Algorithm Assignment 5

> *10185101210 陈俊潼*

1

Using methods that's quite a like to the previous activity schedule problem. The difference is that the schedule time is a loop.

1. Find the activities with shortest length. Denote as A_0
2. Let the begin time T_0 of A_0 as the begin of the daily loop, then sort rest of the problems with their begin time
3. From T_0 to the next day's T_0, there are 24 hours. Unfold the loop into a linear scheduling problem, then find the earliest compatible job recursively.

This method can promise the optimum solution because the first step uses the begin time of the shortest activity, which is the optimum choice for deciding the begin time of the day.

16.1-2

By finding the last activity that is compatible, this approach is exactly the same as the original problem but executes in reverse order. It's gives out the optimum solution because every time when selecting an activity, we are selecting the best choice for this step.

16.2-6

The pseudocode is shown as follows:

```
LINEAR_KNAPSACK_PROBLEM(w, v, n, W, start, end, currentSelectedWeight, currentValue):
    if (currentSelectedWeight >= W)
        return currentValue
    let unit_value = []
    for i = 0 to n:
        unit_value[n] = v / w
    while (currentSelectedWeight < W)
        mediam = LINEAR_FIND_MIDIAM(unit_value)
        for i = 0 to n:
            larger_sum = 0
            larger_value = 0
            if unit_value[i] > mediam:
                larger_sum += w[i]
                larger_value += v[i]
           if larger_sum > w
               LINEAR_KNAPSACK_PROBLEM(w, v, n, W, start + end / 2, end, currentSelectedWeight, currentValue)
           else
               currentSelectedValue += larger_value
               LINEAR_KNAPSACK_PROBLEM(w, v, n, W - larger_sum, start , start + end / 2 , currentSelectedWeight, currentValue)

```

This is a recursive function whose runtime is $T(n) = T(n/2) + cn$, which gives a complexity of O(n).