1.6 饮料供货

★★★

在微软亚洲研究院上班，大家早上来的第一件事是干啥呢？查看邮件？No，是去水房拿饮料：酸奶，豆浆，绿茶、王老吉、咖啡、可口可乐……（当然，还是有很多同事把拿饮料当做第二件事）。

管理水房的阿姨们每天都会准备很多的饮料给大家，为了提高服务质量，她们会统计大家对每种饮料的满意度。一段时间后，阿姨们已经有了大批的数据。某天早上，当实习生小飞第一个冲进水房并一次拿了五瓶酸奶、四瓶王老吉、三瓶鲜橙多时，阿姨们逮住了他，要他帮忙。

从阿姨们统计的数据中，小飞可以知道大家对每一种饮料的满意度。阿姨们还告诉小飞，STC（Smart Tea Corp.）负责给研究院供应饮料，每天总量为 V。STC 很神奇，他们提供的每种饮料之单个容量都是 2 的方幂，比如王老吉，都是 2^3=8 升的，可乐都是 2^5=32 升的。当然 STC 的存货也是有限的，这会是每种饮料购买量的上限。统计数据中用饮料名字、容量、数量、满意度描述每一种饮料。

那么，小飞如何完成这个任务，求出保证最大满意度的购买量呢？

分析与解法

【解法一】

我们先把这个问题“数学化”一下吧。

假设 STC 共提供 n 种饮料，用（S_i、V_i、C_i、H_i、B_i）（对应的是饮料名字、容量、可能的最大数量、满意度、实际购买量）来表示第 i 种饮料（$i = 0, 1, \cdots, n-1$），其中可能的最大数量指如果仅买某种饮料的最大可能数量，比如对于第 i 中饮料 $C_i=V/V_i$。

基于如上公式：

饮料总容量为$\sum_{i=0}^{n-1}(V_i * B_i)$；

总满意度为$\sum_{i=0}^{n-1}(H_i * B_i)$；

那么题目的要求就是，在满足条件$\sum_{i=0}^{n-1}(V_i * B_i)=V$ 的基础上，求解 $\max\{\sum_{i=0}^{n-1}(H_i * B_i)\}$。

对于求最优化的问题，我们来看看动态规划能否解决。用 Opt（V', i）表示从第 i, $i+1$, $i+2$, …, $n-1$, n 种饮料中，算出总量为 V'的方案中满意度之和的最大值。

因此，Opt（V, n）就是我们要求的值。

那么，我们可以列出如下的推导公式：Opt（V', i）= max { $k* H_i$ + Opt（$V' - V_i * k$, $i-1$）}（$k = 0, 1, \cdots, C_i$, $i = 0, 1, \cdots, n-1$）。

即：最优化的结果 = 选择第 k 种饮料×满意度+减去第 k 种饮料×容量的最优化结果根据这样的推导公式，我们列出如下的初始边界条件：

Opt（0, n）= 0，即容量为 0 的情况下，最优化结果为 0。

Opt（x, n）= −INF（x != 0）（－INF 为负无穷大），即在容量不为 0 的情况下，把最优化结果设为负无穷大，并把它作为初值。

那么，根据以上的推导公式，就不难列出动态规划求解代码，如下所示：

代码清单 1-9

```
int Cal(int V, int type)
{
    opt[0][T] = 0;// 边界条件
    for(int i = 1; i <= V; i++)// 边界条件
    {
        opt[i][T] = -INF;
    }
    for(int j = T - 1; j >= 0; j--)
    {
        for(int i = 0; i <= V; i++)
        {
```

```
            opt[i][j] = -INF;
            for(int k = 0; k <= C[j]; k++)     // 遍历第j种饮料选取数量k
            {
                if(i <= k * V[j])
                {
                    break;
                }
                int x = opt[i - k * V[j]][j + 1];
                if(x != -INF)
                {
                    x += H[j] * k;
                    if(x > opt[i][j])
                    {
                        opt[i][j] = x;
                    }
                }
            }
        }
    }
    return opt[V][0];
}
```

在上面的算法中，空间复杂度为 $O(V*N)$，时间复杂度约为 $O(V*N*\text{Max}(C_i))$。

因为我们只需要得到最大的满意度，则计算 opt[i][j]的时候不需要 opt[i][j+2]，只需要 opt[i][j]和 opt[i][j+1]，所以空间复杂度可以降为 $O(v)$。

【解法二】

应用上面的动态规划法可以得到结果，那么是否有可能进一步地提高效率呢？我们知道动态规划算法的一个变形是备忘录法，备忘录法也是用一个表格来保存已解决的子问题的答案，并通过记忆化搜索来避免计算一些不可能到达的状态。具体的实现方法是为每个子问题建立一个记录项。初始化时，该纪录项存入一个特殊的值，表示该子问题尚未求解。在求解的过程中，对每个待求解的子问题，首先查看其相应的纪录项。若记录项中存储的是初始化时存入的特殊值，则表示该子问题是第一次遇到，此时计算出该子问题的解，并保存在其相应的记录项中。若记录项中存储的已不是初始化时存入的初始值，则表示该子问题已经被计算过，其相应的记录项中存储的是该子问题的解答。此时只需要从记录项中取出该子问题的解答即可。

因此，我们可以应用备忘录法来进一步提高算法的效率。

代码清单 1-10

```
int[V + 1][T + 1] opt;      // 子问题的记录项表，假设从 i 到 T 种饮料中，
                            // 找出容量总和为V’ 的一个方案，快乐指数最多能够达到
                            // opt（V'，i，T-1），存储于opt[V’][i]，
                            // 初始化时opt中存储值为-1，表示该子问题尚未求解。
int Cal(int V, int type)
{
    if(type == T)
    {
        if(V == 0)
            return 0;
        else
            return -INF;
    }
    if(V < 0)
        return -INF;
    else if(V == 0)
        return 0;
    else if(opt[V][type] != -1)
        return opt[V][type];    // 该子问题已求解，则直接返回子问题的解;
                                // 子问题尚未求解，则求解该子问题
    int ret = -INF;
    for(int i = 0; i <= C[type]; i++)
    {
        int temp = Cal(V – i * C[type], type + 1);
        if(temp != -INF)
        {
            temp += H[type] * i;
            if(temp > ret)
            ret = temp;
        }

    }
    return opt[V][type] = ret;
}
```

【解法三】

请注意这个题目的限制条件，看看它能否给我们一些特殊的提示。

我们把信息重新整理一下，按饮料的容量（单位为 L）排序：

Volume	TotalCount	Happiness
20L	TC_00	H_00
20L	TC_01	H_01
...	...	...
21L	TC_10	H_10
...	...	...
2000L	TC_M0	H_M0
...	...	...

假设最大容量为 2 000L。一开始，如果 V%（21）非零，那么，我们肯定需要购买 20L 容量的饮料，至少一瓶。在这里可以使用贪心规则，购买快乐指数最高的一瓶。除去这个，我们只要再购买总量（V−20）L 的饮料就可以了。这时，如果我们要购买 21L 容量的饮料怎么办呢？除了 21L 容量里面快乐指数最高的，我们还应该考虑，两个容量为 20L 的饮料组合的情况。其实我们可以把剩下的容量为 20L 的饮料之快乐指数从大到小排列，并用最大的两个快乐指数组合出一个新的“容量为 2L”[1]的饮料。不断地这样使用贪心原则，即得解。这是不是就简单了很多呢？

[1] 如果各种饮料数量都无限的话，这种方法是很简单。但是如果饮料有个数限制，复杂度可能达到指数级，您有更好的办法么？