 나올 때 까지 stack에 있는 값들을 pop해 주고, 만약에 같은 음이 있다면 입력을 하지 않는다.  ~할 때까지의 상황을 구현하기 위해서는 while 문을 이용하는 것이 제일 좋다. |

#17608 – 막대기

|  |
| --- |
| n개의 막대기의 높이가 주어지고 이에 대해 오른쪽에서 볼 때 보이는 막대기의 개수를 출력해야 한다. |

|  |  |
| --- | --- |
| **import** sys n = int(sys.stdin.readline()) count = 0 num = [int(sys.stdin.readline()) **for** \_ **in** range(n)] **while** num:  now = num.pop()  count += 1  **if** num:  **while** now >= num[-1]:  num.pop()  **if** len(num) == 0:  **break** print(count) | 이 문제 역시도 stack문제에 대한 나의 고정관념을 깨 주었다.  무조건 빈 리스트/deque에 숫자를 넣으면서 시작해야 한다고 생각했으나 이 문제를 고민하는 과정에서 그게 아니라 그냥 기존에 주어진 리스트를 스택으로 이용해서 문제를 해결하면 되는 것이었다. |

#10773 – 제로

스택으로 푸는 게 맞긴 하지만 굳이 그렇게 하지 않아도 너무 쉽게 풀 수 있는 문제였다.

#1874 – 스택 수열

|  |
| --- |
| 1부터 n까지의 수를 스택에 넣었다가 뽑아 늘어놓음으로써, 하나의 수열을 만들 수 있다. 이때, 스택에 push하는 순서는 반드시 오름차순을 지키도록 한다고 하자. 임의의 수열이 주어졌을 때 스택을 이용해 그 수열을 만들 수 있는지 없는지, 있다면 어떤 순서로 push와 pop 연산을 수행해야 하는지를 알아낼 수 있다. 이를 계산하는 프로그램을 작성하라. |

|  |  |
| --- | --- |
| **import** sys stack = [] n = int(sys.stdin.readline()) now = [int(sys.stdin.readline()) **for** \_ **in** range(n)] ans = **''** idx, before = 0,1 **while** idx < n:  **if** stack:  **if** stack[-1] != now[idx]:  **if** before > now[idx]:  print(**'NO'**)  sys.exit()  **else**:  stack.append(before)  ans += **'+'** before += 1  **else**:  ans += **'-'** idx += 1  stack.pop()  **else**:  stack.append(before)  before += 1  ans += **'+' for** i **in** ans:  print(i) | 처음에는 문제를 아예 잘못 이해하고 주어진 tc조차도 통과를 못했었다. 그리고 문제를 이해한 뒤에는 NO를 출력하도록 하는 부분에서 애를 먹었는데, 1부터 n까지 모든 수를 각각 한번씩만 사용이 가능하다는 것이 문제의 조건이기 때문에 문제를 풀 때 만약에 이전에 스택에 넣은, 혹은 관여한 숫자가 현재 배열로서 넣으려고 하는/출력해야 하는 수보다 크면서 현재 스택 속의 수도 더 작을 때 NO를 출력하고 sys.exit()가 사용이 가능하다.  그리고 마지막에 정답인 ans를 그냥 출력하는 것이 아니라 각 요소별로 한 줄에 하나씩 출력하는 것을 잊지 말아야 한다. |

|  |  |
| --- | --- |
| **import** sys n = int(sys.stdin.readline()) stack = [] ans = **''** before = 1 **for** \_ **in** range(n):  num = int(sys.stdin.readline())  **if** len(stack) > 0:  **if** before > num:  **if** stack[-1] > num:  print(**'NO'**)  sys.exit()  **if** stack[-1] == num:  stack.pop()  ans += **'-'  elif** stack[-1] < num:  print(**'NO'**)  sys.exit()  **elif** before == num:  stack.append(num)  ans += **'+'** stack.pop()  ans += **'-'** before = num+1  **else**:  **for** k **in** range(before, num+1):  stack.append(k)  ans += **'+'** stack.pop()  ans += **'-'** before = num+1 | **else**:  **if** num >= before:  **for** k **in** range(before,num+1):  stack.append(k)  ans += **'+'** stack.pop()  ans += **'-'** before = num+1  **else**:  print(**'NO'**)  sys.exit()   **for** i **in** ans:  print(i)  #숫자를 미리 입력 받지 않고 입력 받을 때 마다 갱신을 하고 NO를 출력하는 등의 방법으로도 코드를 짜 보았는데, 이 방법으로 할 때는 당시에는 계속 before = num+1로 갱신해주지 않아 오류가 났으나 그것을 수정해 준 이후로 정답 처리가 되었다. |

#10799 – 쇠막대기

|  |  |
| --- | --- |
| **import** sys file = sys.stdin.readline() board = file.replace(**'()'**, str(0)) stack = [] metal,count = 0,0 **for** i **in** range(len(board)):  **if** board[i] == **'('**:  metal += 1  **elif** board[i] == **')'**:  metal -= 1  count += 1  **else**:  count += metal print(count) | 처음에는 레이저와 막대기의 구분을 처리하는 방법을 생각해내기까지 조금 시간이 걸렸지만, 처음부터 리스트로 받지 않고 문자열로 받아서 replace함수로 문자열을 다루고 ‘()’를 ‘0’으로 바꾼 뒤 시작했다.  그리고 레이저가 나올 때 마다 지금까지의 막대기의 개수를 더하고 막대기가 끝나는 경우에 한 번 더 답에 1을 더함으로써 프로그램을 마무리 하도록 했다. |

#2493 – 탑

|  |  |
| --- | --- |
| **import** sys n = int(sys.stdin.readline()) num = list(map(int, sys.stdin.readline().split())) stack = [] ans = [0 **for** \_ **in** range(n)] **for** i **in** range(n):  **if** stack:  **while** stack[-1][1] <= num[i]:  stack.pop()  **if** len(stack) == 0:  **break  if** len(stack):  ans[i] = stack[-1][0]  stack.append((i+1, num[i])) **for** i **in** ans:  print(i, end = **' '**) | 처음 풀었을 때는 말 그대로 모든 경우를 index를 거꾸로 되짚어 가며 더 높은 빌딩을 찾아 갔기 때문에 왜 스택으로 풀어야만 하는지 의문이 있었다. 그러나 역시나 시간 초과가 발생했고, 스택의 특성인 LIFO를 이용해야 한다는 생각이 들었다.  그래서 현재 입력하려 하는 높이보다 낮을 스택에 있던 빌딩들은 무조건 pop해주는 형식으로 탐색해야 하는 가짓수를 줄였다.  그리고 pop할 때에 주의할 점은 len(stack) == 0일 때를 항상 체크해 주어야 한다는 것이다. |