62、内存资源分配 II， 考点 or 实现——二分查找

题目描述

有一个简易 内存池，内存按照大小粒度分类，每个粒度有若干个可用内存资源，用户会进行一系列内存申请，需要按需分配内存池中的资源返回申请结果成功失败列表。

分配规则如下：

1. 分配的内存要大于等于内存的申请量，存在满足需求的内存就必须分配，优先分配粒度小的，但内存不能拆分使用；
2. 需要按申请顺序分配，先申请的先分配，有可用内存分配则申请结果为true；
3. 没有可用则返回false。

注意：不考虑内存释放

输入描述

输入为两行字符串

第一行为内存池资源列表，包含内存粒度数据信息，粒度数据间用逗号分割

- 一个粒度信息内用冒号分割，冒号前为内存粒度大小，冒号后为数量
- 资源列表不大于1024
- 每个粒度的数量不大于4096

第二行为申请列表，申请的内存大小间用逗号分割

- 申请列表不大于100000

如

64:2,128:1,32:4,1:128

50,36,64,128,127

输出描述

输出为内存池分配结果

如true,true,true,false,false

用例

输入	64:2,128:1,32:4,1:128 50,36,64,128,127
输出	true,true,true,false,false
说明	内存池资源包含：64K共2个、128K共1个、32K共4个、1K共128个的内存资源； 针对50,36,64,128,127的内存申请序列，分配的内存依次是：64,64,128,NULL,NULL, 第三次申请内存时已经将128分配出去，因此输出结果是：true,true,true,false,false

题目解析

本题最简单的方法是用一个双重for，外层循环申请列表，内层循环内存池资源列表（先按内存大小升序）

```
/**
 * @param {*} pools 内存池资源列表，是一个二维数组，元素是[内存粒度大小， 内存粒度数量]
 * @param {*} applies 申请列表
 * @returns 内存池分配结果
 */
function getResult(pools, applies) {
  pools.sort((a, b) => a[0] - b[0]);

  const ans = [];
  outer: for (let apply of applies) {
    for (let pool of pools) {
      const [size, count] = pool;

      // 如果该内存池资源数为0，则继续查找下一个内存池资源
      if (count <= 0) continue;
      // 如果该内存池资源大小不够申请的内存，则继续查找下一个内存池资源
      if (size < apply) continue;

      // 如果该内存池有资源，且大小满足申请的内存，则可以使用
      pool[1]--;
      ans.push(true);
      continue outer;
    }

    ans.push(false);
  }

  return ans.join(",");
}
```

但是，本题中有数量级说明，其中：

- 资源列表m不大于1024
- 申请列表n不大于100000

也就是说，最多有1024 * 100000次循环，这是超时的节奏。

因此，上面算法要进行优化，由于申请列表有顺序要求：

> 需要按申请顺序分配，先申请的先分配，有可用内存分配则申请结果为true；

因此，我们不能改变申请列表顺序。

而资源列表没有顺序要求，因此我们可以将资源列表升序，然后每次申请一个内存时，都在资源列表中进行二分查找，这样即使资源列表有1024长度，最多也只需要10次即可查询到合适的内存池资源，此时整体时间复杂度为NlogM，循环次数降低到了 最大百万级别。

JavaScript算法源码

```javascript
/* 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 = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const pools = lines[0]
      .split(",")
      .map((line) => line.split(":").map(Number));

    const applies = lines[1].split(",").map(Number);

    console.log(getResult(pools, applies));
    lines.length = 0;
  }
});

/**
 * @param {*} pools 内存池资源列表，是一个二维数组，元素是[内存粒度大小， 内存粒度数量]
 * @param {*} applies 申请列表
 * @returns 内存池分配结果
```

```
 */
function getResult(pools, applies) {
  pools.sort((a, b) => a[0] - b[0]);

  const ans = [];

  for (let apply of applies) {
    let idx = binarySearch(pools, apply);

    if (idx < 0) {
      idx = -idx - 1;
    }

    if (pools[idx] === undefined) {
      ans.push(false);
      continue;
    }

    pools[idx][1]--;

    ans.push(true);

    if (pools[idx][1] === 0) {
      pools.splice(idx, 1);
    }
  }

  return ans.join();
}

function binarySearch(arr, key) {
  let low = 0;
  let high = arr.length - 1;

  while (low <= high) {
    let mid = (low + high) >>> 1;
    let midVal = arr[mid][0]; // 专门用于pools二分查找的binarySearch

    if (midVal < key) {
      low = mid + 1;
    } else if (midVal > key) {
      high = mid - 1;
    } else {
      return mid;
    }
  }
  return -low - 1;
}
```

Java算法源码

```java
import java.util.*;

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

    Integer[][] pools =
        Arrays.stream(sc.nextLine().split(","))
            .map(
                str -> Arrays.stream(str.split(":")).map(Integer::parseInt).toArray(Integer[]::new))
            .toArray(Integer[][]::new);

    Integer[] applies =
        Arrays.stream(sc.nextLine().split(",")).map(Integer::parseInt).toArray(Integer[]::new);

    System.out.println(getResult(pools, applies));
  }

  public static String getResult(Integer[][] pools, Integer[] applies) {
    Integer[][] sortedPools =
        Arrays.stream(pools).sorted((a, b) -> a[0] - b[0]).toArray(Integer[][]::new);

    ArrayList<Integer> sizes = new ArrayList<>();
    ArrayList<Integer> counts = new ArrayList<>();
    for (Integer[] pool : sortedPools) {
      sizes.add(pool[0]);
      counts.add(pool[1]);
    }

    ArrayList<Boolean> ans = new ArrayList<>();

    for (Integer apply : applies) {
      int idx = Collections.binarySearch(sizes, apply);

      if (idx < 0) {
        idx = -idx - 1;
      }

      if (idx >= sizes.size()) {
        ans.add(false);
        continue;
      }

      counts.set(idx, counts.get(idx) - 1);
      ans.add(true);
      if (counts.get(idx) == 0) {
        counts.remove(idx);
        sizes.remove(idx);
      }
    }
```

```
51
52      StringJoiner sj = new StringJoiner(",");
53      for (Boolean an : ans) {
54        sj.add(an + "");
55      }
56      return sj.toString();
57    }
58  }
```

Python算法源码

```
1  # 输入获取
2  pools = list(map(lambda x: list(map(int, x.split(":"))), input().split(",")))
3  applies = list(map(int, input().split(",")))
4
5
6  # 二分查找
7  def binarySearch(arr, key):
8      low = 0
9      high = len(arr) - 1
10
11     while low <= high:
12         mid = (low + high) // 2
13         midVal = arr[mid]
14
15         if midVal > key:
16             high = mid - 1
17         elif midVal < key:
18             low = mid + 1
19         else:
20             return mid
21
22     return -low - 1
23
24
```

```python
# 算法入口
def getResult(pools, applies):
    pools.sort(key=lambda x: x[0])

    sizes = []
    counts = []

    for size, count in pools:
        sizes.append(size)
        counts.append(count)

    ans = []
    for apply in applies:
        idx = binarySearch(sizes, apply)

        if idx < 0:
            idx = -idx - 1

        if idx >= len(sizes):
            ans.append("false")
            continue

        counts[idx] -= 1
        ans.append("true")
        if counts[idx] == 0:
            counts.pop(idx)
            sizes.pop(idx)

    return ",".join(ans)


# 算法调用
print(getResult(pools, applies))
```