

【华为OD机考 统一考试机试C卷】堆内存申请（C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，**C卷真题已基本整理完毕**

抽到原题的概率为2/3到3/3，**也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。**

另外订阅专栏还可以联系笔者开通在线 **OJ** 进行刷题，提高刷题效率。

真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明

专栏：2023华为OD机试(B卷+C卷+D卷) (C++JavaJSPy)

华为OD面试真题精选：华为OD面试真题精选

在线OJ：点击立即刷题，模拟真实机考环境

题目描述

有一个总空间为100字节的堆，现要从中新申请一块内存，内存分配原则为：优先紧接着前一块已使用内存，分配空间足够且最接近申请大小的空闲内存。

输入描述

第1行是1个整数，表示期望申请的内存字节数

第2到第N行是用空格分割的两个整数，表示当前已分配的内存的情况，每一行表示一块已分配的连续内存空间，每行的第1和第2个整数分别表示偏移地址和内存块大小，如：

0 1

3 2

表示 0 偏移地址开始的 1 个字节和 3 偏移地址开始的 2 个字节已被分配，其余内存空闲。

输出描述

若申请成功，输出申请到内存的偏移；

若申请失败，输出 -1

备注：

- 1. 若输入信息不合法或无效，则申请失败
- 2. 若没有足够的空间供分配，则申请失败
- 3. 堆内存信息有区域重叠或有非法值等都是无效输入

用例

输入	1 0 1 3 2
输出	1
说明	堆中已使用的两块内存是偏移从0开始的1字节和偏移从3开始的2字节，空闲的两块内存是偏移从1开始2个字节和偏移从5开始95字节，根据分配原

解题思路

1. 内存块排序:

- 将 `usedMemory` 列表中的内存块按起始地址进行排序。这样可以更容易地找到连续的空闲内存区域。

2. 寻找最佳匹配内存块:

- 程序初始化一个变量 `start`（起始地址）为0， `bestFitStart`（最佳匹配起始地址）为-1，以及 `minSizeDiff`（最小大小差异）为最大整数值。

然后，它遍历已排序的内存块列表。对于每个内存块，程序执行以下操作：

- 检查内存块的合法性（起始地址是否有效，大小是否合理）。
- 计算当前可用的空闲空间（ `freeSpace` ）。
- 如果这个空闲空间足够大并且比之前找到的空间更接近申请大小，则更新 `bestFitStart` 和 `minSizeDiff`。

3. 检查最后的空闲空间:

- 程序还需要检查列表中最后一个内存块后面的空间。如果那里有足够的空间，且这个空间的大小比之前找到的任何空间更接近申请大小，则更新 `bestFitStart`。

C++

```
1 | #include <iostream>
2 | #include <vector>
~ |
```

```

3  #include <algorithm>
4  #include <limits>
5
6  using namespace std;
7
8  int main() {
9      int mallocSize; // 需要分配的内存大小
10     cin >> mallocSize; // 从标准输入读取内存大小
11
12     // 用于存储已被使用的内存块的向量, 每个内存块由一对整数表示 (起始地址, 大小)
13     vector<pair<int, int> > usedMemory;
14
15     int start, size;
16     while (cin >> start >> size) { // 循环读取已使用内存块的起始地址和大小
17         usedMemory.push_back(make_pair(start, size)); // 将读取的内存块添加到向量中
18     }
19
20     // 如果分配的内存大小不合理, 则输出-1并结束程序
21     if (mallocSize <= 0 || mallocSize > 100) {
22         cout << -1 << endl;
23         return 0;
24     }
25
26     // 对已使用的内存块按起始地址进行排序
27     sort(usedMemory.begin(), usedMemory.end());
28
29     start = 0; // 初始化用于搜索空闲内存的起始地址
30     int bestFitStart = -1; // 存储最佳匹配的内存块起始地址
31     int minSizeDiff = numeric_limits<int>::max(); // 最小大小差异, 初始化为int的最大值
32
33     // 遍历所有已使用的内存块
34     for (size_t i = 0; i < usedMemory.size(); ++i) {
35         int blockStart = usedMemory[i].first; // 内存块的起始地址
36         int blockSize = usedMemory[i].second; // 内存块的大小
37
38         // 检查内存块是否有效
39         if (blockStart < start || blockSize <= 0 || blockStart + blockSize > 100) {
40             cout << -1 << endl;
41             return 0;
42         }
43     }

```

```

44
45 // 计算当前内存块和上一个内存块之间的空闲空间
46 int freeSpace = blockStart - start;
47 // 如果找到足够的空闲空间且空间差异比之前找到的更小, 则更新最佳匹配的起始地址和最小大小差异
48 if (mallocSize <= freeSpace && (freeSpace - mallocSize) < minSizeDiff) {
49     bestFitStart = start;
50     minSizeDiff = freeSpace - mallocSize;
51 }
52
53 // 更新搜索的起始地址为当前内存块的结束地址
54 start = blockStart + blockSize;
55 }
56
57 // 检查最后一个内存块后是否有足够的空闲空间
58 if (100 - start >= mallocSize && (100 - start - mallocSize) < minSizeDiff) {
59     bestFitStart = start;
60 }
61
62 // 输出最佳匹配的起始地址
63 cout << bestFitStart << endl;
64
65 return 0;
}

```

Java

```

1 import java.util.ArrayList;
2 import java.util.List;
3 import java.util.Scanner;
4 import java.util.Arrays;
5
6 public class Main {
7     public static void main(String[] args) {
8         // 创建一个扫描器来读取用户输入
9         Scanner sc = new Scanner(System.in);
10
11         // 读取第一行输入, 这是我们要分配的内存大小
12         int mallocSize = Integer.parseInt(sc.nextLine());
13         // 创建一个列表来存储已使用的内存块
14         List<int[]> usedMemory = new ArrayList<>();
15

```

```

15
16 // 循环读取后续的输入行，每行代表一个已使用的内存块
17 while (sc.hasNextLine()) {
18     String line = sc.nextLine();
19     // 如果读取到空行，结束输入
20     if (line.isEmpty()) {
21         break;
22     }
23     // 将输入行分割成字符串数组，然后转换成整数数组
24     int[] memoryBlock = Arrays.stream(line.split(" "))
25                               .mapToInt(Integer::parseInt)
26                               .toArray();
27     // 将这个内存块添加到已使用的内存列表中
28     usedMemory.add(memoryBlock);
29 }
30
31 // 如果要分配的内存大小不在合法范围内，输出-1并结束程序
32 if (mallocSize <= 0 || mallocSize > 100) {
33     System.out.println(-1);
34     return;
35 }
36
37 // 按照内存块的起始地址对已使用的内存列表进行排序
38 usedMemory.sort((a, b) -> a[0] - b[0]);
39
40 // 初始化起始地址为0
41 int start = 0;
42 // 初始化最佳适配的起始地址为-1
43 int bestFitStart = -1;
44 // 初始化最小大小差为最大整数
45 int minSizeDiff = Integer.MAX_VALUE;
46
47 // 遍历已使用的内存列表
48 for (int[] block : usedMemory) {
49     // 获取内存块的起始地址和大小
50     int blockStart = block[0];
51     int blockSize = block[1];
52
53     // 如果内存块的起始地址小于当前的起始地址，或者内存块的大小小于等于0，或者内存块的结束地址大于100，输出-1并结束程序
54     if (blockStart < start || blockSize <= 0 || blockStart + blockSize > 100) {
55

```

```

56         System.out.println(-1);
57         return;
58     }
59
60     // 计算当前的起始地址和内存块的起始地址之间的空闲空间
61     int freeSpace = blockStart - start;
62     // 如果空闲空间大于等于要分配的内存大小, 并且空闲空间和要分配的内存大小的差小于当前的最小大小差, 更新最佳适配的起始地址和最小大小差
63     if (mallocSize <= freeSpace && (freeSpace - mallocSize) < minSizeDiff) {
64         bestFitStart = start;
65         minSizeDiff = freeSpace - mallocSize;
66     }
67
68     // 更新当前的起始地址为内存块的结束地址
69     start = blockStart + blockSize;
70 }
71
72 // 检查最后一个内存块之后的空闲空间, 如果空闲空间大于等于要分配的内存大小, 并且空闲空间和要分配的内存大小的差小于当前的最小大小差, 更新最佳适配的起始地址
73 if (100 - start >= mallocSize && (100 - start - mallocSize) < minSizeDiff) {
74     bestFitStart = start;
75 }
76
77 // 输出最佳适配的起始地址
78 System.out.println(bestFitStart);
79 }
}

```

javaScript

```

1 // 引入readLine模块以从标准输入读取数据
2 const readline = require('readline');
3
4 // 创建readLine接口
5 const rl = readline.createInterface({
6     input: process.stdin, // 将标准输入流作为输入源
7     output: process.stdout // 将标准输出流作为输出源
8 });
9
10 let mallocSize; // 需要分配的内存大小
11 let usedMemory = []; // 用于存储已使用内存块的数组
12
13

```

```

13 // 当新的行被接收到时触发
14 rl.on('line', (line) => {
15   if (!mallocSize) { // 如果还未读取到mallocSize
16     mallocSize = parseInt(line); // 解析并设置需要分配的内存大小
17     // 如果内存大小不合理, 则输出-1并结束程序
18     if (mallocSize <= 0 || mallocSize > 100) {
19       console.log(-1);
20       process.exit(0);
21     }
22   } else { // 如果已读取到mallocSize, 则读取内存块
23     const memoryBlock = line.split(' ').map(Number); // 将行分割并转换为数字数组
24     usedMemory.push(memoryBlock); // 将内存块添加到usedMemory数组中
25   }
26 });
27
28 // 当输入流被关闭时触发
29 rl.on('close', () => {
30   // 对已使用的内存块按起始地址进行排序
31   usedMemory.sort((a, b) => a[0] - b[0]);
32
33   let start = 0; // 初始化用于搜索空闲内存的起始地址
34   let bestFitStart = -1; // 存储最佳匹配的内存块起始地址
35   let minSizeDiff = Number.MAX_SAFE_INTEGER; // 最小大小差异, 初始化为最大安全整数值
36
37   // 遍历所有已使用的内存块
38   for (let block of usedMemory) {
39     let blockStart = block[0]; // 内存块的起始地址
40     let blockSize = block[1]; // 内存块的大小
41
42     // 检查内存块是否有效
43     if (blockStart < start || blockSize <= 0 || blockStart + blockSize > 100) {
44       console.log(-1);
45       process.exit(0);
46     }
47
48     // 计算当前内存块和上一个内存块之间的空闲空间
49     let freeSpace = blockStart - start;
50     // 如果找到足够的空闲空间且空间差异比之前找到的更小, 则更新最佳匹配的起始地址和最小大小差异
51     if (mallocSize <= freeSpace && (freeSpace - mallocSize) < minSizeDiff) {
52       bestFitStart = start;
53     }

```

```

54     minSizeDiff = freeSpace - mallocSize;
55 }
56
57 // 更新搜索的起始地址为当前内存块的结束地址
58 start = blockStart + blockSize;
59 }
60
61 // 检查最后一个内存块后是否有足够的空闲空间
62 if (100 - start >= mallocSize && (100 - start - mallocSize) < minSizeDiff) {
63     bestFitStart = start;
64 }
65
66 // 输出最佳匹配的起始地址
67 console.log(bestFitStart);
});

```

Python

```

1  import sys
2
3  # 读取第一行输入, 这是我们要分配的内存大小
4  mallocSize = int(sys.stdin.readline())
5  # 创建一个列表来存储已使用的内存块
6  usedMemory = []
7
8  # 循环读取后续的输入行, 每行代表一个已使用的内存块
9  for line in sys.stdin:
10     # 将输入行分割成字符串数组, 然后转换成整数数组
11     memoryBlock = list(map(int, line.split()))
12     # 将这个内存块添加到已使用的内存列表中
13     usedMemory.append(memoryBlock)
14
15 # 如果要分配的内存大小不在合法范围内, 输出-1并结束程序
16 if mallocSize <= 0 or mallocSize > 100:
17     print(-1)
18     sys.exit(0)
19
20 # 按照内存块的起始地址对已使用的内存列表进行排序
21 usedMemory.sort(key=lambda x: x[0])
22
23

```



```

43 # 初始化起始地址为0
24 start = 0
25 # 初始化最佳适配的起始地址为-1
26 bestFitStart = -1
27 # 初始化最小大小差为最大整数
28 minSizeDiff = float('inf')
29
30 # 遍历已使用的内存列表
31 for block in usedMemory:
32     # 获取内存块的起始地址和大小
33     blockStart, blockSize = block
34
35     # 如果内存块的起始地址小于当前的起始地址, 或者内存块的大小小于等于0, 或者内存块的结束地址大于100, 输出-1并结束程序
36     if blockStart < start or blockSize <= 0 or blockStart + blockSize > 100:
37         print(-1)
38         sys.exit(0)
39
40     # 计算当前的起始地址和内存块的起始地址之间的空闲空间
41     freeSpace = blockStart - start
42     # 如果空闲空间大于等于要分配的内存大小, 并且空闲空间和要分配的内存大小的差小于当前的最小大小差, 更新最佳适配的起始地址和最小大小差
43     if mallocSize <= freeSpace and (freeSpace - mallocSize) < minSizeDiff:
44         bestFitStart = start
45         minSizeDiff = freeSpace - mallocSize
46
47     # 更新当前的起始地址为内存块的结束地址
48     start = blockStart + blockSize
49
50 # 检查最后一个内存块之后的空闲空间, 如果空闲空间大于等于要分配的内存大小, 并且空闲空间和要分配的内存大小的差小于当前的最小大小差, 更新最佳适配的起始地址
51 if 100 - start >= mallocSize and (100 - start - mallocSize) < minSizeDiff:
52     bestFitStart = start
53
54 # 输出最佳适配的起始地址
55 print(bestFitStart)

```

C语言

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // 定义一个结构体, 用于表示内存块
5

```

```

5 typedef struct {
6     int start; // 内存块的起始地址
7     int size;  // 内存块的大小
8 } MemoryBlock;
9
10 int compareMemoryBlocks(const void *a, const void *b) {
11     // 用于qsort的比较函数, 按内存块的起始地址排序
12     MemoryBlock *blockA = (MemoryBlock *)a;
13     MemoryBlock *blockB = (MemoryBlock *)b;
14     return blockA->start - blockB->start;
15 }
16
17 int main() {
18     int mallocSize; // 需要分配的内存大小
19     scanf("%d", &mallocSize); // 从标准输入读取内存大小
20
21     MemoryBlock usedMemory[100]; // 存储已分配内存块的数组
22     int count = 0; // 已分配内存块的数量
23
24     int start, size;
25     while (scanf("%d %d", &start, &size) == 2) {
26         // 循环读取已分配内存块的起始地址和大小
27         usedMemory[count].start = start;
28         usedMemory[count].size = size;
29         count++;
30     }
31
32     // 如果分配的内存大小不合理, 则输出-1并结束程序
33     if (mallocSize <= 0 || mallocSize > 100) {
34         printf("-1\n");
35         return 0;
36     }
37
38     // 对已使用的内存块按起始地址进行排序
39     qsort(usedMemory, count, sizeof(MemoryBlock), compareMemoryBlocks);
40
41     int bestFitStart = -1; // 存储最佳匹配的内存块起始地址
42     int minSizeDiff = 101; // 最小大小差异, 初始化为一个大于最大内存的值
43
44     start = 0; // 初始化用于搜索空闲内存的起始地址
45

```

```

46     for (int i = 0; i < count; i++) {
47         // 遍历所有已使用的内存块
48         int blockStart = usedMemory[i].start;
49         int blockSize = usedMemory[i].size;
50
51         // 检查内存块是否有效
52         if (blockStart < start || blockSize <= 0 || blockStart + blockSize > 100) {
53             printf("-1\n");
54             return 0;
55         }
56
57         // 计算当前内存块和上一个内存块之间的空闲空间
58         int freeSpace = blockStart - start;
59         if (mallocSize <= freeSpace && (freeSpace - mallocSize) < minSizeDiff) {
60             // 如果找到足够的空闲空间且空间差异比之前找到的更小
61             bestFitStart = start;
62             minSizeDiff = freeSpace - mallocSize;
63         }
64
65         // 更新搜索的起始地址为当前内存块的结束地址
66         start = blockStart + blockSize;
67     }
68
69     // 检查最后一个内存块后是否有足够的空闲空间
70     if (100 - start >= mallocSize && (100 - start - mallocSize) < minSizeDiff) {
71         bestFitStart = start;
72     }
73
74     // 输出最佳匹配的起始地址
75     printf("%d\n", bestFitStart);
76
77     return 0;
}

```

文章目录

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

[题目描述](#)

[输入描述](#)

输出描述

用例

解题思路

C++

Java

javaScript

Python

C语言

机考真题 华为OD



CSDN @算法大师