

```

1  /*
2  给定只含 "I"（增大）或 "D"（减小）的字符串 S，令 N = S.length。
3
4  返回 [0, 1, ..., N] 的任意排列 A 使得对于所有 i = 0, ..., N-1，都有：
5
6      如果 S[i] == "I"，那么 A[i] < A[i+1]
7      如果 S[i] == "D"，那么 A[i] > A[i+1]
8
9  示例 1：
10
11  输出: "IDID"
12  输出: [0,4,1,3,2]
13
14  示例 2：
15
16  输出: "III"
17  输出: [0,1,2,3]
18
19  示例 3：
20
21  输出: "DDI"
22  输出: [3,2,0,1]
23
24
25
26  提示：
27
28      1 <= S.length <= 1000
29      S 只包含字符 "I" 或 "D"。
30
31  来源：力扣（LeetCode）
32  链接：https://leetcode-cn.com/problems/di-string-match
33  著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
34  */

```

分析:

- 方法一：
 - 首先获取原始字符长度，申请返回需要的空间
 - 首位默认先给0， $num_i = 0, num_d = 0$ ，遇到I就把 num_i 自加赋值给当前位置，遇到D就把 num_d 自减1赋值给当前位置。
 - 遍历完字符串后，若 num_d 小于0，就把数组整体抬升 $-num_d$ 。
- 方法二：
 - $|num_i| + |num_d|$ 一定等于原字符串数组大小 N 。
 - $[0, N]$ ，遇到 i 就从小往大填，遇到 d 就从大往小填。

方法一:C_双计数法+整体抬升法.

```

1
2  /**
3  * Note: The returned array must be malloced, assume caller calls free().

```

```

4  */
5  int* diStringMatch(char * S, int* returnSize)
6  {
7      int str_len = strlen(S);
8      int* ret_val = (int*)malloc((str_len+1)*sizeof(int));
9      int i = 0;
10     int num_i = 0;
11     int num_d = 0;
12
13     ret_val[0] = 0;
14     for (i = 0; i < str_len; i++)
15     {
16         switch (S[i])
17         {
18             case 'I':
19                 ret_val[i+1] = ++num_i;
20                 break;
21             case 'D':
22                 ret_val[i+1] = --num_d;
23                 break;
24             default:
25                 break;
26         }
27     }
28     str_len++;
29     if(num_d<0)
30     {
31         num_d = -num_d;
32         for (i = 0; i < str_len; i++)
33         {
34             ret_val[i] += num_d;
35         }
36     }
37
38     *returnSize = str_len;
39     return ret_val;
40 }
41
42 /*
43 执行结果:
44 通过
45 显示详情
46 执行用时 :104 ms, 在所有 C 提交中击败了9.57% 的用户
47 内存消耗 :12.8 MB, 在所有 C 提交中击败了89.01%的用户
48 */

```

方法二:C_双计数直接填充法

```

1  /**
2   * Note: The returned array must be malloced, assume caller calls free().
3   */
4  int* diStringMatch(char * S, int* returnSize)
5  {
6      int str_len = strlen(S);
7      int* ret_val = (int*)malloc((str_len+1)*sizeof(int));
8      int i = 0;

```

```

9      int num_i    = 0;
10     int num_d    = str_len;
11
12     *returnSize = str_len+1;
13     for(i = 0; i < str_len; i++)
14     {
15         //switch (S[i])
16         //{
17         //    case 'I':
18         //        ret_val[i] = num_i++;
19         //    break;
20         //    case 'D':
21         //        ret_val[i] = num_d--;
22         //    break;
23         //    default:
24         //    break;
25         //}
26         if (S[i]=='I')
27         {
28             ret_val[i] = num_i++;
29         }
30         else
31         {
32             ret_val[i] = num_d--;
33         }
34     }
35     ret_val[str_len] = num_i; /*此时num_i 和 num_d 应该相等*/
36     return ret_val;
37 }
38 /*
39 执行结果(switch):
40 通过
41 显示详情
42 执行用时 :88 ms, 在所有 C 提交中击败了26.60% 的用户
43 内存消耗 :12.7 MB, 在所有 C 提交中击败了92.31%的用户
44 */
45
46 /*
47 执行结果:
48 通过
49 显示详情
50 执行用时 :84 ms, 在所有 C 提交中击败了30.85% 的用户
51 内存消耗 :12.8 MB, 在所有 C 提交中击败了90.11%的用户
52 */

```