```
/*
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。返回所有可能得到的字符串集合。

示例:
输入: S = "a1b2"
输出: ["a1b2", "a1B2", "A1b2", "A1B2"]

输入: S = "3z4"
输出: ["3z4", "3Z4"]

输入: S = "12345"
输出: ["12345"]

注意：

    S 的长度不超过12。
    S 仅由数字和字母组成。

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/letter-case-permutation
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：

- 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:

```
char ** letterCasePermutation(char * S, int* returnSize)
{

    int     str_len     = strlen(S)                             ;
    char*   mark        = (char*)malloc(str_len*sizeof(char))   ;
    char*   en_ind      = (char*)malloc(str_len*sizeof(char))   ;
    int     num_en      = 0                                     ;
    int     i           = 0                                     ;
    char**  ret_val     = NULL                                  ;
    int     _returnSize = 0                                     ;
```

```
    char*   temp        = NULL                                  ;
    int     j = 0;
    for(i = 0; i < str_len ; i++)
    {
        if((S[i] >='A') && (S[i]<='Z'))
        {
            mark[i]  = 1;
            en_ind[num_en++] = i;
        }
        else if((S[i] >='a') && (S[i]<='z'))
        {
            mark[i]  = 2;
            en_ind[num_en++] = i;
        }
    }

    _returnSize = pow(2,num_en);
    ret_val = (char**)malloc(sizeof(char*)*_returnSize);
    for(i = 0; i < _returnSize;i++) /*0~2^n-1*/
    {
        temp = (char*)malloc((str_len+1)*sizeof(char));
        temp[str_len] = 0;
        memcpy(temp,S,str_len*sizeof(char));
        for(j=0;j<num_en;j++)
        {
            if(i&arr[j]) /*对应二进制为1就大小写之间互转*/
            {
                if(mark[en_ind[j]]==1) //原为大写字母
                {
                    temp[en_ind[j]] |= 0x20;
                }
                else
                {
                    temp[en_ind[j]] &= ~0x20;
                }
            }
        }

        ret_val[i] = temp;

    }

    free(mark);
    mark = NULL;
    free(en_ind);
    en_ind = NULL;

    *returnSize = _returnSize;
    return ret_val;
}

/*
执行用时 :28 ms, 在所有 C 提交中击败了13.33%的用户
内存消耗 :9.9 MB, 在所有 C 提交中击败了90.91%的用户
*/

```

方法二：C_Solution ，优化$mark$数组.

```
char ** letterCasePermutation(char * S, int* returnSize)
{

    int     str_len         =   strlen(S)                               ;
    char*   mark            =   (char*)malloc(str_len*sizeof(char))     ;
    char*   en_ind          =   (char*)malloc(str_len*sizeof(char))     ;
    int     num_en          =   0                                       ;
    int     i               =   0                                       ;
    char**  ret_val         =   NULL                                    ;
    int     __returnSize    =   0                                       ;
    char*   temp            =   NULL                                    ;
    int     j               =   0                                       ;
    int     arr[12]         =   {   0x0001,0x0002,0x0004,0x0008,
                                    0x0010,0x0020,0x0040,0x0080,
                                    0x0100,0x0200,0x0400,0x0800
                                };

    /*先遍历一遍,获取字符个数和他们出现的位置,O(n)*/
    for(i = 0; i < str_len ; i++)
    {
        if((S[i] >='A') && (S[i]<='Z'))
        {
            mark[num_en]        = 1;    /*标记为大写字母*/
            en_ind[num_en++]    = i;
        }
        else if((S[i] >='a') && (S[i]<='z'))
        {
            mark[num_en]        = 2;    /*标记为小写字母*/
            en_ind[num_en++]    = i;
        }
    }

    /*计算排列个数及每种排列情况*/
    __returnSize = pow(2,num_en);    /*有2的n次方种可能*/
    ret_val = (char**)malloc(sizeof(char*)*__returnSize);
    for(i = 0; i < __returnSize;i++)
    {
        temp            =   (char*)malloc((str_len+1)*sizeof(char)) ;/*多申请一字节作为字符串结束标记*/
        temp[str_len]   =   0                                       ;
        memcpy(temp,S,str_len*sizeof(char))                         ;/*先拷贝过来*/
        for( j = 0 ; j < num_en ; j++ )
        {
            if(i&arr[j])    // 0x01 << j
            {
                if(mark[en_ind[j]]==1) //原为大写字母
                {
                    temp[en_ind[j]] |= 0x20;
                }
                else
                {
```

```
                    temp[en_ind[j]] &= ~0x20;
                }
            }
        }
        ret_val[i] = temp;
    }

    /*回收动态资源*/
    free(mark);
    mark = NULL;
    free(en_ind);
    mark = NULL;
    /*返回处理*/
    *returnSize = __returnSize;
    return ret_val;
}

/*
执行结果：通过
显示详情
执行用时 :24 ms, 在所有 C 提交中击败了26.67%的用户
内存消耗 :9.9 MB, 在所有 C 提交中击败了90.91%的用户
*/

```

方法三:C++_回溯法

```
class Solution
{

    private:
        void helper(    vector<string>&     vs              ,
                        string&             cur_s           ,
                        int                 left            ,
                        int                 change_times
                    )
        {
            vs.push_back(cur_s);
            for(int i = left ; i < cur_s.size() ; i++)
            {
                /*有字母*/
                if(   (cur_s[i]<='z' && cur_s[i] >= 'a')
                    ||(cur_s[i]<='Z' && cur_s[i] >= 'A')
                )
                {
                    cur_s[i] ^= (int)('z' - 'Z');
                    helper(vs,cur_s,i+1,change_times+1);
                }
            }
        }

    public:
        vector<string> letterCasePermutation(string S)
        {
```

```
            vector<string> vs;

            helper(vs,S,0,0);
            return vs;
        }
};

/*
执行结果：
通过
显示详情
执行用时 :8 ms, 在所有 cpp 提交中击败了97.20% 的用户
内存消耗 :11.7 MB, 在所有 cpp 提交中击败了97.98%的用户
*/
```

---

AlimyBreak
2019.11.26 增加回溯法(方法三)
2019.07.18