贪心算法作业

左逸龙 1120231863 07112303

1 Huffman 编码

1.1 问题 1

字符	编码
h	0
g	10
f	110
e	1110
d	11110
c	111110
b	1111110
a	1111111

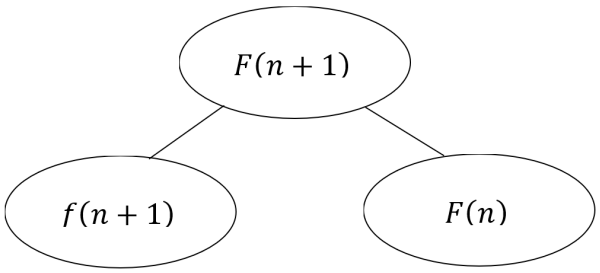
1.2 问题 2

已知 n 个字符的频率恰好是前 n 个 $Fibonacci$ 数，则：

对于第 1 个字符，其编码为：$\underbrace{11\cdots1}_{(n-1)个1}$

对于第 i $(i=2,3,\cdots,n)$ 个字符，其编码为：$\underbrace{11\cdots1}_{(n-i)个1}0$

证明：设 $F(n)$ 表示前 n 项的累计和，$f(n)$ 表示第 n 个 $Fibonacci$ 数，则在第 n 步合并操作，总能形成如下子树结构：



即证对 $\forall n\in\mathbb{N}^+$，总有 $F(n)$ 与 $f(n+1)$ 最小。(此时 $f(i)(i=1,2,\cdots,n)$ 已被合并，故不考虑)

由于对 $\forall n\in\mathbb{N}^+\wedge n\neq 1$，有：

$$f(n+1)=f(n)+f(n-1)>f(n)$$

即 $Fibonacci$ 数列单调递增,则对 $\forall n\in\mathbb{N}^+$,总有 $f(n+1)<f(n+2)$。此时若能证明 $F(n)<f(n+2)$，则原命题得证。下用数学归纳法证明：

1. 当 $n=1$ 时，$F(1)=1$，$f(1+2)=f(3)=2$，有 $F(n)<f(n+2)$ 成立；

2. 对 $\forall k\in\mathbb{N}^+$，若 $F(k)<f(k+2)$ 成立，下证 $F(k+1)<f((k+1)+2)$ 成立:

$$\begin{aligned} F(k+1)<f((k+1)+2) &\Leftrightarrow F(k)+f(k+1)<f(k+2)+f(k+1)\\ &\Leftrightarrow F(k)<f(k+2)\end{aligned}$$

得证。

由数学归纳法可知：对 $\forall n\in\mathbb{N}^+$，总有 $F(n)<f(n+2)$ 成立。

故原命题得证。

2 最优分解问题

算法 1: 贪心算法求解最优分解问题 (仅给出一般情况)

输入: 正整数 n

输出: 最大乘积 $maxProduct$ 和分解方案 $factors$

```
初始化 factors ← []// 存储分解方案的数组
初始化 product ← 1// 最大乘积
初始化 sum ← 0// 当前因子的总和
// 第一遍分配
for i ← 2 to n do
    if sum + i > n then
        break;
    end
    将 i 加入 factors;
    sum ← sum + i;
end
remainder ← n − sum// 处理剩余部分
for i ← len(factors) − 1 to 0 do
    if remainder > 0 then
        factors[i] ← factors[i] + 1;
        remainder ← remainder − 1;
    end
end
if remainder = 1 then
    factors[len(factors) − 1] ← factors[len(factors) − 1] + 1// 分配剩余的 1 到最后一个因子上
end

// 遍历所有因子计算 product
foreach factor in factors do
    product ← product × factor;
end
return product, factors;
```

分析：首先需要判断一些特殊情况

1. $n=1$，此时分解为 $0+1$，最大乘积即为 0；
2. $n=2$，此时分解为 $0+2$，最大乘积即为 0；
3. $n=3$，此时分解为 $1+2$，最大乘积即为 2；
4. $n=4$，此时分解为 $1+3$，最大乘积即为 3；
5. $n\geq 5$，按照下述规则分解:
   i. 将 n 分解为 $2+3+\cdots+t+R$，其中 $R\leq t$；
   ii. 将 R 分解为 $\underbrace{1+1+\cdots+1}_{R个}$；
   iii. 若 $R<t$，则将 R 从大到小依次分摊一个 1 到前 $t-1$ 个因数上，直至分配完毕；
   iv. 若 $R=t$，则在上述过程的基础上，将剩余的一个 1 分配到最大的因数上。

证明[1]：

对任意正的自然数 $k\geq 2$，令 $h(k)=2+3+\cdots+k$。

对任意正的自然数 $n\geq 2$，存在着唯一的正自然数 k 使得 $h(k)\leq n<h(k+1)$。

令 $m=n-h(k)$，必有 $0\leq m\leq k$。

称如下 n 的分解方案为 n 的分解方案 A：

- 若 $m=0$，则分解为 $\{2,3,\ldots,k\}$；
- 若 $1\leq m\leq k-1$，则分解为 $\{2,3,\ldots,k+1\}\setminus\{k-m+1\}$；
- 若 $m=k$，则分解为 $\{3,4,\ldots,k+2\}\setminus\{k+1\}$。

定义函数 $t(n)$ 为：

$$t(n)=\begin{cases}k, & 若m=0,\\ k+1, & 若0<m<k,\\ k+2, & 若m=k.\end{cases}$$

注意到，一方面，n 的分解方案 A 的最大数等于 $t(n)$；另一方面，总有 $t(n)>m$，所以 $n-t(n)<h(k)$。显然，若 n 的一个分解含有 1，则这个分解不可能是最优分解，因为将这个 1 加到最大数上会得到更大的乘积。此外，从分解方案 A 的定义可以看出，如果最大数小于 $t(n)$，那么不可能得到各数互不相同且不含 1 的分解方案。于是，对任意大于 1 的自然数 n，若 $n=a_1+a_2+\cdots+a_k$ 是 n 的各数都大于 1 且不同的任一分解，则 $\max_{1\leq i\leq k}a_i\geq t(n)$。

这里的贪心选择的策略已经呼之欲出：$t(n)$。最优子结构性质也似乎很明显。看起来分解方案 A 就是最优分解。但是我们却无法先独立地证明贪心选择性质和最优子结构性质，再证明分解方案 A 是最优分解。

对任意 $n\geq 2$，定义函数 $g(n)$ 为 n 的分解方案 A 的乘积。首先来分析 $g(n)$ 的性质。对于 $n=2,3,4$，分解方案 A 就是 n 本身，所以 $g(2)=2$，$g(3)=3$，$g(4)=4$。

[1]本人不太会证，参考网页:最优分解问题贪心算法的数学证明

引理 1

对任意 $n > 4$，我们有 $g(n) = t(n) \cdot g[n - t(n)]$，且 $t(n) > t[n - t(n)]$。对任意 $n > 2$，有 $g(n) \cdot g(n-1) \geq (k+2)(k+1)$。(1)

证明： 对于 $n > 4$，设 $h(k) \leq n < h(k+1)$，令 $m = n - h(k)$。

性质 $g(n) = t(n) \cdot g[n - t(n)]$ 和 $t(n) > t[n - t(n)]$ 容易由分解方案 A 的定义验证。

- 当 $1 < m \leq k-1$ 时，$g(n) \cdot g(n-1) = (k-m+2)(k-m+1) \geq (k+2)(k+1)$；
- 当 $m = k$ 时，$g(n) \cdot g(n-1) = (k+2)(k+1) \geq (k+2)(k+1)$；
- 当 $m = 1$ 时，$g(n) \cdot g(n-1) = (k+1)k \geq (k+2)(k+1)$；
- 当 $m = 0$ 且 $k > 3$ 时，$g(n) \cdot g(n-1) = 2k(k+1) \geq (k+2)(k+1)$；
- 当 $m = 0$ 且 $k = 3$，即 $n = 5$ 时，$g(n) \cdot g(n-1) = 6 \cdot 4 \geq (k+2)(k+1)$。

对于 $n = 3, 4$，可直接验证 $g(n) \cdot g(n-1) \geq 4 \cdot 3$。

综合上面情况得（1）。证毕。

定理 2

分解方案 A 是最优分解。

证明： 定义函数 $f(n)$ 为 n 的最优分解的乘积。

我们将证明如下断言 $P(k)$ 对所有 $k \geq 2$ 成立：$P(k)$：若 $h(k) \leq n < h(k+1)$，则 $f(n) = g(n)$。

下面采用数学归纳法证明。

基础情况： 易知 $h(2) = 2$，$h(3) = 5$。同时，容易验证对 $n = 2, 3, 4$，$f(n) = g(n)$，所以 $P(2)$ 成立。

归纳假设： 假设 $P(i)$ 对 $i \leq k-1$ 都成立，要证明 $P(k)$ 成立。

归纳步骤： 考虑如下的 n：$h(k) \leq n < h(k+1)$。根据前面的分析，n 的任意不含 1 的分解的最大数都不小于 $t(n)$，所以

$$f(n) = \max\{f(n-j) \cdot j : t(n) \leq j \leq n-2\} = \max\{g(n-j) \cdot j : t(n) \leq j \leq n-2\},$$

其中第二个等式是由于 $j \geq t(n)$，从而 $n - j < h(k)$，于是由归纳假设有 $f(n-j) = g(n-j)$。

由（1）可得

$$f(n-j) \cdot j > f(n-j-1) \cdot (j+1)$$

所以

$$f(n) = f[n - t(n)] \cdot t(n) = g[n - t(n)] \cdot t(n) = g(n)$$

$P(k)$ 成立。

综上所述，由数学归纳法，分解方案 A 是最优分解，证毕。

3 分发饼干

算法 2: 贪心算法求解分发饼干问题

输入: 孩子胃口值数组 g，饼干尺寸数组 s

输出: 能满足的最大孩子数量 $satisfied$

```
对数组 g 和 s 进行升序排序;
初始化指针 i ← 0，j ← 0，满足的孩子数量 satisfied ← 0;
while i < g.size() 且 j < s.size() do
    if s[j] ≥ g[i] then
        satisfied ← satisfied + 1;
        i ← i + 1// 移动到下一个孩子
    end
    j ← j + 1// 移动到下一个饼干
end
return satisfied;
```

分析 我们需要尽可能将大饼干分给胃口大的孩子，或者尽可能将小饼干分给胃口小的孩子，两种做法都可以，这里我采取了后者。

贪心策略的正确性证明

证法 1：数学归纳法

1. **贪心选择性质：** 假设已知一最优解与贪心策略解不相同，由于贪心策略总是保证将尺寸尽可能小的饼干分配给胃口小的孩子，因此总是可以找到小尺寸的饼干替换大尺寸的饼干，使最优解与贪心策略解相同。
2. **最优子结构：** 假设对于某一状态，已经满足了前 k 个孩子，剩余的问题仍然是一个相同的子问题，即在剩下的饼干中尽量满足剩余的孩子。贪心选择不会影响后续的最优性。
3. **结论：** 贪心策略对于每一子问题总是选择当前局部最优解，由归纳法可知，贪心策略可以保证全局最优。

证法 2：反证法

1. **贪心策略：** 排序后，匹配胃口最小的孩子和能满足他的最小饼干。
2. **假设反例：** 假设存在一更优解使得某次分配没有将当前胃口最小的孩子与最小满足他的饼干匹配，而是选择了其他分配方式。
   i. 由于饼干已经按尺寸排序，匹配其他饼干可能浪费掉最小的饼干，从而使得剩余孩子无法满足。
   ii. 这种情况的结果只可能会比贪心策略产生更小的满足孩子数量。
3. **结论：** 贪心策略可以保证全局最优解。

4 无重叠区间

算法 3: 贪心算法解决无重叠区间问题

输入: n 个区间，每个区间的左右端点

输出: 需要移除的最小区间数量

```
定义数组 intervals 表示所有区间;
sort(intervals) // 对区间按照右端点排序
end ← −∞ // 上一个区间的结束时间
count ← 0 // 可以安排的区间数量
for 从 i ← 0 到 n − 1 do
    if intervals[i].left ≥ end then
        count ← count + 1 // 当前区间可以安排
        end ← intervals[i].right // 更新结束时间
    end
end
return n − count // 总区间数减去可以安排的区间数
```

分析： 本题实际上是活动分配问题的变体，因此可以将问题转化为 “选取最多的互不重叠的区间”，剩余的区间数量即为需要移除的区间数量。贪心策略如下：

- 按区间的右端点升序排序；
- 从左到右遍历区间，选择起点不早于当前结束时间的区间；
- 统计选取的区间数量，剩余的区间数量即为需要移除的数量。

贪心策略的正确性证明

贪心选择性质 每次选择右端点最小且不与已选择区间冲突的区间：

- 选择右端点最小的区间，可以保证留给后续区间的时间最多；
- 若通过不选择右端点最小的区间能够得到一最优解，则将该区间更换为右端点最小的区间，同样能够保证得到最优解。

最优子结构 设当前已经选取的区间集合为 S，当前结束时间为 end。对于后续区间，问题可以表示为一个子问题：

选取所有左端点 $\geq end$ 的区间，使得选取的区间数量最多。

在该子问题中，贪心选择右端点最小的区间依然是最优解。

归纳证明

- 初始状态下，贪心选择右端点最小的区间是最优的；
- 假设在前 $k-1$ 个区间中，贪心策略已经得到最优解，对于第 k 个区间，依然选择右端点最小的区间，可以使得整体结果最优。

因此，贪心策略可以保证全局最优。

5 跳跃游戏

算法 4: 贪心算法解决跳跃游戏问题

输入: 数组 $nums$，表示每个位置的跳跃最大距离

输出: 布尔值，表示是否能够到达终点

```
初始化 maxRange ← 0;
n ← 数组长度;
for i ← 0 to n − 1 do
    if i > maxRange then
        return false// 当前位置超出最远距离，无法到达
    end
    更新 maxRange ← max(maxRange, i + nums[i]);
    if maxRange ≥ n − 1 then
        return true// 最远距离覆盖终点，返回 true
    end
end
return true// 遍历完成，说明可以到达终点
```

分析 贪心算法能够解决问题的关键在于，当前维护的变量 $maxRange$ 始终反映了遍历到当前元素时，能够到达的最远位置。

贪心策略的正确性证明

贪心选择性质 在每个位置 i，更新 $maxRange = \max(maxRange, i + \text{nums}[i])$，确保当前能跳跃的范围最大，这样保证了每一步选择的范围最优。如果存在一种更优的跳跃方式，则它一定选择了某个位置 i 的子最优跳跃范围（即未使用当前贪心策略的跳跃范围）。假设这种方式能够到达终点，则必然可以通过贪心选择的跳跃方式到达终点，因为贪心策略每次都选择了当前跳跃范围内的最优解：

$$\text{贪心策略的最远距离} \geq \text{其他策略的最远距离}$$

因此贪心策略一定能够解决问题。

最优子结构 当前能够到达的 $maxRange$ 已经包含了所有之前跳跃可能性的信息：

- 每一步更新 $maxRange$ 后，只需要考虑从当前位置跳跃的可能性；
- 由于之前的所有跳跃范围已经反映在 $maxRange$ 中，不需要回溯检查其他位置。

因此问题可以分解为一个个独立的子问题，符合最优子结构性质。

归纳证明

- 初始状态下，$maxRange = 0$，可以覆盖起点；
- 假设在第 k 步能够保证 $maxRange$ 是所有跳跃范围的最优解；
- 在第 $k+1$ 步时，更新 $maxRange$，仍然符合最优解条件。

因此，贪心策略可以确保全局最优解。

6 摆动序列

算法 5: 贪心算法解决摆动序列问题

输入: 整数数组 $nums$

输出: 摆动序列的最大长度

```
if size(nums) < 2 then
    return size(nums) // 数组长度小于 2，直接返回长度
end
prevDiff ← 0 // 上一次相邻元素的差值
length ← 1 // 摆动序列的长度
for i ← 1 to size(nums) − 1 do
    diff ← nums[i] − nums[i − 1] // 计算当前差值
    if length = 1 or diff × prevDiff < 0 then
        length ← length + 1 // 差值方向改变，更新序列长度
        prevDiff ← diff // 更新前一个差值
    end
end
return length
```

分析 对于一段连续的上升序列，我们只需要选取首尾两个点即可，因为首部越低，越能保证小于摆动序列中的直接前驱；尾部越高，越能保证大于摆动序列中的直接后继。对于一段连续的下降序列，同样可以选取首尾两点。如果前后两个值相等，则我们只保留其中一个值即可。

贪心策略的正确性证明

贪心选择性质 对于一段连续的上升或下降序列，贪心策略选择首尾两个点，保证总是选择该段序列的最小值与最大值。假设存在另一种最优解与贪心策略所得解不同，则这种方法一定包含某段上升或下降序列中未选择的中间元素，而这些中间元素对摆动方向没有贡献，因此不会增加摆动序列的长度，于是必定会有：

$$\text{贪心策略所得摆动序列长度} \geq \text{另一种最优解所得摆动序列长度}$$

因此贪心策略所得解包含于最优解集合当中。

最优子结构 每次选择当前点后，剩余问题的摆动序列长度只与当前点和后续点有关，与之前的选择无关。于是问题可以分解为独立的子问题。

归纳证明

- 初始状态下，贪心选择单调序列的右端点是最优的；
- 假设遍历到第 k 个单调序列时，摆动序列长度能够保证最优，则遍历到第 $k+1$ 个单调序列时，摆动序列长度仍能够保证最优；

因此，由数学归纳法可以证明贪心策略能够得到全局最优解。

7 餐馆问题

算法 6: 贪心算法解决餐馆问题

输入: n 张桌子，每张桌子的容量 a；m 批客人，每批客人的人数 b 和消费金额 c

输出: 最大预计消费金额

$tables \leftarrow [a_1, a_2, \ldots, a_n]$;
$customers \leftarrow [(b_1, c_1), (b_2, c_2), \ldots, (b_m, c_m)]$;
$sort(tables)$ // 按照桌子容量从小到大排序
$sort(customers)$ // 按照客人消费金额从大到小排序
定义多重集合 $availableTables \leftarrow tables$ // 可能有多张重复桌子，这样查找桌子更快
$totalMoney \leftarrow 0$ // 总消费金额初始化为 0
for $i \leftarrow 0$ to $m-1$ do
    $t \leftarrow lowerBound(availableTables, customers[i].b)$ // 查找第一个能容纳当前客人的桌子
    if $t \neq \emptyset$ then
        $totalMoney \leftarrow totalMoney + customers[i].c$ // 增加当前客人的消费金额
        $erase(availableTables, t)$ // 移除已经使用的桌子
    end
end
return $totalMoney$

分析 每次选择客人和桌子时，应优先考虑消费金额最大的客人，并为其分配能容纳他们的最小桌子。

贪心策略的正确性证明

贪心选择性质 每一次安排均能保证局部最优解：

- 如果消费金额较大的客人未被优先安排，而消费金额较小的客人被安排，则可能会导致总消费金额减少，贪心选择能够避免这种情况；
- 为消费金额较大的客人分配最小的合适桌子，可以为后续客人保留更大容量的桌子，提高后续选择的灵活性。

最优子结构 当前选择的总预计消费金额只依赖于当前的客人安排和剩余的桌子容量，与之前的安排无关。因此，若当前已经安排了一部分客人和桌子，此时剩余的客人和桌子仍然可以通过贪心策略进行选择。整个问题可以分解为多个独立的子问题，满足最优子结构性质。

归纳证明

- 初始状态下，将所有客人按照消费金额从大到小排序，所有桌子按照容量从小到大排序，保证初始选择的最优性；
- 假设在第 k 步贪心选择已经是最优的，对于第 $k+1$ 步，贪心策略仍然优先选择消费金额最大的未分配客人，并为其分配最小的合适桌子，根据贪心选择性质和最优子结构性质，第 $k+1$ 步的选择仍能保证最优。

因此，由数学归纳法可以证明贪心策略能够得到全局最优解。