**DAY-5 PRACTICE PROBLEMS**

|  |  |
| --- | --- |
| 1 | There are n people in a line queuing to buy tickets, where the 0th person is at the front of the line and the (n - 1)th person is at the back of the line.  You are given a 0-indexed integer array of tickets of length n where the number of tickets that the ith person would like to buy is tickets[i].  Each person takes exactly 1 second to buy a ticket. A person can only buy 1 ticket at a time and has to go back to the end of the line (which happens instantaneously) in order to buy more tickets. If a person does not have any tickets left to buy, the person will leave the line.  Return the time taken for the person at position k (0-indexed) to finish buying tickets.    Example 1:  Input: tickets = [2,3,2], k = 2  Output: 6  Explanation:  - In the first pass, everyone in the line buys a ticket and the line becomes [1, 2, 1].  - In the second pass, everyone in the line buys a ticket and the line becomes [0, 1, 0].  The person at position 2 has successfully bought 2 tickets and it took 3 + 3 = 6 seconds.  Example 2:  Input: tickets = [5,1,1,1], k = 0  Output: 8  Explanation:  - In the first pass, everyone in the line buys a ticket and the line becomes [4, 0, 0, 0].  - In the next 4 passes, only the person in position 0 is buying tickets.  The person at position 0 has successfully bought 5 tickets and it took 4 + 1 + 1 + 1 + 1 = 8 seconds.    Constraints:   * n == tickets.length * 1 <= n <= 100 * 1 <= tickets[i] <= 100 * 0 <= k < n |
| 2 | You are keeping the scores for a baseball game with strange rules. At the beginning of the game, you start with an empty record.  You are given a list of strings operations, where operations[i] is the ith operation you must apply to the record and is one of the following:   * An integer x.   + Record a new score of x. * '+'.   + Record a new score that is the sum of the previous two scores. * 'D'.   + Record a new score that is the double of the previous score. * 'C'.   + Invalidate the previous score, removing it from the record.   Return the sum of all the scores on the record after applying all the operations.  The test cases are generated such that the answer and all intermediate calculations fit in a 32-bit integer and that all operations are valid.    Example 1:  Input: ops = ["5","2","C","D","+"]  Output: 30  Explanation:  "5" - Add 5 to the record, record is now [5].  "2" - Add 2 to the record, record is now [5, 2].  "C" - Invalidate and remove the previous score, record is now [5].  "D" - Add 2 \* 5 = 10 to the record, record is now [5, 10].  "+" - Add 5 + 10 = 15 to the record, record is now [5, 10, 15].  The total sum is 5 + 10 + 15 = 30.  Example 2:  Input: ops = ["5","-2","4","C","D","9","+","+"]  Output: 27  Explanation:  "5" - Add 5 to the record, record is now [5].  "-2" - Add -2 to the record, record is now [5, -2].  "4" - Add 4 to the record, record is now [5, -2, 4].  "C" - Invalidate and remove the previous score, record is now [5, -2].  "D" - Add 2 \* -2 = -4 to the record, record is now [5, -2, -4].  "9" - Add 9 to the record, record is now [5, -2, -4, 9].  "+" - Add -4 + 9 = 5 to the record, record is now [5, -2, -4, 9, 5].  "+" - Add 9 + 5 = 14 to the record, record is now [5, -2, -4, 9, 5, 14].  The total sum is 5 + -2 + -4 + 9 + 5 + 14 = 27.  Example 3:  Input: ops = ["1","C"]  Output: 0  Explanation:  "1" - Add 1 to the record, record is now [1].  "C" - Invalidate and remove the previous score, record is now [].  Since the record is empty, the total sum is 0.    Constraints:   * 1 <= operations.length <= 1000 * operations[i] is "C", "D", "+", or a string representing an integer in the range [-3 \* 104, 3 \* 104]. * For operation "+", there will always be at least two previous scores on the record. * For operations "C" and "D", there will always be at least one previous score on the record. |
| 3 | A valid parentheses string is either empty "", "(" + A + ")", or A + B, where A and B are valid parentheses strings, and + represents string concatenation.   * For example, "", "()", "(())()", and "(()(()))" are all valid parentheses.   A valid parentheses string s is primitive if it is nonempty, and there does not exist a way to split it into s = A + B, with A and B nonempty valid parentheses strings.  Given a valid parentheses string s, consider its primitive decomposition: s = P1 + P2 + ... + Pk, where Pi are primitive valid parentheses strings.  Return s after removing the outermost parentheses of every primitive string in the primitive decomposition of s.    Example 1:  Input: s = "(()())(())"  Output: "()()()"  Explanation:  The input string is "(()())(())", with primitive decomposition "(()())" + "(())".  After removing the outer parentheses of each part, this is "()()" + "()" = "()()()".  Example 2:  Input: s = "(()())(())(()(()))"  Output: "()()()()(())"  Explanation:  The input string is "(()())(())(()(()))", with primitive decomposition "(()())" + "(())" + "(()(()))".  After removing the outer parentheses of each part, this is "()()" + "()" + "()(())" = "()()()()(())".  Example 3:  Input: s = "()()"  Output: ""  Explanation:  The input string is "()()", with primitive decomposition "()" + "()".  After removing the outer parentheses of each part, this is "" + "" = "".    Constraints:   * 1 <= s.length <= 105 * s[i] is either '(' or ')'. * s is a valid parentheses string. |
| 4 | Given an array, for each element find the value of the nearest element to the right which is having a frequency greater than that of the current element. If there does not exist an answer for a position, then make the value ‘-1’.  **Examples:**  Input : a[] = [1, 1, 2, 3, 4, 2, 1]  Output : [-1, -1, 1, 2, 2, 1, -1]  Explanation:  Given array a[] = [1, 1, 2, 3, 4, 2, 1]  Frequency of each element is: 3, 3, 2, 1, 1, 2, 3  Lets calls Next Greater Frequency element as NGF  1. For element a[0] = 1 which has a frequency = 3,  As it has frequency of 3 and no other next element  has frequency more than 3 so '-1'  2. For element a[1] = 1 it will be -1 same logic  like a[0]  3. For element a[2] = 2 which has frequency = 2,  NGF element is 1 at position = 6 with frequency  of 3 > 2  4. For element a[3] = 3 which has frequency = 1,  NGF element is 2 at position = 5 with frequency  of 2 > 1  5. For element a[4] = 4 which has frequency = 1,  NGF element is 2 at position = 5 with frequency  of 2 > 1  6. For element a[5] = 2 which has frequency = 2,  NGF element is 1 at position = 6 with frequency  of 3 > 2  7. For element a[6] = 1 there is no element to its  right, hence -1  Input : a[] = [1, 1, 1, 2, 2, 2, 2, 11, 3, 3]  Output : [2, 2, 2, -1, -1, -1, -1, 3, -1, -1] |
| 5 | Given a number , write a program to reverse this number using stack.  **Examples:**  Input : 365  Output : 563  Input : 6899  Output : 9986 |