傅申 PB20000051
2022 秋

算法基础 作业 7

16.2–2. 设计动态规划算法求解 0–1 背包问题, 要求运行时间为 $O(nW)$, n 为商品数量, W 为小偷能放进背包的最大商品总重量.

解: 假设商品 i 在最优解中, 则最优解去掉商品 i 后为最大总重量 $W-w_1$ 子问题的最优解, 满足最优子结构性质. 假设 $V_{i,j}$ 为前 i 个商品最大总重量 j 子问题中小偷能拿走的最大价值, 则若 $w_i > j$, 则 $V_{i,j} = V_{i-1,j}$; 否则应该考虑商品 i 是否会在解中的情况, 即 $V_{i,j} = \max\left(V_{i-1,j}, V_{i-1,j-w_i} + v_i\right)$, 动态规划算法如下:

Algorithm 1: 0–1-KNAPSACK(w, v, n, W)

```
let V[0..n, 0..W] be a new table
for i = 1 to n
    V[i, 0] = 0
for j = 1 to W
    V[0, j] = 0
for i = 1 to n
    for j = 1 to W
        if j < w[i]
            V[i, j] = V[i − 1, j]
        else
            V[i, j] = max(V[i − 1, j], V[i − 1, j − w[i]] + v[i])
return V[n, W]
```

16.2–4. Gekko 教授一直梦想用直排轮滑的方式横穿北达科他州. 他计划沿 U.S. 2 号高速公路横穿, 这条高速公路从明尼苏达州东部边境的大福克斯市到靠近蒙大拿州西部边境的威利斯顿市. 教授计划带两公升水, 在喝光水之前能滑行 m 英里 (由于北达科他州地势相对平坦, 教授无需担心在上坡路喝水速度比平地或下坡路快). 教授从大福克斯市出发时带整整两公升水. 他携带的北达科他州官方地图显示了 U.S. 2 号公路上所有可以补充水的地点, 以及这些地点间的距离.

教授的目标是最小化横穿途中补充水的次数. 设计一个高效的方法, 以帮助教授确定应该在那些地点补充水. 证明你的策略会生成最优解, 分析其运行时间.

解: 使用贪心算法, 每次都选择在喝光水之前能滑行到的最远的补水点, 能生成最优解. 证明如下:

假设最优解为 O, 对应一系列补水点 $o_1, o_2, \cdots, o_m$, 贪心算法生成的解为 G, 对应一系列补水点 $g_1, g_2, \cdots, g_n$, 其中 $o_m = g_n$ 为终点. 因为每次都选择在喝光水之前能滑行到的最远的补水点, 所以有 $g_1 \geqslant o_1$,

$g_2 \geqslant o_2, \cdots, g_m \geqslant o_m$, 因为 O 是最优的, 所以 $n \geqslant m$, 而 o_m 为终点, 所以 $g_m = o_m$ 是终点, 因此 $n = m$, 即贪心算法是最优的.

设共有 n 个补水点, 则算法的运行时间为 $O(n)$.

16.3–3. 如下所示, 8 个字符对应的出现频率是斐波纳契数列的前 8 个数, 此频率集合的赫夫曼编码是怎样的?

a: 1, b: 1, c: 2, d: 3, e: 5, f: 8, g: 13, h: 21

你能否推广你的结论, 求频率集为前 n 个斐波那契数的最优前缀码?

解: 赫夫曼编码依次为 a: 0000000, b: 0000001, c: 000001, d: 00001, e: 0001, f: 001, g: 01, h: 1.

下面证明 $\sum_{i=1}^{n-1} F(i) = F(n+1) - 1$, 其中 $F(i)$ 为斐波那契数列的第 i 个数. 显然当 $n = 1$ 时成立, 假设 $n = k \geqslant 1$ 时成立, 则当 $n = k + 1$ 时有 $\sum_{i=1}^{k} F(i) = F(k) + \sum_{i=1}^{k-1} F(i) = F(k) + F(k+1) - 1 = F(k+2) - 1$, 由数学归纳法可知等式成立. 因此, 若频率集为前 n 个斐波那契数, 则构造赫夫曼编码树的过程中, 每次都选择频率最小的未被选择的字符, 和已经构造好的赫夫曼编码树合并, 生成新的赫夫曼编码树, 直到所有字符都被选择. 设字符依次为 $c_1, c_2, \cdots, c_n$, 则赫夫曼编码为 $c_1 : \underbrace{0\cdots0}_{n-1个0}$, $c_i : \underbrace{0\cdots0}_{n-i个0}1$, $2 \leqslant i \leqslant n$.

16.3–4. 证明: 编码树的总代价还可以表示为所有内部结点的两个孩子结点的联合频率之和.

解: 考虑使用数学归纳法, 当编码树的高度为 1 时, 内部结点只有根结点, 两个孩子的联合频率之和即为唯二两个字符的频率之和, 与总代价相等. 假设编码树高度小于等于 k 时题中命题成立, 则当编码树的高度为 $k + 1$ 时, 编码树的总代价即为两个子编码树的总代价之和再加上所有叶结点的频率之和. 编码树的所有内部结点的两个孩子结点的联合频率之和等于两个子编码树的所有内部结点的两个孩子结点的联合频率之和加上两个子编码树根结点的频率之和, 因为叶结点的频率之和就是两个子编码树根结点的频率之和, 所以编码树的所有内部结点的两个孩子结点的联合频率之和等于编码树的总代价. 由数学归纳法可知题中命题成立.

17.1–2. 证明: 如果 k 位计数器的粒子中允许 Decrement 操作, 那么 n 个操作的运行时间可能达到 $\Theta(nk)$.

解: 假设计数器中保存的数为 0, 依次执行 Decrement 操作和 Increment 操作 n 次, 每次都需要改变计数器的所有 k 位 ($0\cdots0 \to 1\cdots1 \to 0\cdots0 \to \cdots$), 因此运行时间为 $\Theta(nk)$.

17.2–3. 假定我们不仅对计数器进行增 1 操作, 还会进行置 0 操作 (即将所有位复位). 设检测或修改一个位的时间为 $\Theta(1)$, 说明如何用一个位数组来实现计数器, 使得对一个初值为 0 的计数器执行一个由任意 n 个 INCREMENT 和 RESET 操作组成的序列花费时间为 $O(n)$.

解: 维护一个指针, 其值为位数组最高位的 1 对应的下标, 初始值为 -1. 在 INCREMENT 操作中按原来的步骤对各位进行置位/复位, 然后判断是否需要更新指针的值, 如果需要则更新. 在 RESET 操作中, 只需要依次对下标为 0 和指针之间的位进行复位即可.

对一次置位操作, 设其摊还代价为 3 元, 1 元用于支付置位操作的实际代价, 剩下 2 元存为信用: 1 元用于在 INCREMENT 中被复位, 1 元用于在 RESET 中被置为 0. 设判断和更新指针值的代价为 1 元. 因为 INCREMENT 操作中最多置位一次, 所以其摊还代价最多为 4 元; 而对于 RESET 操作, 可以不缴纳任何费用, 将下标为 0 和指针值之间的每一位中的 1 元信用取出来支付其实际代价即可. 因此, 对于一个由任意 n 个 INCREMENT 和 RESET 操作组成的序列, 其总摊还代价为 $O(n)$, 为实际代价的上界.

17.3–3. 考虑一个包含 n 个元素的普通二叉最小堆数据结构, 它支持 INSERT 和 EXTRACT-MIN 操作, 最坏情况时间均为 $O(\lg n)$. 给出一个势函数 Φ, 使得 INSERT 操作的摊还代价为 $O(\lg n)$, 而 EXTRACT-MIN 操作的摊还代价为 $O(1)$, 证明它是正确的.

解: 设势函数为 $\Phi(D_i) = \sum_{j=1}^{n} \lg j$, n 是堆的大小, INSERT 操作的摊还代价为

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = \lg n + \sum_{j=1}^{n+1} \lg i - \sum_{j=1}^{n} \lg j = \lg n + \lg(n+1) = O(\lg n) \tag{3.1}$$

EXTRACT-MIN 操作的摊还代价为

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = \lg n + \sum_{j=1}^{n} \lg i - \sum_{j=1}^{n-1} \lg j = \lg n - \lg n = 0 = O(1) \tag{3.2}$$