第五章-作业

1

（30 分）假定我们不再一直选择最早结束的活动，而是选择最晚开始的活动，前提仍然是与之前选出的所有活动兼容。描述如何利用这一方法设计贪心算法，并证明算法会产生最优解。

1.1 概念和问题定义

1.1.1活动

设 $S=\{1,2,...,n\}$ 是 n 个活动的集合，各个活动使用同一个资源，资源在同一时间只能为一个活动使用每个活动 i 有起始时间 s_i，终止时间 f_i，$s_i \leq f_i$。

1.1.2 相容活动

活动 i 和 j 是相容的，若 $s_j \leq f_i$ 或 $s_i \leq f_j$。

1.1.3 问题

- 输入：$S=\{1,2,...,n\}$，$F=\{[s_i,f_i]\}$，$n \geq i \geq 1$
- 输出：S 的最大相容集合

1.2 优化子结构 & 贪心选择性

1.2.1 优化子结构

引理 1

设 $S=\{1,2,...,n\}$ 是 n 个活动的集合，$[s_i,f_i]$ 是活动的起始终止时间，且 $s_1 \geq s_2 \geq \cdots \geq s_n$，$S$ 的活动选择问题的某个优化解包括活动 1。

证明：

设 A 是一个优化解，按起始时间排序 A 中活动，设其第一个活动为 k，第二个活动为 j。

- 如果 $k=1$，引理成立。
- 如果 $k \neq 1$，令 $B=A-\{k\}+\{1\}$，由于 A 中活动相容，$s_1 \geq s_k \geq f_j$，B 中活动相容。因为 $|B|=|A|$，所以 B 是一个优化解，且包括活动 1。

引理 2

设 $S=\{1,2,...,n\}$ 是 n 个活动的集合，$[s_i,f_i]$ 是活动的起始终止时间，且 $s_1 \geq s_2 \geq \cdots \geq s_n$，设 A 是 S 的调度问题的一个优化解且包括活动 1，则 $A'=A-\{1\}$ 是 $S'=\{i \in S | f_i \leq s_1\}$ 的调度问题的优化解。

证明：

- 显然，$A'=A-\{1\}$ 中的活动是相容的。
- 我们仅需要证明 A' 是 S' 最大的。设不然，存在一个 S' 的活动选择问题的优化解 B'，$|B'|>|A'|$。
- 令 $B=\{1\}+B'$，对于 $\forall i \in S'$，$f_i < s_1$，B 中活动相容。B 是 S 的一个解。由于 $|A|=|A'|+1$，$|B|=|B'|+1>|A'|+1=|A|$，与 A 最大矛盾。

引理 2 说明活动选择问题具有优化子结构

1.2.2 贪心选择性

引理 3

设 $S=\{1,2,...,n\}$ 是 n 个活动的集合，$s_0=\infty$，l_i 是 $S_i=\{j \in S | f_j \leq s_{i-1}\}$ 中具有最大起始时间 s_{l_i} 的活动，设 A 是 S 的包含活动 1 的优化解，则

$$A=\bigcup_{i=1}^{k}\{l_i\}$$

证明：

对 $|A|$ 作归纳法。

- 但 $|A|=1$ 时，由引理 1，命题成立。
- 设 $|A|<k$ 时，命题成立。当 $|A|=k$ 时，由引理 2，$A=\{1\}+A_1$，A_1 是 $S_2=\{j \in S | f_j \leq s_1\}$ 的优化解。
- 由归纳假设，$A_1=\bigcup_{i=2}^{k}\{l_i\}$。于是，$A_1=\bigcup_{i=1}^{k}\{l_i\}$。

1.3 算法

1.3.1 算法设计

贪心思想

为了选择最多的相容活动，每次选 s_i 最大的活动，使我们能够选择更多的活动。

1.3.2 算法

Greedy-Activity-Selector

Input：活动的开始，结束时间数组 S，F，设$s_1 \geq s_2 \geq \cdots \geq s_n$（已排序）

Output：选择的活动集合

执行：Greedy-Activity-Selector(S, F)

过程：Greedy-Activity-Selector(S, F)

```
n <- length(S)
A <- {1}
j <- 1
for i <- 2 to n do
    if F[i] <= S[j] (当前第 i 个活动起始的时间小于等于第 j 个活动结束的时间)
        then A <- A + {i}
             j <- i
return A
```

1.3.3 复杂性分析

- 如果结束时间 $\{f_i\}_{i=1}^{1}$ 已排序，则 $\boldsymbol{T(n)}=\Theta(\boldsymbol{n})$
- 如果结束时间 $\{f_i\}_{i=1}^{1}$ 未排序（先对序列 进行排序）
  $\boldsymbol{T(n)}=\Theta(\boldsymbol{n})+\Theta(n \lg n)=\Theta(n \lg n)$

1.3.4 正确性证明（算法会产生最优解）

- 活动选择问题具有优化子结构

  由引理 2 可知活动选择问题具有优化子结构

- 活动选择问题具有贪心选择性

  由引理 3 知活动选择问题具有贪心选择性

- 算法按照贪心选择性计算优化解

  Greedy-Activity-Selector 算法按照引理 3 的贪心选择性进行局部优化选择

2

（30 分）考虑用最少的硬币找 n 美分零钱的问题。假定每种硬币的面额都是整数。

$a.$ 设计贪心算法求解找零问题。假定有 25 美分、10 美分、5 美分和1 美分 4 种面额的硬币。

$b.$ 设计一组硬币面额，使得贪心算法不能保证的到最优解。这组硬币面额中应该包含 1 美分，使得对每个零钱值都存在找零方案。

答：

$a.$ 为了用最少的硬币，每次选面额最大的硬币，直到最后剩余找零数为 0。

$b.$ 因为包含 1 美分，所以使得对每个零钱值都存在找零方案。（真妙）

- 硬币面额：$\{1, 7, 12\}$
- 找 14 美分零钱
- 贪心结果：$\{12, 1, 1\}$
- 最优解：$\{7, 7\}$

3

（40 分）编程题：柠檬水找零

题目描述：

在柠檬水摊上，每一杯柠檬水的售价为 5 美元。

顾客排队购买你的产品，（按账单 $bills$ 支付的顺序）一次购买一杯。

每位顾客只买一杯柠檬水，然后向你付 5 美元、10 美元或 20 美元。你必须给每个顾客正确找零，也就是说净交易是每位顾客向你支付 5 美元。

注意，一开始你手头没有任何零钱。

如果你能给每位顾客正确找零，返回 $true$ ，否则返回 $false$ 。

提示：

$0 \leq bills.length \leq 10000$

$bills[i]$ 不是 5 就是 10 或是 20

示例 1：

> 输入：$[5, 5, 5, 10, 20]$
>
> 输出：$true$
>
> 解释：

> 前 3 位顾客那里，我们按顺序收取 3 张 5 美元的钞票。
>
> 第 4 位顾客那里，我们收取一张 10 美元的钞票，并返还 5 美元。
>
> 第 5 位顾客那里，我们找还一张 10 美元的钞票和一张 5 美元的钞票。
>
> 由于所有客户都得到了正确的找零，所以我们输出 $true$。

示例 2：

> 输入：$[5, 5, 10]$
>
> 输出：$true$

示例 3：

> 输入：$[10, 10]$
>
> 输出：$false$

示例 4：

> 输入：$[5, 5, 10, 10, 20]$
>
> 输出：$false$
>
> 解释：
>
> 前 2 位顾客那里，我们按顺序收取 2 张 5 美元的钞票。
>
> 对于接下来的 2 位顾客，我们收取一张 10 美元的钞票，然后返还 5 美元。
>
> 对于最后一位顾客，我们无法退回 15 美元，因为我们现在只有两张 10 美元的钞票。
>
> 由于不是每位顾客都得到了正确的找零，所以答案是 $false$。

要求：运用贪心思想作答，请写出分析过程，并用一种语言（最好是 $C++$ 或 $JAVA$ ）实现你的思路，复杂度尽可能低。

原题：[860. 柠檬水找零 - 力扣(LeetCode)]

答：

分析：

输入是数组，每一个数的取值都是：$\{5, 10, 20\}$。

10 美元只能用 5 美元找，20 美元能用 3 个 5 美元找，或是用 1 个 10 美元和 1 个 5 美元找。5 美元直接收下。

收到的 20 美元对于找零毫无用处（所以甚至不用统计 20 美元），甚至损失了能用来找零的钱；**找零的时候倾向于先用 1 张 10 美元和 1 张 5 美元找，没有 10 美元在考虑用 3 张 5 美元找。因为 5 美元在什么情况下都通用。**这就是贪心思想。

代码：

```
#include <stdio.h>
#include <stdbool.h>
bool lemonadeChange(int *bills, int billsSize)
```

```c
{
    int five = 0;
    int ten = 0;

    for (int i = 0; i < billsSize; i++)
    {

        if (bills[i] == 5)
            five++;
        else if (bills[i] == 10)
        {
            ten++;
            five--;
        }
        else
        {
            if (ten == 0)
                five = five - 3;
            else
            {
                ten--;
                five--;
            }
        }

        if (five < 0 || ten < 0)
            return false;
    }

    return true;
}

int main()
{
    int bills[1000] = {5, 5, 5, 10, 20};
    int length = 5;
    printf("%d", lemonadeChange(bills, length));
    return 0;
}
```

源代码：[3.c]

吐槽：

我的评价是：做生意都不带零钱，没有电子支付...还是找个厂上班吧。

似乎没有什么用的考虑：

1. 如果存在一场这样找零成功的交易，则一定有数组的各项和等于 5 乘以数组长度。故如果不满足这个必要条件，则找零必不成功，返回 $false$。（使用这一条件的时间复杂度为 $O(n)$）

2. 顺着思考：

- 显然第一位顾客给的必须是 5 美元，如果不是 5 美元可以直接返回 $false$。
- 在第一位顾客给 5 美元的情况下，我们现在手上有 1 张 5 美元。所以第二位顾客给出 5 或 10 美元均可；而若是 20 美元可以直接返回 $false$。

- 在第二位顾客给 5 美元的情况下（默认能到第二位，之后也相同），我们现在手上有 2 张 5 美元。所以第三位顾客给出 5 或 10 美元均可；而若是 20 美元可以直接返回 $false$。
- 在第二位顾客给 10 美元的情况下，我们手上有 1 张 5 美元，1 张 10 美元，找给顾客 5 美元，最后手上有 1 张 10 美元。所以第三位顾客必须给出 5 美元；如果不是 5 美元可以直接返回 $false$。
- 在 $[5, 5, 5]$ 的情况下，我们现在手上有 3 张 5 美元。所以第四位顾客给出的钞票都能接受。
- 在 $[5, 5, 10]$ 的情况下，我们手上有 2 张 5 美元，1 张 10 美元，找给顾客 5 美元，最后手上有 1 张 5 美元，1 张 10 美元。所以第四位顾客给出的钞票都能接受。
- 在 $[5, 10, 5]$ 的情况下，我们手上有 1 张 5 美元，1 张 10 美元。所以第四位顾客给出的钞票都能接受。
- ...这么下去应该能行...