

```

1  /*
2  给定一个字符串S，通过将字符串S中的每个字母转变大小写，我们可以获得一个新的字符串。返回所有
   有可能得到的字符串集合。
3
4  示例：
5  输入：S = "a1b2"
6  输出：["a1b2", "a1B2", "A1b2", "A1B2"]
7
8  输入：S = "3z4"
9  输出：["3z4", "3Z4"]
10
11 输入：S = "12345"
12 输出：["12345"]
13
14 注意：
15
16     S 的长度不超过12。
17     S 仅由数字和字母组成。
18
19 来源：力扣（LeetCode）
20 链接：https://leetcode-cn.com/problems/letter-case-permutation
21 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。
22 */

```

分析：

- 若要知道所有可能的组数，显然需要先遍历一遍所有原始字符串数组，得到英文字母的总个数（及他们出现的位置），根据排列组合的知识，若英文字母的总个数为 $k$ ，那么待返回字符串集合的元素个数为 $2^k$ 。
- $2^k$ 中情况应该如何遍历了？我想到了二进制数的定义，对于 $k$ 个二进制位的数刚好能表示 $2^k$ 次方情况（ $0 \rightarrow 2^k - 1$ ），那么对应二进制上如果为0，则对应位置字母不做翻转，若对应二进制位上为1，则对应字母做翻转，二进制位与英文字母位置的对应关系，通过一个辅助数组 $en\_ind$ 来维护，基本原则是低bit位对应低索引。
- 时间复杂度： $O(N) + O(2^k k)$ ，空间复杂度 $O(N)$ 。
- 上述两种算法均为位运算法
- 方法三：回溯法。

方法一:C\_Solution:

```

1  char ** letterCasePermutation(char * S, int* returnSize)
2  {
3
4      int     str_len     = strlen(S)                ;
5      char*   mark        = (char*)malloc(str_len*sizeof(char)) ;
6      char*   en_ind      = (char*)malloc(str_len*sizeof(char)) ;
7      int     num_en      = 0                        ;
8      int     i           = 0                        ;
9      char**  ret_val     = NULL                    ;
10     int     _returnSize = 0                        ;

```

```

11     char*    temp        = NULL                                ;
12     int      j = 0;
13     for(i = 0; i < str_len ; i++)
14     {
15         if((S[i] >='A') && (S[i]<='Z'))
16         {
17             mark[i] = 1;
18             en_ind[num_en++] = i;
19         }
20         else if((S[i] >='a') && (S[i]<='z'))
21         {
22             mark[i] = 2;
23             en_ind[num_en++] = i;
24         }
25     }
26
27     _returnSize = pow(2,num_en);
28     ret_val = (char**)malloc(sizeof(char*)*_returnSize);
29     for(i = 0; i < _returnSize;i++) /*0~2^n-1*/
30     {
31         temp = (char*)malloc((str_len+1)*sizeof(char));
32         temp[str_len] = 0;
33         memcpy(temp,S,str_len*sizeof(char));
34         for(j=0;j<num_en;j++)
35         {
36             if(i&arr[j]) /*对应二进制为1就大小写之间互转*/
37             {
38                 if(mark[en_ind[j]]==1) //原为大写字母
39                 {
40                     temp[en_ind[j]] |= 0x20;
41                 }
42                 else
43                 {
44                     temp[en_ind[j]] &= ~0x20;
45                 }
46             }
47         }
48
49         ret_val[i] = temp;
50
51     }
52
53     free(mark);
54     mark = NULL;
55     free(en_ind);
56     en_ind = NULL;
57
58     *returnSize = _returnSize;
59     return ret_val;
60 }
61
62 /*
63 执行用时 :28 ms, 在所有 C 提交中击败了13.33%的用户
64 内存消耗 :9.9 MB, 在所有 C 提交中击败了90.91%的用户
65 */
66

```

方法二：C\_Solution，优化mark数组。

```
1 char ** letterCasePermutation(char * S, int* returnSize)
2 {
3
4     int    str_len      =   strlen(S)                                ;
5     char*   mark        =   (char*)malloc(str_len*sizeof(char))      ;
6     char*   en_ind      =   (char*)malloc(str_len*sizeof(char))      ;
7     int     num_en      =   0                                        ;
8     int     i           =   0                                        ;
9     char**  ret_val     =   NULL                                    ;
10    int     __returnSize =   0                                        ;
11    char*   temp         =   NULL                                    ;
12    int     j           =   0                                        ;
13    int     arr[12]      =   {   0x0001,0x0002,0x0004,0x0008,
14                                0x0010,0x0020,0x0040,0x0080,
15                                0x0100,0x0200,0x0400,0x0800
16                                };
17
18    /*先遍历一遍,获取字符个数和他们出现的位置,o(n)*/
19    for(i = 0; i < str_len ; i++)
20    {
21        if((S[i] >='A') && (S[i]<='Z'))
22        {
23            mark[num_en]      = 1;    /*标记为大写字母*/
24            en_ind[num_en++]   = i;
25        }
26        else if((S[i] >='a') && (S[i]<='z'))
27        {
28            mark[num_en]      = 2;    /*标记为小写字母*/
29            en_ind[num_en++]   = i;
30        }
31    }
32
33    /*计算排列个数及每种排列情况*/
34    __returnSize = pow(2,num_en);    /*有2的n次方种可能*/
35    ret_val = (char**)malloc(sizeof(char*)*__returnSize);
36    for(i = 0; i < __returnSize;i++)
37    {
38        temp      =   (char*)malloc((str_len+1)*sizeof(char)) ;/*多申
39        请一字节作为字符串结束标记*/
40        temp[str_len] = 0 ;
41        memcpy(temp,S,str_len*sizeof(char)) ;/*先拷
42        贝过来*/
43        for( j = 0 ; j < num_en ; j++ )
44        {
45            if(i&arr[j])    // 0x01 << j
46            {
47                if(mark[en_ind[j]]==1) //原为大写字母
48                {
49                    temp[en_ind[j]] |= 0x20;
50                }
51            }
52            else
53            {
54                temp[en_ind[j]] |= 0x40;
55            }
56        }
57    }
58    *returnSize = __returnSize;
59    return ret_val;
60 }
```

```

51         temp[en_ind[j]] &= ~0x20;
52     }
53 }
54 }
55     ret_val[i] = temp;
56 }
57
58 /*回收动态资源*/
59 free(mark);
60 mark = NULL;
61 free(en_ind);
62 mark = NULL;
63 /*返回处理*/
64 *returnSize = __returnSize;
65 return ret_val;
66 }
67
68 /*
69 执行结果: 通过
70 显示详情
71 执行用时 :24 ms, 在所有 C 提交中击败了26.67%的用户
72 内存消耗 :9.9 MB, 在所有 C 提交中击败了90.91%的用户
73 */
74

```

### 方法三:C++\_回溯法

```

1  class Solution
2  {
3
4      private:
5          void helper(    vector<string>&    vs
6                        string&            cur_s
7                        int                left
8                        int                change_times
9                        )
10         {
11             vs.push_back(cur_s);
12             for(int i = left ; i < cur_s.size() ; i++)
13             {
14                 /*有字母*/
15                 if( (cur_s[i]<='z' && cur_s[i] >= 'a')
16                   ||(cur_s[i]<='Z' && cur_s[i] >= 'A')
17                 )
18                 {
19                     cur_s[i] ^= (int)('z' - 'Z');
20                     helper(vs,cur_s,i+1,change_times+1);
21                 }
22             }
23         }
24
25     public:
26         vector<string> letterCasePermutation(string S)
27         {

```

```
28         vector<string> vs;
29
30         helper(vs,s,0,0);
31         return vs;
32     }
33 };
34
35 /*
36 执行结果:
37 通过
38 显示详情
39 执行用时 :8 ms, 在所有 cpp 提交中击败了97.20% 的用户
40 内存消耗 :11.7 MB, 在所有 cpp 提交中击败了97.98%的用户
41 */
```

---

AlimyBreak  
2019.11.26 增加回溯法(方法三)  
2019.07.18