算法设计与分析: 作业 #5

201928015059003
杨天健

2019 年 11 月 3 日

Question 1

(1)

The naive algorithm is $\Theta(n^2)$ in complexity.

(2)

The divide-and-conquer algorithm is described as below. From the master's theorem, the time complexity is $\Theta(nlog_2(n))$.

Algorithm 1 Devide-and-conquer algorithm

Input: $A[l...r]$ is the sub-array of A. The interval is closed here.

Ouput: $\max_{l,r} \sum_{i=l}^{r} A[i]$

Function solve(A, l, r)

if $l = r$ **then return** A[l]
$mid = (l + r)/2$
$ansl = solve(A, l, mid)$
$ansr = solve(A, mid + 1, r)$
$p = mid$, $q = mid + 1$, $ansm1 = 0$, $ansm2 = 0$
$cur = 0$
while $p >= l$ **do**
    $cur = cur + A[p]$
    if $cur > ansm1$ **then**
        $ansm1 = cur$
    $p = p + 1$
$cur = 0$
while $q <= r$ **do**
    $cur = cur + A[q]$
    if $cur > ansm2$ **then**
        $ansm2 = cur$
    $q = q + 1$
$ansm = ansm1 + ansm2$
return $\max(ansl, ansr, ansm)$

(3)

The dynamic programming formula is:

$$b[j] = \begin{cases} a[1] & j = 1 \\ max(a[j], b[j-1] + a[j]) & j \geq 2 \end{cases} \tag{1}$$

And the answer is $\max_{j=1,2,\ldots,n} b[j]$. The complexity is just $\Theta(n)$.

Question 2

The problem has a property that if we change the sequence of the tasks within those machines, the solution will remain the same. Therefore, despite the solution is like a permutation, we can still consider them by their index order.

Let $f[i][t]$ be the solution after taking task $1, 2, ..., i$ in consideration, given machine A and B has a difference t as to the time finishing their own works.

This time, we can think from top to bottom, we have:

$$f[i+1][t+a[i+1]] = f[i][t] + \max(0, a[i+1] + \min(t, 0)) \tag{2}$$

$$f[i+1][t-b[i+1]] = f[i][t] + \max(0, b[i+1] - \max(t, 0)) \tag{3}$$

The initialization is $f[0][0] = 0$ and $f[i][-T...T] = \infty$. Here $T = \sum_{i=1}^{n} \max(a[i], b[i])$. The answer is $\min_{t=-T,-T+1,...,T} f[n][t]$. The code below has passed the test:

```python
a = [0, 2, 5, 7, 10, 5, 2]
b = [0, 3, 8, 4, 11, 3, 4]
n = len(a) - 1
f = []
g = []
T = 0
for i in range(n + 1):
    f.append(dict())
    g.append(dict())
    T += max(a[i], b[i])
def assign_min(dic, idx, val):
    if idx not in dic:
        dic[idx] = val
        return True
    else:
        if val < dic[idx]:
            dic[idx] = val
            return True
        else:
            return False
f[0][0] = 0
for i in range(n):
    for t in range(-T, T + 1):
        if t in f[i]:
            if a[i + 1] < -t and t < 0:
                if assign_min(f[i + 1], t + a[i + 1], f[i][t]):
                    g[i + 1][t + a[i + 1]] = {"prev": (i, t), "decision": 'A'}
            elif a[i + 1] >= -t and t < 0:
                if assign_min(f[i + 1], t + a[i + 1], f[i][t] + a[i + 1] + t):
                    g[i + 1][t + a[i + 1]] = {"prev": (i, t), "decision": 'A'}
            else:
                if assign_min(f[i + 1], t + a[i + 1], f[i][t] + a[i + 1]):
                    g[i + 1][t + a[i + 1]] = {"prev": (i, t), "decision": 'A'}

            if b[i + 1] < t and t > 0:
                if assign_min(f[i + 1], t - b[i + 1], f[i][t]):
                    g[i + 1][t - b[i + 1]] = {"prev": (i, t), "decision": 'B'}
            elif b[i + 1] >= t and t > 0:
                if assign_min(f[i + 1], t - b[i + 1], f[i][t] + b[i + 1] - t):
                    g[i + 1][t - b[i + 1]] = {"prev": (i, t), "decision": 'B'}
            else:
                if assign_min(f[i + 1], t - b[i + 1], f[i][t] + b[i + 1]):
                    g[i + 1][t - b[i + 1]] = {"prev": (i, t), "decision": 'B'}
```

```python
trace = []
opt = 9999999999
t0 = None
for t in range(-T, T + 1):
    if t in f[n]:
        if f[n][t] < opt:
            opt = f[n][t]
            t0 = t
trace.append(g[n][t0])
for i in range(n, 1, -1):
    t0 = g[i][t0]['prev'][1]
    trace.append(g[i - 1][t0])

print(opt)
trace = list(reversed(trace))
print([x['decision'] for x in trace])
```

And the result is:

```
15
['A', 'A', 'B', 'B', 'A', 'A']
```

So the task 1, 2, 4, 6 should be assigned to A and others should be assigned to B. The total time is 15.

Question 3

Denote $f[i][j]$ as $\max \sum_{k=1}^{i} c_i \cdot x_i$, such that $\sum_{k=1}^{i} a_i \cdot x_i \leq j$.

Therefore, $f[i][j] = \max_{k \in \{0,1,2\} \, and \, k \cdot a[i] \leq j} f[i-1][j-k*a[i]] + k*c[i]$. Here $1 \leq i \leq n$ and $0 \leq j \leq b$. And $f[0][j] = 0$ for $j = 0, 1, ..., b$. The answer required is $f[n][b]$ .Similar to the knapsack problem, the time complexity is $\Theta(min(3^n, nb, n\sum_{i=1}^{n} c_i))$.

Question 4

Denote $f[i][j]$ as the maximum log-reliability after considering the $1, 2, ..., i$-th machine. The dp formula is:

$$f[i][j] = \max_{k \in N \mid j-k*c[i] \geq 0} f[i-1][j-k*c[i]] + ln(g_i(k)) \tag{4}$$

The answer required is $e^{f[n][c]}$.