1189. “气球” 的最大数量（简单）

1. 题目描述

   给你一个字符串 `text`，你需要使用 `text` 中的字母来拼凑尽可能多的单词 **"balloon"（气球）**。

   字符串 `text` 中的每个字母最多只能被使用一次。请你返回最多可以拼凑出多少个单词 **"balloon"**。

   示例 1：


   ```
   输入：text = "nlaebolko"
   输出：1
   ```

   示例 2：


   ```
   输入：text = "loonbalxballpoon"
   输出：2
   ```

   示例 3：

   ```
   输入：text = "leetcode"
   输出：0
   ```

   提示：

   - `1 <= text.length <= 10^4`
   - `text` 全部由小写英文字母组成

2. 简单实现

```cpp
class Solution {
public:
    int maxNumberOfBalloons(string text) {
        int b_cnt = 0, a_cnt = 0, l_cnt = 0, o_cnt = 0, n_cnt = 0;
        for(int i = 0; i < text.size(); i++){
            if(text[i] == 'b')
                b_cnt++;
            else if(text[i] == 'a')
                a_cnt++;
            else if(text[i] == 'l')
                l_cnt++;
            else if(text[i] == 'o')
```

```
                o_cnt++;
            else if(text[i] == 'n')
                n_cnt++;
        }
        int ans = min(min(min(min(b_cnt, a_cnt), l_cnt/2), o_cnt/2), n_cnt);
        return ans;
    }
};
```

1190. 反转每对括号间的子串（中等）

1. 题目描述

   给出一个字符串 `s`（仅含有小写英文字母和括号）。请你按照从括号内到外的顺序，逐层反转每对匹配括号中的字符串，并返回最终的结果。

   注意，您的结果中 **不应** 包含任何括号。

   示例 1：

   ```
   输入：s = "(abcd)"
   输出："dcba"
   ```

   示例 2：

   ```
   输入：s = "(u(love)i)"
   输出："iloveu"
   ```

   示例 3：

   ```
   输入：s = "(ed(et(oc))el)"
   输出："leetcode"
   ```

   示例 4：

   ```
   输入：s = "a(bcdefghijkl(mno)p)q"
   输出："apmnolkjihgfedcbq"
   ```

   提示：

   - `0 <= s.length <= 2000`
   - `s` 中只有小写英文字母和括号
   - 我们确保所有括号都是成对出现的

2. 简单实现

   用双向队列，逐层翻转即可

```
class Solution {
public:
    string reverseParentheses(string s) {
        deque<string> cache;
```

```
        int idx = 0;
        string temp = "";
        while(idx < s.size()){
            if(s[idx] == '('){
                cache.push_back(temp);//即使为空也要Push，因为要处理"a((bc)d)"中的连续左
括号
                temp = "";
                idx++;
            }
            else if(s[idx] == ')'){
                reverse(temp.begin(), temp.end());//翻转
                temp = cache.back() + temp;//与上一层的前缀合并，并继续向后寻找后缀
                cache.pop_back();
                idx++;
            }
            else{
                temp += s[idx];
                idx++;
            }
        }
        cache.push_back(temp);//最后一层的记得push进去
        string ans = "";
        while(!cache.empty()){
            ans += cache.front();
            cache.pop_front();
        }
        return ans;
    }
};
```

1191. K 次串联后最大子数组之和（中等）

1. 题目描述

给你一个整数数组 `arr` 和一个整数 `k`。

首先，我们要对该数组进行修改，即把原数组 `arr` 重复 `k` 次。

> 举个例子，如果 `arr = [1, 2]` 且 `k = 3`，那么修改后的数组就是 `[1, 2, 1, 2, 1, 2]`。

然后，请你返回修改后的数组中的最大的子数组之和。

注意，子数组长度可以是 `0`，在这种情况下它的总和也是 `0`。

由于 **结果可能会很大**，所以需要 **模（mod）** `10^9 + 7` 后再返回。

示例 1：

```
输入：arr = [1,2], k = 3
输出：9
```

示例 2：

```
输入：arr = [1,-2,1], k = 5
输出：2
```

示例 3：

```
输入：arr = [-1,-2], k = 7
输出：0
```

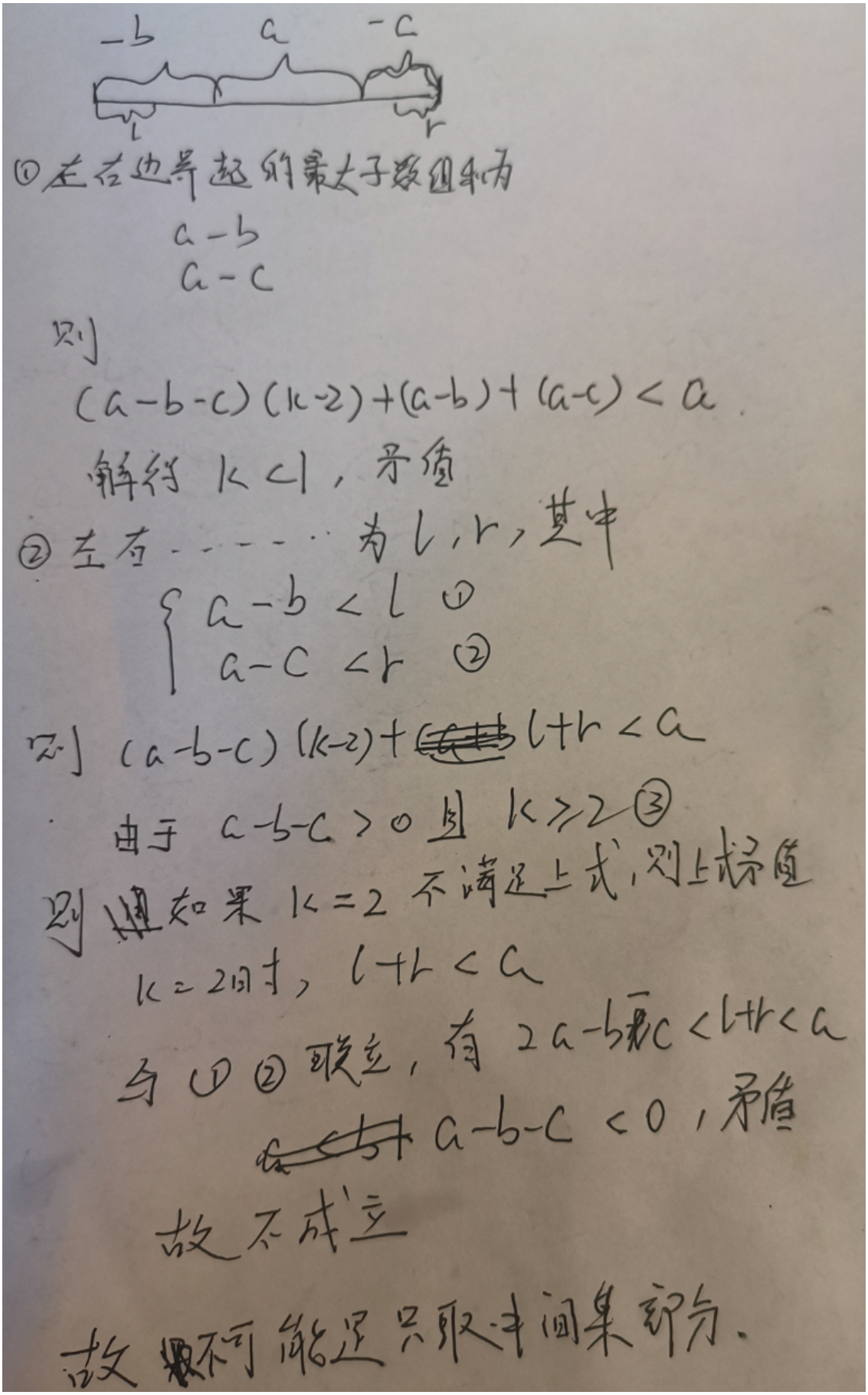
提示：

- `1 <= arr.length <= 10^5`
- `1 <= k <= 10^5`
- `-10^4 <= arr[i] <= 10^4`

2. 简单实现

分类讨论：

- 如果数组内所有元素都小于等于0，则无论k为多少，结果都是0
- 如果数组内所有元素都大于等于0，则结果为`整个数组的元素和 * k`
- 如果数组所有元素和小于等于0，则对所有k>=2的情况，其拼接后求得的结果都和k=2相同（因为最大子数组不可能（小于0）或没必要（等于0）跨越一整个数组）
- 如果数组所有元素和大于0，则结果为`整个数组的元素和 * （k-2）+ 以数组右边界为起点的最大子数组和 + 以数组左边界为起点的最大子数组和`
  - PS：一开始觉得还可能出现在数组的中间部分，后来列式子算了一下不可能，过程如下：




```
class Solution {
public:
```

```cpp
int MOD = 1e9 + 7;
void getMax(vector<int>& arr, int& ans){//获取arr数组的最大子数组和
    int idx = 0;
    int temp = 0;
    ans = 0;
    while(idx < arr.size()){
        temp += arr[idx];
        ans = max(ans, temp);
        temp = max(temp, 0);//小于0则清零temp
        idx++;
    }
}
int kConcatenationMaxSum(vector<int>& arr, int k) {
    if(k == 1){//k=1时无需考虑拼接，直接找即可
        int ans;
        getMax(arr, ans);
        return ans;
    }
    int sum = 0, pos_num = 0, neg_num = 0;
    int len = arr.size();
    for(int i = 0; i < len; i++){
        if(arr[i] > 0)
            pos_num++;
        else if(arr[i] < 0)
            neg_num++;
        sum += arr[i];
    }
    int ans;
    if(pos_num == 0)//全小于等于0
        return 0;
    if(neg_num == 0)//全大于等于0
        ans = long(sum) * k % MOD;
    else if(sum <= 0){
        vector<int> cur = arr;
        cur.insert(cur.end(), arr.begin(), arr.end());//拼接一次
        getMax(cur, ans);
    }
    else{
        ans = long(sum) * (k-2) % MOD;
        int left = 0, l_tmp = 0;
        for(int i = len-1; i >= 0; i--){
            l_tmp += arr[i];
            left = max(left, l_tmp);
        }
        int right = 0, r_tmp = 0;
        for(int i = 0; i < len; i++){
            r_tmp += arr[i];
            right = max(right, r_tmp);
        }
        ans = (ans + left + right) % MOD;
    }
    return ans;
}
```

```
};
```

1192. 查找集群内的「关键连接」（困难）

1. 题目描述

力扣数据中心有 `n` 台服务器，分别按从 `0` 到 `n-1` 的方式进行了编号。

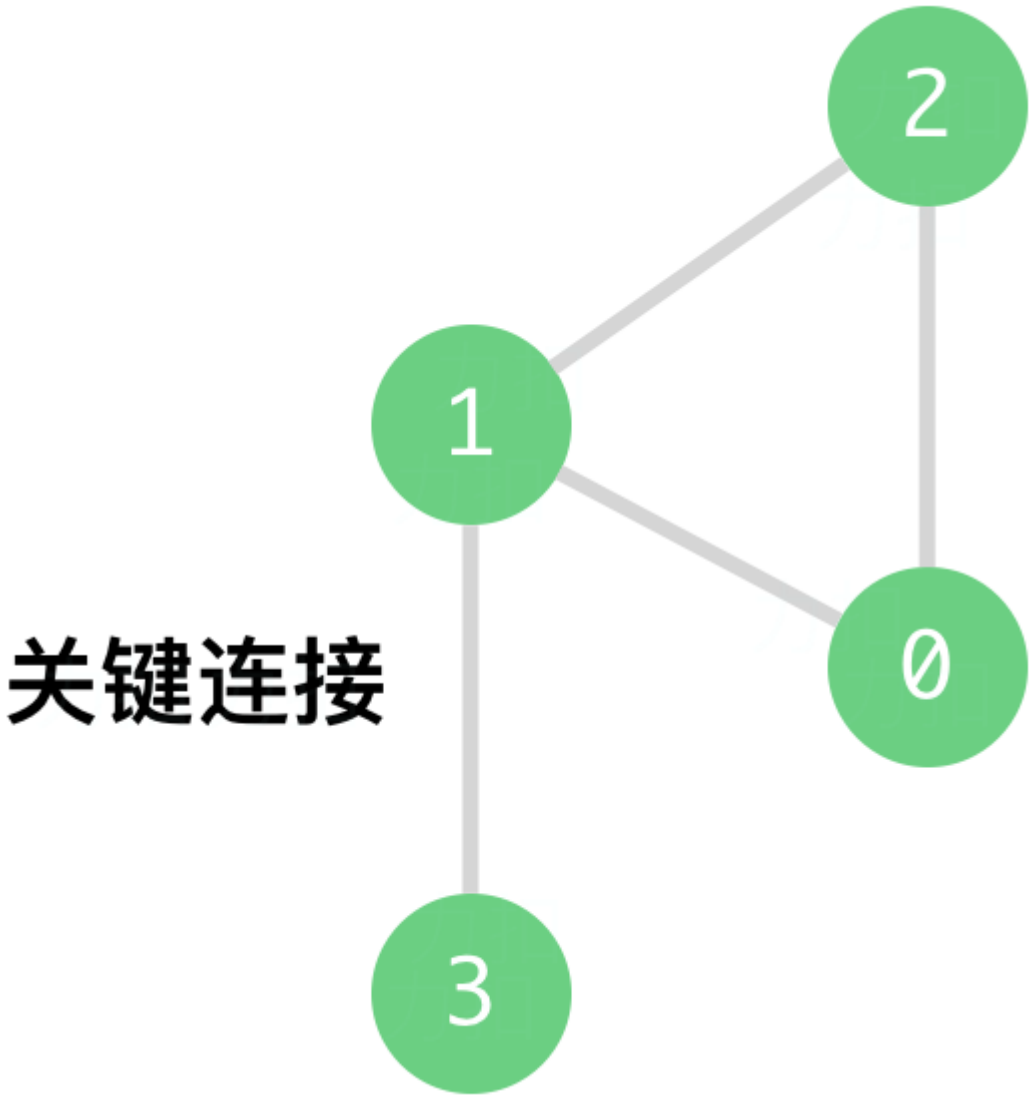
它们之间以「服务器到服务器」点对点的形式相互连接组成了一个内部集群，其中连接 `connections` 是无向的。

从形式上讲，`connections[i] = [a, b]` 表示服务器 `a` 和 `b` 之间形成连接。任何服务器都可以直接或者间接地通过网络到达任何其他服务器。

「关键连接」是在该集群中的重要连接，也就是说，假如我们将它移除，便会导致某些服务器无法访问其他服务器。

请你以任意顺序返回该集群内的所有 「关键连接」。

示例 1：



```
输入: n = 4, connections = [[0,1],[1,2],[2,0],[1,3]]
输出: [[1,3]]
解释: [[3,1]] 也是正确的。
```

提示：

- `1 <= n <= 10^5`
- `n-1 <= connections.length <= 10^5`
- `connections[i][0] != connections[i][1]`
- 不存在重复的连接

2. 正确解法——tarjan算法

解释见https://www.cnblogs.com/nullzx/p/7968110.html

```
class Solution {
public:
    vector<vector<int>> edgs;
    vector<int> dfn;
    vector<int> low;
    vector<bool> visit;
    vector<vector<int>> ret;
    int times;

    void tarjan(int node, int parent){
        times++;
        dfn[node] = times;
        low[node] = times;
        visit[node] = true;

        for (auto e : edgs[node]){
            if (e == parent) continue;
            if (!visit[e]){
                tarjan(e, node);
                low[node] = min(low[e], low[node]);

                if(low[e] > dfn[node]){
                    ret.push_back({node, e});
                }
            }else{
                low[node] = min(low[node], dfn[e]);
            }

        }
    }

    vector<vector<int>> criticalConnections(int n, vector<vector<int>>&
connections) {
        dfn = vector<int>(n,0);
        low = vector<int>(n,0);
        visit = vector<bool>(n,false);
        edgs = vector<vector<int>>(n);
        times = 0;
```

```
        for(auto & conn : connections){
            edgs[conn[0]].push_back(conn[1]);
            edgs[conn[1]].push_back(conn[0]);
        }

        tarjan(0,-1);
        return ret;
    }
};
```