```
/*
对于非负整数 X 而言，X 的数组形式是每位数字按从左到右的顺序形成的数组。例如，如果 X = 1231，那么其数组形式为 [1,2,3,1]。

给定非负整数 X 的数组形式 A，返回整数 X+K 的数组形式。



示例 1：

输入：A = [1,2,0,0], K = 34
输出：[1,2,3,4]
解释：1200 + 34 = 1234
解释 2：

输入：A = [2,7,4], K = 181
输出：[4,5,5]
解释：274 + 181 = 455
示例 3：

输入：A = [2,1,5], K = 806
输出：[1,0,2,1]
解释：215 + 806 = 1021
示例 4：

输入：A = [9,9,9,9,9,9,9,9,9,9], K = 1
输出：[1,0,0,0,0,0,0,0,0,0,0]
解释：9999999999 + 1 = 10000000000


提示：

1 <= A.length <= 10000
0 <= A[i] <= 9
0 <= K <= 10000
如果 A.length > 1，那么 A[0] != 0

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/add-to-array-form-of-integer
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:

- 和67.二进制求和类似，只不过此题的不是两个同类型数组,所以这里要率先统计K对应数组形式的长度。
  - 计算得到K对应数组的长度k_len，取$ASize$和k_len中的最大者作为下面依次按位累加的上限，并根据这个上限来确定处理类型,时间复杂度$O(5)$.
  - 申请动态空间，由于可能存在位溢出问题，默认多申请一个int的空间

- 循环处理每位加法，根据处理类型，确定谁的位数先被处理完，就直接加0；时间复杂度为$O(\max(ASize, k_len))$
- 处理最后的一位的进位和处理返回长度，时间复杂度$O(1)$
- 反转数组,时间复杂度$O(\max(ASize, k_len))$
- 综合时间复杂度:$O(\max(ASize, k\_len))$,空间复杂度:$O(1)$

方法一：C_Solution

```
/*
对于非负整数 X 而言，X 的数组形式是每位数字按从左到右的顺序形成的数组。例如，如果 X = 1231，那么其数组形式为 [1,2,3,1]。

给定非负整数 X 的数组形式 A，返回整数 X+K 的数组形式。



示例 1：

输入：A = [1,2,0,0], K = 34
输出：[1,2,3,4]
解释：1200 + 34 = 1234
解释 2：
输入：A = [2,7,4], K = 181
输出：[4,5,5]
解释：274 + 181 = 455
示例 3;
输入：A = [2,1,5], K = 806
输出：[1,0,2,1]
解释：215 + 806 = 1021
示例 4：

输入：A = [9,9,9,9,9,9,9,9,9,9], K = 1
输出：[1,0,0,0,0,0,0,0,0,0,0]
解释：9999999999 + 1 = 10000000000


提示：

1 <= A.length <= 10000
0 <= A[i] <= 9
0 <= K <= 10000
如果 A.length > 1，那么 A[0] != 0
*/


/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
/* [left,right]*/
void int_arr_reverse(int* arr, int left , int right)
{
    int temp = 0;
```

```
    while(left<right)
    {
        temp        =   arr[left]   ;
        arr[left]   =   arr[right]  ;
        arr[right]  =   temp        ;
        left++                      ;
        right--                     ;
    }
}

int* addToArrayForm(    int*        A           ,
                        int         ASize       ,
                        int         K           ,
                        int*        returnSize
                    )
{
    int     k_len       = 0     ;
    int     len         = 0     ;
    int     temp1       = K     ;
    int     temp2       = 0     ;
    int     carry_flag  = 0     ;
    int*    ret_val     = NULL  ;
    int     i           = 0     ;
    int     type        = 0     ;

    while(temp1)
    {
        k_len++;
        temp1 /= 10;
    }
    if(k_len>ASize)
    {
        len     =   k_len   ;
        type    =   1       ;
    }
    else
    {
        len     =   ASize   ;
        type    =   2       ;
    }

    ret_val = (int*)malloc((len+1)*sizeof(int));

    for(i = 0; i < len ; i++)
    {
        if(type==2)
        {
            temp2 = K%10 + A[ASize-1-i] +carry_flag;
        }
        else
        {
            temp2 = K%10 + carry_flag;
            if(i<=(ASize-1))
```

```
            {
                temp2 += A[ASize-1-i];
            }
        }
        K = K / 10;
        if(temp2>=10)
        {
            ret_val[i] = temp2 - 10;
            carry_flag = 1;
        }
        else
        {
            ret_val[i] = temp2;
            carry_flag = 0;
        }

    }

    if(carry_flag == 0)
    {
        int_arr_reverse(ret_val,0,len-1);

    }
    else
    {
        ret_val[len] = 1;
        int_arr_reverse(ret_val,0,len);
        len ++;
    }

    *returnSize = len;
    return ret_val;
}

/*
执行结果：通过
显示详情
执行用时 :120 ms
, 在所有 C 提交中击败了100.00%的用户
内存消耗 :17.1 MB, 在所有 C 提交中击败了81.58%的用户
*/
```

AlimyBreak
2019.08.06