```
/*
给定只含 "I"（增大）或 "D"（减小）的字符串 S ，令 N = S.length。

返回 [0, 1, ..., N] 的任意排列 A 使得对于所有 i = 0, ..., N-1，都有：

    如果 S[i] == "I"，那么 A[i] < A[i+1]
    如果 S[i] == "D"，那么 A[i] > A[i+1]

示例 1：

输出："IDID"
输出：[0,4,1,3,2]

示例 2：

输出："III"
输出：[0,1,2,3]

示例 3：

输出："DDI"
输出：[3,2,0,1]



提示：

    1 <= S.length <= 1000
    S 只包含字符 "I" 或 "D"。

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/di-string-match
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。
*/
```

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就从小往到大填,遇到d就从大往小填.

方法一:C_双计数法+整体抬升法.

```

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

```c
 */
int* diStringMatch(char * S, int* returnSize)
{
    int  str_len = strlen(S);
    int* ret_val = (int*)malloc((str_len+1)*sizeof(int));
    int  i       = 0;
    int  num_i   = 0;
    int  num_d   = 0;

    ret_val[0] = 0;
    for (i = 0; i < str_len; i++)
    {
        switch (S[i])
        {
            case 'I':
                ret_val[i+1] = ++num_i;
            break;
            case 'D':
                ret_val[i+1] = --num_d;
            break;
            default:
            break;
        }
    }
    str_len++;
    if(num_d<0)
    {
        num_d = -num_d;
        for (i = 0; i < str_len; i++)
        {
            ret_val[i] += num_d;
        }
    }

    *returnSize = str_len;
    return ret_val;
}

/*
执行结果:
通过
显示详情
执行用时 :104 ms, 在所有 C 提交中击败了9.57% 的用户
内存消耗 :12.8 MB, 在所有 C 提交中击败了89.01%的用户
*/
```

方法二:C_双计数直接填充法

```c
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* diStringMatch(char * S, int* returnSize)
{
    int  str_len = strlen(S);
    int* ret_val = (int*)malloc((str_len+1)*sizeof(int));
    int  i       = 0;
```

```c
    int  num_i   = 0;
    int  num_d   = str_len;

    *returnSize = str_len+1;
    for(i = 0; i < str_len; i++)
    {
        //switch (S[i])
        //{
        //    case 'I':
        //        ret_val[i] = num_i++;
        //    break;
        //    case 'D':
        //        ret_val[i] = num_d--;
        //    break;
        //    default:
        //    break;
        //}
        if (S[i]=='I')
        {
            ret_val[i] = num_i++;
        }
        else
        {
            ret_val[i] = num_d--;
        }
    }
    ret_val[str_len] = num_i;/*此时num_i 和 num_d 应该相等*/
    return ret_val;
}
/*
执行结果(switch):
通过
显示详情
执行用时 :88 ms, 在所有 C 提交中击败了26.60% 的用户
内存消耗 :12.7 MB, 在所有 C 提交中击败了92.31%的用户
*/

/*
执行结果:
通过
显示详情
执行用时 :84 ms, 在所有 C 提交中击败了30.85% 的用户
内存消耗 :12.8 MB, 在所有 C 提交中击败了90.11%的用户
*/
```

AlimyBreak
2019.09.05