```c
/**
 * Return an array of arrays of size *returnSize.
 * The sizes of the arrays are returned as *returnColumnSizes array.
 * Note: Both returned array and *columnSizes array must be malloced, assume caller calls free().
 */
#define TOTAL_LETTER_COUNT          (26)

struct group {
    int* letter_map;
    struct group* next;
    struct node* list;
    int letter_count;
    int count;
};

struct group_head {
    int next;
    struct group* group;
};

struct node {
    char* text;
    struct node* next;
};

void resetGrouphead(struct group_head* mem, int length)
{
    int index;

    for(index = 0; index < length; index++)
    {
        mem[index].next = INT_MIN;
        mem[index].group = NULL;
    }
}

struct group* createGroup(int letter_count, struct node* list, int* letter_map)
{
    struct group* group;

    group = (struct group*)malloc(sizeof(struct group));
    group->letter_count = letter_count;
    group->list = list;
    group->letter_map = letter_map;
    group->count = 1;
    group->next = NULL;
    return group;
}


#define ALLOC_LENGTH    (2)

char*** groupAnagrams(char** strs, int strsSize, int* returnSize, int** returnColumnSizes) {
    char*** result;
    char** list;
    struct group_head* num_map;
    int* letter_map;
    int last_head;
    int first_head;
    struct node* tmp_node;
    struct group* group;
    int index;
    int str_index;
    int alloc_length;
    int str_count;
    int letter_count;
    int letter_index;

    alloc_length = ALLOC_LENGTH;
    *returnColumnSizes = (int*)malloc(sizeof(int)*strsSize);
    num_map = (struct group_head*)malloc(sizeof(struct group_head)*(alloc_length+1));
    resetGrouphead(num_map,alloc_length+1);
    tmp_node = (struct node*)malloc(sizeof(struct node)*strsSize);

    last_head = INT_MIN;
    first_head = INT_MIN;
    *returnSize = 0;
    letter_map = NULL;

    for(index = 0; index < strsSize; index++)
    {
        str_index = 0;
        str_count = 0;
        letter_count = 0;
        tmp_node[index].text = strs[index];
        tmp_node[index].next = NULL;

        if(letter_map == NULL)
        {
            letter_map = (int*)calloc(TOTAL_LETTER_COUNT, sizeof(int));
        }else
        {
            free(letter_map);
            letter_map = (int*)calloc(TOTAL_LETTER_COUNT, sizeof(int));
        }

        //memset(letter_map, 0x0, sizeof(int)*TOTAL_LETTER_COUNT);
        while(strs[index][str_index] != '\0')
        {
            str_count += strs[index][str_index] - 'a' + 1;
            letter_map[strs[index][str_index] - 'a']++;
            str_index++;
        }

        //printf("s:[%s], str_count:[%d], alloc_length:[%d]\n", strs[index], str_count, alloc_length);


        if(str_count > alloc_length)
        {
            num_map = (struct group_head*)realloc(num_map,sizeof(struct group_head)*(str_count*2+1));
            resetGrouphead(&num_map[alloc_length+1],str_count*2-alloc_length);
            alloc_length = str_count*2;
        }

        if(num_map[str_count].group == NULL)
        {
            (*returnSize)++;
            group = createGroup(str_index, &tmp_node[index], letter_map);
            //printf("Add group:[0x%x], tmp_node:[0x%x]\n", group, &tmp_node[index]);

            num_map[str_count].group = group;

            if(first_head == INT_MIN)
            {
                first_head = str_count;
            }

            if(last_head != INT_MIN && last_head != str_count)
            {
                num_map[last_head].next = str_count;
            }

            last_head = str_count;
            letter_map = NULL;
        }else
        {
            group = num_map[str_count].group;
            //printf("Search group:[0x%x]\n", group);
            while(group)
            {
                letter_index = 0;

                if(group->letter_count == str_index)
                {
                    while(letter_index < TOTAL_LETTER_COUNT)
                    {
                        if(letter_map[letter_index] != group->letter_map[letter_index])
                        {
                            break;
                        }
                        letter_index++;
                    }
                }

                if(letter_index == TOTAL_LETTER_COUNT)
                {

                    group->count++;
                    tmp_node[index].next = group->list;
                    group->list = &tmp_node[index];
                    break;
                }

                group = group->next;
            }

            if(group == NULL)
            {
                (*returnSize)++;
                group = createGroup(str_index, &tmp_node[index], letter_map);
                group->next = num_map[str_count].group;
                num_map[str_count].group = group;
                letter_map = NULL;
            }
        }

    }

    result = (char***)malloc(sizeof(char**)*(*returnSize));
    index = 0;
    //printf("first_head:[%d], total_count:[%d]\n", first_head, *returnSize);
    while(first_head != INT_MIN)
    {
        //printf("first_head:[%d], next:[%d]\n", first_head, num_map[first_head].next);
        group = num_map[first_head].group;

        while(group)
        {
            //printf("Index:[%d], list:[0x%x], group count:[%d]\n", index, group->list, group->count);
            list = (char**)malloc(sizeof(char*)*group->count);
            result[index] = list;
            //printf("xxxxxxx\n");
            (*returnColumnSizes)[index] = group->count;
            tmp_node = group->list;

            for(str_index = 0; str_index < group->count; str_index++)
            {
                list[str_index] = tmp_node->text;
                //printf("str_index:[%d], s:[%s]\n", str_index, tmp_node->text);
                tmp_node = tmp_node->next;

            }
            //printf("xxx\n");
            group = group->next;
            index++;
        }

        first_head = num_map[first_head].next;
    }

    //printf("done\n");

    return result;

}
```