Problem: Minimum Number from D/I Sequence

Given a positive integer n and a string s consisting only of letters D or I, you have to find any permutation of first n positive integer that satisfy the given input string.

D means the next number is smaller, while I means the next number is greater.

**Notes**

1. Length of given string s will always equal to n - 1
2. Your solution should run in linear time and space.

**Additional Examples sourced from Geeks for Geeks**

Input: D Output: 21

Input: I Output: 12

Input: DD Output: 321

Input: II Output: 123

Input: DIDI Output: 21435

Input: IIDDD Output: 126543

Input: DDIDDIID Output: 321654798

Solution (Original)

1. public class Solution {
3. public ArrayList<Integer> findPerm(final String A, int B) {
5. int size = B;
6. ArrayList<Integer> result = new ArrayList(size);
7. for(int i=0; i<size; i++){
8. result.add(0);
9. }
10. int[] mode = new int[size];
11. int len = A.length();
13. int max=1;
14. int localMax=1;
15. result.set(0, 1);
16. mode[0] = 2;
18. // System.out.println("Starting the main loop: ");
19. for(int i=0; i<len; i++){
20. char ch = A.charAt(i);
21. if( ch == 'I' ){
22. //       System.out.println("\tCharacter is I at string position: "+(i+1));
23. int sol\_index = i+1;
24. result.set(sol\_index, max+1);
25. mode[sol\_index]=2;
26. //     System.out.println("\tSet solution for index: "+(max+1));
27. max+=1;
28. //   System.out.println("\tSol index is: "+sol\_index+" and latest max is: "+max);
29. // System.out.println("");
31. }
32. else{
33. //System.out.println("\tCharacter is D at string position: "+(i+1));
34. /\*<-- in this direction finds the last index that has I
35. character or mode as 2 and increment it's answer\*/
36. int tempIndex = i;
37. int sol\_index = i+1;
38. mode[sol\_index]=1;
39. while(mode[tempIndex]!=2){
40. tempIndex-=1;
41. }
42. /\*fouund the index and incrementing the solution for this
43. index\*/
44. //System.out.println("\tThe last index with mode I is: "+tempIndex);
45. max+=1;
46. result.set(tempIndex, max);
47. localMax = max;
48. //System.out.println("\tThe answer set for I is: "+max);
49. //System.out.println("\tGetting back to original index now from i: "+tempIndex);
50. while(tempIndex!=sol\_index){
51. tempIndex+=1;
52. localMax-=1;
53. //  System.out.println("\tfor index: "+tempIndex+" ans is: "+localMax);
54. result.set(tempIndex, localMax);
55. }
56. //System.out.println("");
58. }
59. }
60. return result;
61. }
62. }

Solution (Optimized)

1. public class Solution {
2. public ArrayList<Integer> findPerm(final String A, int B) {
4. int size = B;
5. int largest = B;
6. int smallest = 1;
7. ArrayList<Integer> result = new ArrayList(size);
9. int len = A.length();
10. for(int i=0; i<len; i++){
11. char ch = A.charAt(i);
12. if( ch=='I'){
13. result.add(smallest);
14. smallest+=1;
15. }else{
16. result.add(largest);
17. largest-=1;
18. }
19. }
20. result.add(smallest);
22. return result;
23. }
24. }

**Node:**

The Streamlined version finds any possible permutation while , the above code find the smallest possible permutation hence, it’s better. However, if not asked for smallest used streamlined, it’s much shorter and elegant.

`**’Anshul Aggarwal**