2022 年 3 月 25 日

划分为 K 个等和子集

尼基尔·库马尔·辛格
弗里希克

期间
5分钟

类别

竞争性编程

如何

标签

C++

分享

学到更多

TOPCODER 蓬勃发展

关于问题：

给定一个整数向量和一个整数 k，我们要找出是否可以将数组分成 k 个和相等的非空子集。如果可以，则返回 true，否则返回 false。

示例 1：
输入：arr = [5,2,1,2,3,4,3], k = 4
输出：true
解释：可能的子集是 - [2,3],[2,3],[5] ,[1,4]

示例 2：
输入：arr = [5,2,1,2,3,4,1], k = 4
输出：false

预计算

在讨论这些方法之前，我们应该检查将数组划分为 k 个相等的集合是否可行。为此，计算数组的整数之和是否可以被 k 整除。如果不是，我们不能分割数组，并且数组的大小必须等于或大于 k。

对于上述输入 [5,2,1,2,3,4,3]，k = 4。这里数组的长度，即 n >= k 和和，即 20 可以被 4 整除，所以20%4==0。

方法#1：回溯

1. 将当前子集的总和视为 currSum，我们在索引 i 处，我们希望每个子集的总和等于 reqSum = sum(nums[])/k。
2. 我们将使用visited[] 来保存已使用的元素。
3. 如果 currSum + nums[i] > reqSum，我们只能跳过第 i 个元素，去下一个元素。
4. 如果 !visited[i] 和 currSum + nums[i] <= resSum，我们有两种选择，要么将其包含在当前子集中，要么不将其包含在当前子集中。

5. 如果我们包含它，我们将使visited[i] = true 以便当前元素不能再次添加到任何其他子集中。
6. 如果我们发现选择这个选项给了我们真的，我们将返回真。
7. 否则，我们将通过设置visited[i] = false 从currSum 中排除nums[i] 并转到下一个元素。
8. 如果在任何时候我们找到 currSum == reqSum，这意味着我们找到了一个具有所需总和的子集，所以从第 0 个索引开始调用第 k-1 个子集的回溯。
9. 如果你的 k == 0，这意味着我们找到了 k 个总和为 reqSum 的子集。

代码：

```
bool backtrack(vector < int > & arr, int i, int reqSum, int currSum, vector < bool > &
if (k == 0)
  return true;
if (reqSum == currSum)
  return backtrack(arr, 0, reqSum, 0, isVisited, k - 1);

if (i == arr.size())
  return false;

if (!isVisited[i] && arr[i] + currSum <= reqSum) {
  isVisited[i] = true;
  if (backtrack(arr, i + 1, reqSum, currSum + arr[i], isVisited, k))
    return true;
  isVisited[i] = false;
 }

 return backtrack(arr, i + 1, reqSum, currSum, isVisited, k);
}

bool canPartitionIntoKSubsets(vector < int > & arr, int k) {

 int sum = 0;
 for (int ele: arr) sum += ele;
 if (sum % k != 0) return false;

 int n = arr.size();

 vector < bool > vis(n, false);
 return backtrack(arr, 0, sum / k, 0, vis, k);
}
```

时间复杂度： O(k*2^n)，对于每个子集，我们遍历整个数组并几乎在每次遍历中进行两次递归调用。

方法#2：位掩码

在这种方法中，我们检查每个子集的 sum = sum/k，如果子集有 req sum，我们增加计数。如果总计数大于等于 k，我们返回 true。我们使用位掩码来检查每个子集是否包含 sum = totalSum/k。因为每个元素只能在我们使用变量gone来跟踪选择的元素时才被获取。
例如，如果gone 变量二进制表示为10110，那么我们选择了第二个、第三个和第五个元素。

代码：

```
bool canPartitionIntoKSubsets(vector < int > & nums, int k) {

int n = nums.size();
int t = (1 << n);
sort(nums.begin(), nums.end(), greater < int > ());

int sum = 0;
for (int i = 0; i < n; i++) {
 sum += nums[i];
}

if (sum % k != 0) {
 return false;
}

sum = sum / k;
int cnt = 0;
int gone = 0;

for (int m = 0; m < t; m++) {
 int x = 0;
 if (!(m & gone)) {
  for (int i = 0; i < n; i++) {
   if (m & (1 << i)) {
    x += nums[i];

   }
  }
  if (x == sum) {
```

```
30          gone = m;
31          cnt++;
32        }
33      }
34    }
35    return cnt >= k;
36  }
```

时间复杂度： O((2^n)*n)

为你推荐