题目描述

- 请实现一个简易内存池,根据请求命令完成内存分配和释放。
- 内存池支持两种操作命令，REQUEST和RELEASE，其格式为：
- REQUEST=请求的内存大小 表示请求分配指定大小内存，如果分配成功，返回分配到的内存首地址；如果内存不足，或指定的大小为0，则输出error。
- RELEASE=释放的内存首地址 表示释放掉之前分配的内存，释放成功无需输出，如果释放不存在的首地址则输出error。

注意：

1. 内存池总大小为100字节。
2. 内存池地址分配必须是连续内存，并优先从低地址分配。
3. 内存释放后可被再次分配，已释放的内存在空闲时不能被二次释放。
4. 不会释放已申请的内存块的中间地址。
5. 释放操作只是针对首地址所对应的单个内存块进行操作，不会影响其它内存块。

输入描述

首行为整数 N , 表示操作命令的个数，取值范围：0 < N <= 100。

接下来的N行, 每行将给出一个操作命令，操作命令和参数之间用 “=”分割。

输出描述

请求分配指定大小内存时，如果分配成功，返回分配到的内存首地址；如果内存不足，或指定的大小为0，则输出error

释放掉之前分配的内存时，释放成功无需输出，如果释放不存在的首地址则输出error。

用例

输入	2 REQUEST=10 REQUEST=20
输出	0 10
说明	无

输入	5 REQUEST=10 REQUEST=20 RELEASE=0 REQUEST=20 REQUEST=10
输出	0 10 30 0
说明	第一条指令，申请地址0~9的10个字节内存，返回首地址0 第二条指令，申请地址10~29的20字节内存，返回首地址10 第三条指令，释放首地址为0的内存申请，0~9地址内存被释放，变为空闲，释放成功，无需输出 第四条指令，申请20字节内存，09地址内存连续空间不足20字节，往后查找到3049地址，返回首地址30 第五条指令，申请10字节，0~9地址内存空间足够，返回首地址0

题目解析

我的解题思路如下：

定义一个used数组，用来存储已被占用的内存区间，即[起始位置，结束位置]。

初始化给used数组一个 [100,101]，表示存在一个已占有内存区间[100,101]，这个内存区间将作为尾边界使用。

当REQUEST申请size大小的内存时，我们从start=0位置开始申请，即申请[start, start+size-1]区间，接下来看该区间是否和used[i]区间存在交叉，如果存在交xian叉，则说明申请的内存区间中部分内存已被使用，因此我们应该更新 start = used[i][1] + 1位置，重新申请一个区间，这样就必然不和used[i]区间交叉了，但是要继续和used[i+1]区间比较。

直到找到一个不存在交叉的内存区间，打印此时的start，并将申请到的内存区间插入到used数组中，注意插入位置是 i 。

如果一直都找不到不存在交叉的内存区间，则打印error。

当RELEASE释放起始位置addr的内存时，我们只需要遍历每一个used[i]，比较used[i][0]和addr是否相同，若相同，则表示找到了要释放的内存，此时只要将used[i]从used中删除即可。

如果没有找到，则打印error。

JavaScript算法源码

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let n;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    n = lines[0] - 0;
  }

  if (n && lines.length === n + 1) {
    lines.shift();
    getResult(lines.map((line) => line.split("=")));
    lines.length = 0;
  }
});

function getResult(commands) {
  // used保存被占用的内存 [起始地址，结束地址]，初始时有一个[100,101]作为左边界限定
  const used = [[100, 101]];

  for (let [key, val] of commands) {
    // 申请内存
    if (key === "REQUEST") {
      // 当指令为REQUEST时，对应值为要申请的内存的大小，即size
      const size = val - 0;

      // 我们默认从start=0位置开始检查可用内存区间
      let start = 0;
      let flag = true;

```

```
      for (let i = 0; i < used.length; i++) {
        let end = start + size - 1;
        // 要申请的内存区间
        const range = [start, end];
        // 检查要申请的内存区间和已占有的内存区间是否交叉
        if (!hasIntersection(used[i], range)) {
          // 若不存在交叉，则将申请区间加入used中
          used.splice(i, 0, range);
          flag = false;
          // 并打印此时申请区间的起始位置
          console.log(start);
          break;
        } else {
          // 若存在交叉，则将发更要申请的内存区间的起始位置
          start = used[i][1] + 1;
        }
      }

      // 一旦申请到内存，那么flag就会被赋值为false，否则就保持true，意味着每申请到内存，则打印error
      if (flag) console.log("error");
    }
    // 释放内存
    else {
      //  当指令为RELEASE时，值为要释放内存的起始地址addr
      const addr = val - 0;
      let flag = true;

      for (let i = 0; i < used.length; i++) {
```

```
      for (let i = 0; i < used.length; i++) {
        // 到已占有内存中找起始位置是addr的，找到则将该区间从used中删除，表示解除占用
        if (used[i][0] === addr) {
          used.splice(i, 1);
          flag = false;
          break;
        }
      }

      // 一旦释放成功，则flag就会被置为false，否则就保持true,意味着没有内存释放，则打印error
      if (flag) console.log("error");
    }
  }
}

// 判断两个区间是否存在交集
function hasIntersection(area1, area2) {
  const [s1, e1] = area1;
  const [s2, e2] = area2;

  if (s1 === s2) return true;
  else if (s1 < s2) return e1 >= s2;
  else return e2 >= s1;
}
```

Java算法源码

```java
import java.util.LinkedList;
import java.util.Scanner;

public class Main {
  // 输入获取
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();

    String[][] cmds = new String[n][2];
    for (int i = 0; i < n; i++) cmds[i] = sc.next().split("=");

    getResult(n, cmds);
  }

  // 算法入口
  public static void getResult(int n, String[][] cmds) {
    // used保存被占用的内存 [起始地址，结束地址]，初始时有一个[100,101]作为笔边界限定
    LinkedList<Integer[]> used = new LinkedList<>();
    used.add(new Integer[] {100, 101});

    for (String[] cmd : cmds) {
      String key = cmd[0];
      String val = cmd[1];

      // 申请内存
      if ("REQUEST".equals(key)) {
        // 当指令为REQUEST时，对应值为要申请的内存的大小，即size
        int size = Integer.parseInt(val);

        // 我们默认从start=0位置开始检查可用内存区间
        int start = 0;
        boolean flag = true;

        for (int i = 0; i < used.size(); i++) {
          int end = start + size - 1;
          // 要申请的内存区间
          Integer[] range = {start, end};

```

```
41          // 检查要申请的内存区间和已占有的内存区间是否交叉
42          if (!hasIntersection(used.get(i), range)) {
43            // 若不存在交叉，则将申请区间加入used中
44            used.add(i, range);
45            flag = false;
46            // 并打印此时申请区间的起始位置
47            System.out.println(start);
48            break;
49          } else {
50            // 若存在交叉，则将变更要申请的内存区间的起始位置
51            start = used.get(i)[1] + 1;
52          }
53        }
54
55        // 一旦申请到内存，那么flag就会被赋值为false，否则就保持true，意味着每申请到内存，则打印error
56        if (flag) System.out.println("error");
57      }
58      // 释放内存
59      else {
60        //  当指令为RELEASE时，值为要释放内存的起始地址addr
61        int addr = Integer.parseInt(val);
62        boolean flag = true;
63
64        for (int i = 0; i < used.size(); i++) {
65          // 到已占有内存中找起始位置是addr的，找到则将该区间从used中删除，表示解除占用
66          if (used.get(i)[0] == addr) {
67            used.remove(i);
68            flag = false;
69            break;
70          }
71        }
72
```

```
73        // 一旦释放成功，则flag就会被置为false，否则就保持true,意味着没有内存释放，则打印error
74        if (flag) System.out.println("error");
75      }
76    }
77  }
78
79  // 判断两个区间是否存在交集
80  public static boolean hasIntersection(Integer[] range1, Integer[] range2) {
81    int s1 = range1[0];
82    int e1 = range1[1];
83
84    int s2 = range2[0];
85    int e2 = range2[1];
86
87    if (s1 == s2) return true;
88    else if (s1 < s2) return e1 >= s2;
89    else return e2 >= s1;
90  }
91 }
```

Python算法源码

```python
# 输入获取
n = int(input())
cmds = [input().split("=") for _ in range(n)]


# 判断两个区间是否存在交集
def hasIntersection(a1, a2):
    s1, e1 = a1
    s2, e2 = a2

    if s1 == s2:
        return True
    elif s1 < s2:
        return e1 >= s2
    else:
        return e2 >= s1


# 算法入口
def getResult():
    # used缓存被占用的内存 [起始地址，结束地址]，初始时有一个[100,101]作为毫边界限定
    used = [[100, 101]]

    for key, val in cmds:
        # 申请内存
        if key == "REQUEST":
            # 当指令为REQUEST时，对应值为要申请的内存的大小，即size
            size = int(val)

            #  我们默认从start=0位置开始检查可用内存区间
            start = 0
            flag = True

            for i in range(len(used)):
                end = start + size - 1

                # 要申请的内存区间
                ran = [start, end]

```

```
                # 检查要申请的内存区间和已占有的内存区间是否交叉
                if not hasIntersection(used[i], ran):
                    # 若不存在交叉，则将申请区间加入used中
                    used.insert(i, ran)
                    flag = False
                    # 并打印此时申请区间的起始位置
                    print(start)
                    break
                else:
                    #  若存在交叉，则将发现要申请的内存区间的起始位置
                    start = used[i][1] + 1

            # 一旦申请到内存，那么flag就会被赋值为false，否则就保持true，意味着申请到内存，则打印error
            if flag:
                print("error")
        # 释放内存
        else:
            # 当指令为RELEASE时，值为要释放内存的起始地址addr
            addr = int(val)
            flag = True

            for i in range(len(used)):
                # 到已占有内存中找起始位置是addr的，找到则将该区间从used中删除，表示解除占用
                if used[i][0] == addr:
                    used.pop(i)
                    flag = False
                    break

            # 一旦释放成功，则flag就会被置为false，否则就保持True,意味着没有内存释放，则打印error
            if flag:
                print("error")


# 算法调用
getResult()
```