

题目描述

小明正在规划一个大型 **数据中心** 机房，为了使得机柜上的机器都能正常满负荷工作，需要确保在每个机柜边上至少要有有一个电箱。

为了简化题目，假设这个机房是一整排，M表示机柜，I表示间隔，请你返回这整排机柜，至少需要多少个电箱。如果无解请返回 -1。

输入描述

无

输出描述

无

用例

输入	MIIM
输出	2
说明	无

题目解析

本题，“无解”的情况应该是指：M的左右两边都是M，或者边界，比如：

- **MMIIM**：标红M的左边是边界，所以无法放电箱，右边是M，所以也无法放电箱
- **IMMMI**：标红M的左右两边都是M，因此无法放电箱

因此，遇到上面情况，我们应该返回-1。

接下来就是最少电箱的分配策略了，本题要求每台机柜的两边，至少有一边上配有电箱，那么最少电箱分配，肯定是尽可能地让两个机柜共享一个电箱。

我这里利用了区间交集的思想：

即，如果机柜处于 i 位置，则该机柜的区间为 $[i-1, i+1]$ ，需要注意的是 $i-1$ 和 $i+1$ 不能超过边界，如果超过边界，则取 **边界值**。

如果后面一个机柜的区间和前面一个机柜有交集，则代表两个机柜可以共享一个电箱。

这里，我使用栈结构来实现电箱数的统计，即将机柜区间尝试压入栈中，如果要压入的机柜区间 $range$ 和栈顶区间 top 有交集，则弹出栈顶区间 top ，并压入 $range$ 。这样最终栈中区间数，就是电箱数了。

但是上面逻辑存在一个漏洞：那就是 **MIMIM** 这种情况，这这情况栈中只会剩下最后一个M的区间，但是实际上需要的电箱数是2。

这是因为，第2个M和第1个M存在交集后，我们就将第1个M弹出了，因此第2个M不仅代表了自身需要的电箱数，还代表了被弹出的第1个M的电箱数，因此即使第3个M和第2个M有交集，我们也不能将第2个M弹出，因为弹出的话，就会丢失第1个M需要的电箱数。

因此，我额外定义了一个 $stick$ 标识，初始 $stick = false$ ，表示栈顶区间未形成共享电箱

- 如果压入区间A和栈顶区间B存在交集，则将栈顶区间B弹出，并将 $stick=true$ ，表示A区间已形成共享电箱

如果栈顶区间已形成共享电箱，即 $stick=true$ ，则此时，我们可以直接压入新区间，并将 $stick$ 重置为 $false$ 。

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  rl.on("line", (line) => {
10    console.log(getResult(line));
11  });
12
13 function getResult(str) {
```



伏城之外 已关注

👍 0



★ 6



💬 7



专栏目录

已订阅

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  rl.on("line", (line) => {
10    console.log(getResult(line));
11  });
12
13  function getResult(str) {
14    const n = str.length;
15    const stack = [];
16    let stick = false;
17
18    for (let i = 0; i < str.length; i++) {
19      if (str[i] == "M") {
20        // 如果机柜A两边都是机柜, 或者没有间隔, 则无法给机柜A放电箱, 返回-1
21        let left = i - 1 < 0 || str[i - 1] == "M";
22        let right = i + 1 >= n || str[i + 1] == "M";
23        if (left && right) return -1;
24
25        // 将求解最少电箱问题, 转化为区间交集问题
26        const range = [Math.max(0, i - 1), Math.min(n - 1, i + 1)];
27
28        if (stack.length && !stick) {
29          const e1 = stack.at(-1)[1];
30          const s2 = range[0];
31
32          if (e1 === s2) {
33            stack.pop();
34            stick = true;
35          }
36        } else {
37          stick = false;
38        }
39        stack.push(range);
40      }
41    }
42
43    return stack.length;
44  }
```

Java算法源码

```
1  import java.util.LinkedList;
2  import java.util.Scanner;
3
4  public class Main {
5    public static void main(String[] args) {
6      Scanner sc = new Scanner(System.in);
7
8      String str = sc.next();
9
10     System.out.println(getResult(str));
11   }
12
13   public static int getResult(String str) {
14     int n = str.length();
15     LinkedList<Integer> stack = new LinkedList<>();
16     boolean stick = false;
17
18     for (int i = 0; i < n; i++) {
19       if (str.charAt(i) == 'M') {
20         // 如果机柜A两边都是机柜, 或者没有间隔, 则无法给机柜A放电箱, 返回-1
21         boolean left = i - 1 < 0 || str.charAt(i - 1) == 'M';
22         boolean right = i + 1 >= n || str.charAt(i + 1) == 'M';
23         if (left && right) return -1;
24
25         // 将求解最少电箱问题, 转化为区间交集问题
26         Integer[] range = {Math.max(0, i - 1), Math.min(n - 1, i + 1)};
27
28         if (stack.size() > 0 && !stick) {
29           int e1 = stack.getLast()[1];
30           int s2 = range[0];
31
32           if (e1 == s2) {
33             stack.removeLast();
34             stick = true;
35           }
36         } else {
37           stick = false;
38         }
39         stack.addLast(range);
40       }
41     }
42
43     return stack.size();
44   }
```


Java算法源码

```
1 import java.util.LinkedList;
2 import java.util.Scanner;
3
4 public class Main {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7
8         String str = sc.next();
9
10        System.out.println(getResult(str));
11    }
12
13    public static int getResult(String str) {
14        int n = str.length();
15        LinkedList<Integer> stack = new LinkedList<>();
16        boolean stick = false;
17
18        for (int i = 0; i < n; i++) {
19            if (str.charAt(i) == 'M') {
20                // // 如果机柜A两边都是机柜，或者没有间隔，则无法给机柜A放电箱，返回-1
21                boolean left = i - 1 < 0 || str.charAt(i - 1) == 'M';
22                boolean right = i + 1 >= n || str.charAt(i + 1) == 'M';
23                if (left && right) return -1;
24
25                // 将求解最少电箱问题，转化为区间交集问题
26                Integer[] range = {Math.max(0, i - 1), Math.min(n - 1, i + 1)};
27
28                if (stack.size() > 0 && !stick) {
29                    int e1 = stack.getLast()[1];
30                    int s2 = range[0];
31
32                    if (e1 == s2) {
33                        stack.removeLast();
34                        stick = true;
35                    }
36                } else {
37                    stick = false;
38                }
39                stack.addLast(range);
40            }
41        }
42
43        return stack.size();
44    }
45 }
```

Python算法源码

```
1 # 输入获取
2 s = input()
3
4
5 # 算法入口
6 def getResult(s):
7     n = len(s)
8     stack = []
9     stick = False
10
11     for i in range(n):
12         if s[i] == 'M':
13             # 如果机柜A两边都是机柜，或者没有间隔，则无法给机柜A放电箱，返回-1
14             left = i - 1 < 0 or s[i - 1] == 'M'
15             right = i + 1 >= n or s[i + 1] == 'M'
16
17             if left and right:
18                 return -1
19
20             # 将求解最少电箱问题，转化为区间交集问题
21             ran = [max(0, i - 1), min(n - 1, i + 1)]
22
23             if len(stack) > 0 and not stick:
24                 e1 = stack[-1][1]
25                 s2 = ran[0]
26
27                 if e1 == s2:
28                     stack.pop()
29                     stick = True
30             else:
31                 stick = False
32
33             stack.append(ran)
34
35     return len(stack)
36
37
38 print(getResult(s))
```

墨 文章知识点与官方知识档案匹配，可进一步学习相关知识

 伏城之外 已关注

👍 0 🗨 0 ⭐ 6 📁 0 💬 7 📧 0

专栏目录 已订阅