对“编程小练习：拆分自然数”的解答

张帅

2014 年 8 月 19 日

偶然间见到老赵的这道“编程小练习：拆分自然数”[1]，觉得非常有意思，难度适中，非常适合作为面试题。题目如下：

> 给出 sum、min、max 和 n 四个正整数，请输出所有将 sum 拆分为 n 个正整数之和，其中每个正整数 k 都满足：$\min \leq k \leq \max$。这 n 个正整数之间可以重复，不过由于加法交换率的作用，$1+2$ 和 $2+1$ 便算是重复的拆分了。
>
> 例如，$\text{sum}=5$，$n=3$，$\min=1$，$\max=3$，这时候满足条件的拆分方式只有两种：
>
> ◁ $1+1+3$
>
> ◁ $1+2+2$
>
> 这个练习和上次不同，我们假设所有的输入都是正整数。您无需对其进行合法性判断，只要专注于业务实现便是。

题目不难理解，难的是怎么想。

一 暴力解法

最容易想到的就是暴力解法了。

[1] http://blog.zhaojie.me/2009/06/1507847.html

1. 构造一条长为 n 的序列，序列中的每一个数都可以取 $[\min, \max]$ 的任意自然数。依次去验证这些序列中所有数字的和是否为 sum，得到一批序列。
2. 从上一步中得到的序列去除重复项。

算法也不难实现，首先是生成序列，见清单 1.1。

清单 1.1: 枚举所有序列

```
static IEnumerable<IEnumerable<int>> EnumerateSequences(int min, int max, int n) {
    if (n == 0) {
        return Enumerable.Repeat(Enumerable.Empty<int>(), 1);
    } else {
        return EnumerateSequences(min, max, n - 1).SelectMany(acc =>
            Enumerable
                .Range(min, max - min + 1)
                .Select(k => Enumerable.Repeat(k, 1).Concat(acc)));
    }
}
```

我们总是靠在长度为 $n-1$ 的序列头部加上一个在区间 $[\min, \max]$ 中的数来得到一个新的序列。当 $n=0$ 时，得到空序列。

这样，我们只需为生成的序列加上筛选条件 `Where(lst => lst.Sum() == sum)`，就可以得到第一步的结果。

上面那段代码中的 `Enumerable.Repeat(x, 1)` 太丑了，但是没办法，谁让 C# 中没有个像 **F#** 中 `Seq.singleton` 之类的方法呢？

第二步稍微复杂一点，但是既然都用了暴力解法了，时间复杂度稍微想一下至少也是 $O((\max-\min)^n)$ 了，也无所谓牺牲一点性能把功能先实现了。很好想到的方法是先把得到的每个序列排一下序，然后串成字符串看是否重复了，代码见清单 1.2。

这样一来，惨不忍睹的暴力解法就很容易写出来了（清单 1.3）

在细节上提高这一算法的性能已经没有任何实际意义了，但是我们的工作并没有白费。这个暴力解法简单易懂，几乎不可能写错，可以用来生成一些较小的测试用例。比如 `BruteForce(8, 0, 5, 3)` 的结果有：

清单 1.2: 比较序列相等

```
internal class EnumerableEqualityComparer<T> : IEqualityComparer<IEnumerable<T>>
{
    public bool Equals(IEnumerable<T> x, IEnumerable<T> y) {
        return ConvertToString(x) == ConvertToString(y);
    }

    public int GetHashCode(IEnumerable<T> obj) {
        return ConvertToString(obj).GetHashCode();
    }

    private static string ConvertToString(IEnumerable<T> source) {
        return string.Join(",", source);
    }
}
```

清单 1.3: 暴力解法

```
public static IEnumerable<IEnumerable<int>> BruteForce(
    int sum,
    int min,
    int max,
    int n)
{
    return EnumerateSequences(min, max, n)
        .Where(lst => lst.Sum() == sum)
        .Select(lst => lst.OrderBy(x => x))
        .Distinct(new EnumerableEqualityComparer<int>());
}
```

◁ 0, 3, 5
◁ 0, 4, 4
◁ 1, 2, 5
◁ 1, 3, 4
◁ 2, 2, 4
◁ 2, 3, 3

接下来，我们将使用这一测试用例验证其他比较复杂（但是高效）的算法。

二 仔细思考，收集信息

暴力解法给我们带来的启发性并不大，但是也能带来一定有用的信息。

稍微想一下我们就会发现，这个程序对于输入是有一定要求的：

1. $\text{sum} \geq 0$
2. $n \geq 1$
3. $\min \geq 0$
4. $\max \geq 0$
5. $\min \leq \max$ （仔细想一下，可不可以相等呢？）
6. $\min \cdot n \leq \text{sum} \leq \max \cdot n$

除此之外，在满足这些输入要求的情况下，**一定有解**。一定有解其实是一个很重要的性质。这意味着：

1. 我们不需要检查给定的输入是否有解，直接就可以开始运算；
2. 我们的算法不会因为输入的不同，而陷入死循环（不能停机）；
3. 我们的算法一定有输出。

三 使用递归的思想解决问题

应用数学思维来解决问题时，首先应当把问题的描述形式化，即使用数学语言来精确地描述问题。因此原问题转变为如下形式：

寻找恰当的函数 f，满足：

输入：$\mathrm{sum}, \min, \max, n \in \mathcal{N}$，其中 $\min \leq \max$，$n \geq 1$，$\min \cdot n \leq \mathrm{sum} \leq \max \cdot n$。

输出：所有的序列 $\{a_1, a_2, \ldots, a_n\}$，满足 $\sum_{k=1}^{n} a_k = \mathrm{sum}$，且 $\min \leq a_1 \leq a_2 \leq \cdots \leq a_n \leq \max$。

从这里开始，我们只考虑输入满足要求的情况。

递归的本质就是数学归纳法，而数学归纳法要从基础情况开始。因此我们首先应该考虑基础情况。什么是这个问题的基础情况呢？答案是当 $n = 1$ 的时候。此时我们有且只有一个解——只有一个元素 sum 的序列。

然后我们假设自己已经能解决 $n = k - 1$ $(k > 1)$ 情况下的所有问题，考虑怎么利用这些结果解决 $n = k$ 时的问题。假设我们让 a_1 随便取一个在区间 $[\min, \max]$ 中的值，那么剩下的序列就应该是 $f(\mathrm{sum} - a_1, a_1, \max, n - 1)$ 中的任意一个序列。但是我们还应该仔细的检查，调用 f 时的参数是否满足 f 对输入的要求。

现在，只有 a_1 是可以变化的。依次检查输入限制条件，显然 $\min' \leq \max'$ 成立，由 $k > 1$ 保证 $n' = n - 1 = k - 1 \geq 1$ 成立。求解 $\min' \cdot n' \leq \mathrm{sum}' \leq \max' \cdot n'$，得到 $a_1 \leq \frac{\mathrm{sum}}{n}$ 和 $\mathrm{sum} - \max \cdot (n - 1) \leq a_1$。因为 $a_1 \in \mathcal{N}$，所以 $a_1 \leq \lfloor \frac{\mathrm{sum}}{n} \rfloor$。现在，我们得到关于 a_1 的三个条件：

1. $a_1 \in [\min, \max], a_1 \in \mathcal{N}$
2. $a_1 \leq \lfloor \frac{\mathrm{sum}}{n} \rfloor$
3. $a_1 \geq \mathrm{sum} - \max \cdot (n - 1)$

现在我们得到了所有的，从 $n = k - 1$ 得到 $n = k$ 的解的信息：v 对于所有满足上述条件的 a_1，在其后加上 $f(\mathrm{sum} - a_1, a_1, \max, n - 1)$ 得到的一组序列，得到一组新的序列，即为所求。根据数学归纳法，我们的解是正确的。

至此，写出代码已经不再是一个问题了。（见清单 3.1）

最后不要忘了写一个接口，检查用户的输入是否满足我们对输入的要求。

写完之后瞄了一眼老赵的答案，他没检查 $fracsumn$ 应该向上还是向下取整。

清单 3.1: 深度优先搜索遍历所有解

```
public static IEnumerable<IEnumerable<int>> DFS(int sum, int min, int max, int n) {
    if (n == 1) {
        yield return Enumerable.Repeat(sum, 1);
    } else {
        for (int a0 = Math.Max(min, sum - max * (n - 1));
            a0 <= sum / n;
            a0++)
        {
            foreach (var lst in DFS(sum - a0, a0, max, n - 1)) {
                yield return Enumerable.Repeat(a0, 1).Concat(lst);
            }
        }
    }
}
```