091M4041H - Algorithm Design and Analysis

Assignment 3

2019 年 8 月 30 日

目录

1 Problem One

Given a list of n natural numbers $d_1, d_2, ..., d_n$, show how to decide in polynomial time whether there exists an undirected graph G = (V, E) whose node degrees are precisely the numbers $d_1, d_2, ..., d_n$. G should not contain multiple edges between the same pair of nodes, or ”loop” edges with both endpoints equal to the same node.

1.1 Optimal Substructure and greedy-choice property

考虑每次选择一个节点出列，则其他的节点的度数均会减一。贪心的选择是每次寻找最大每次选择度数最大的节点 d_1 出列，其他所有的度数减一。因此贪心选择为公式（1）所示：

$$choose\ max\{d_1, d_2, ..., d_n\} \tag{1}$$

最优子结构为公式（2）所示：

$$OPT(d_1, d_2, ..., d_n) = OPT(d_2 - 1, ..., d_n - 1)\ if\ d_1 = max\{d_1, d_2, ..., d_n\} \tag{2}$$

1.2 Algorithm Description

算法首先将 S 中的自然数排序，每次将最大度数的自然数取出 S 集合，并且其他在 S 中的度数减 1，直到 S 中出现负数时，说明 S 中的序列不能构成一张无环图，当 S 中所有节点值均为 0 时，则说明其能构成一张无环图。其中在初始条件时存在一定的限制条件，当度数和为奇数时，一定不是无环图；当仅有一个节点是，其确定是一张图；因最大度数必不超过 n-1，当最大节点度数大于等于节点总个数时，其不是一张无环图。

1.3 Proof of Algorithm Correctness

采用数学归纳法证明。

对于 n=1 时，因一个节点时，其自成一张图，因此算法返回 1。

当 n=k 时，因最大度数必不超过 n-1，当最大节点度数大于等于节点总个数时，其不是一张无环图。所以余下的序列组合最大值不超过 n-1，共有 n 个节点，因此算法每去除一个节点所有节点减 1，必能使所有节点达到 0。若途中已达到负数则退出返回不能构成一张无环图。

综上所述，该算法是正确的。

1.4 Complexity Analysis

度数求和时的时间复杂度为 O(n), 排序算法时，最快排序时间复杂度为 O(nlogn)，每个度数减 1 时时间复杂度为 O（n^2），判断是否存在负数或度数全 0 时时间复杂度为 O(n)，因此算法的时间复杂度为：

$$T(n) = O(n^2) \tag{3}$$

Algorithm 1 Judge whether there exists an undirected graph without circle

```
function SearchAcyclicGraph($S(d_1, d_2, ..., d_n)$)
    if sum(S) is odd then
        return 0;
    end if
    if len(S) == 1 then
        *return* 1;
    end if
    *sort* $S(d_1, d_2, ..., d_n)$;
    if $len(S) \leq d_1$ then
        return 0;
    end if
    for $i$ *from* 1 *to* $n$ do
        $S = S - d_i$;
        for $j$ *from* $i + 1$ *ton* do
            $d_j = d_j - 1$;
        end for
        if *all* $d_i$ *in* $S$ *are* 0 then
            return 1;
        end if
        if *exist* $d_i$ *in* $S$ $< 0$ then
            return 0;
        end if
    end for
end function
```

2 Problem Twe

There are n distinct jobs, labeled $J_1, J_2, ..., J_n$, which can be performed completely independently of one another. Each job consists of two stages: first it needs to be preprocessed on the supercomputer, and then it needs to be finished on one of the PCs. Let's say that job J_i needs p_i seconds of time on the supercomputer, followed by f_i seconds of time on a PC. Since there are at least n PCs available on the premises, the finishing of the jobs can be performed on PCs at the same time. However, the supercomputer can only work on a single job a time without any interruption. For every job, as soon as the preprocessing is done on the supercomputer, it can be handed off to a PC for finishing.

Let's say that a schedule is an ordering of the jobs for the supercomputer, and the completion time of the schedule is the earlist time at which all jobs have finished processing on the PCs. Give a polynomial-time algorithm that finds a schedule with as small a completion time as possible.

2.1 Optimal Substructure and greedy-choice property

因为作业最早结束时间是在 PC 端和超级计算机共有的时间，并且 PC 共有 n 台且能同时处理，因此选择在 PC 端时间长的部分优先处理，这样多个 PC 机可以并行执行多个较长的作业，不占用多余的时间，直到算法结束。因此贪心选择如公式（4）所示：

$$choose\ max\{f_1, f_2, ..., f_n\} \tag{4}$$

最优子结构为公式（5）所示：

$$OPT(p_1, p_2, ...p_n, f_1, f_2, ..., f_n) = OPT(p_2, ...p_n, f_2, ..., f_n)\ if\ p_1 = max\{p_1, p_2, ..., p_n\} \tag{5}$$

2.2 Algorithm Description

算法首先将作业在 PC 的执行时间 f_i 由大到小排序，每次寻找最长时间的 f_i，同时找到该作业对应在超级计算机上执行的时间 p，执行完 p 时间后将该作业较由 PC 处理。

Algorithm 2 Find a schedule with as small a completion time as possible

```
1: function SmallestTimeSchedule($J(j_1, j_2, ..., j_n), P(p_1, p_2, ..., p_n), F(f_1, f_2, ..., f_n)$)
2:     sort $F(f_1, f_2, ..., f_n)$;
3:     for $i$ from 1 to $n$ do
4:         $F = F - f_i$;
5:         find p correspond to $f_i$;
6:         $P = P - p$;
7:         wait for p time and PC processes j corresponding to $f_i$;
8:     end for
9:     return 0;
10: end function
```

2.3 Proof of Algorithm Correctness

采用数学归纳法证明。

当 n=1 时，只有一个作业，选取该作业即可完成调度工作。

当 n=k 时，由于 PC 能同时处理，让时间较长的 f 尽早挂起有助于减少 PC 端的等待时间，由于超级计算机每次只能执行一个作业，所以 p 的总时间一定，但较大的 f 尽早挂起，假设 $f_{max} > \sum_{i=1}^{n} p_i$, 使得总时间为 $p_1 * 2 + f_{max} - \sum_{i=1}^{n} p_i$，若将 f_{max} 放在第 k 个执行, 则存在可能性，使得在执行完所有 p 任务时时间为 $\sum_{i=1}^{k} p_i + f_{max} - \sum_{i=k+1}^{n} p_i$。显然前者时间更少。

综上所述，算法是正确的。

2.4 Complexity Analysis

排序算法的时间复杂度为 O(nlogn)，寻找对应的 p,f,j 时的时间复杂度为 O(n)。因此算法的时间复杂度为:

$$T(n) = O(nlogn) \tag{6}$$

3 Problem Three

Given two strings s and t, check if s is subsequence of t? A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, ”ace” is a subsequence of ”abcde” while ”aec” is not).

3.1 Optimal Substructure and greedy-choice property

由于需要确定字符串的顺序不变，则每次选择 s 中的第一个字符与 t 从头开始匹配，匹配到后将 s_1 抛弃，再选择剩余 s 中的第一个字符，与 t 中接下来的部分继续匹配。因此贪心选择如公式 (7) 所示：

$$choose\ first\ s_1\ in\ S(s_1, s_2, ..., s_n)\ and\ if\ t_k\ matched,\ choose\ T(t_{k+1}, ..., t_n) \tag{7}$$

最优子结构为公式（8）所示：

$$OPT(s_1, s_2, ...s_n, t_1, t_2, ..., t_m) = OPT(s_2, ...s_n, t_{k+1}, ..., t_m)\ if\ \ s_1 = t_k \tag{8}$$

3.2 Algorithm Description

算法每次选取 s 中的数来对 t 查找，若找到对应的字符，则下一个 s 会在 t 中接下来的位置继续查找, 若任一个 s 在 t 中未找到则返回 0，若都找到了，则返回 1。

Algorithm 3 Check if s is subsequence of t

```
function CheckSubsequence($S(s_1, s_2, ..., s_n), T(t_1, t_2, ..., t_m)$)
    temp = 1;
    for $i$ from 1 to $n$ do
        for j from temp to m do
            if $s_i == t_j$ then
                temp = j++;
                break;
            end if
            return 0;
        end for
    end for
    return 1;
end function
```

3.3 Proof of Algorithm Correctness

采用数学归纳法证明。

当 s 和 t 只有 1 个字符时，算法显然是成立的。

当 s 和 t 分别有 n 和 m 个字符时，由于 s 是按顺序执行的，且 t 在执行过程中只遍历了一遍，所以必定找到 t 中的所有字符与 s 进行比较，即寻找 t 的子集过程。

综上所述，算法是正确的。

3.4 Complexity Analysis

由于算法保证对 t 中的字符进行了遍历，所以时间复杂度与 t 的长度有关，因此算法的时间复杂度为:

$$T(n) = O(m) \tag{9}$$

4 Problem Four

Given a rope whose length is n, please cut the rope to m parts to get the maximum product of the length of each part $\prod_{l_1+l_2+..+l_m=n} l_1 * l_2 * ... * l_m$. For example, if a rope with length 8, when we cut it to 2, 3, 3, we can get the maximum product 18.

4.1 Optimal Substructure and greedy-choice property

因为绳子的总长不变，要得到最大的乘积，只需要让切分的绳子尽可能相等，因此贪心选择如公式（10）所示：

$$choose\ l_i\ is\ nearly\ equal\ in\ L(l_1, l_2, ..., l_n) \tag{10}$$

最优子结构为公式（11）所示：

$$OPT(l_1, l_2, ..., l_n) = OPT(l_i, l_i, ..., l_i)\ if\ l_i\ is\ nearly\ equal\ in\ L(l_1, l_2, ..., l_n) \tag{11}$$

4.2 Algorithm Description

算法需要判断 n/m 的小数部分是大于 0.5 还是小于 0.5，若大于 0.5，则说明存在 m-1 个 n/m 上取整的数和 1 个 n/m 下取整的数其总和为 n，若小于 0.5，则说明存在 m-1 个 n/m 下取整的数和 1 个 n/m 上取整的数其总和为 n。

Algorithm 4 Get the maximum product of the length

```
function MaximumProduct($n, m$)
    if m==1 then
        return n;
    end if
    if $n/m - \lfloor n/m \rfloor > 0.5$ then
        $product = \lceil n/m \rceil^{m-1} * \lfloor n/m \rfloor$;
    else
        $product = \lfloor n/m \rfloor^{m-1} * \lceil n/m \rceil$;
    end if
    return product;
end function
```

4.3 Proof of Algorithm Correctness

采用数学归纳法证明。

当 m=1 时，其乘积就为长度 n 本身。

当 m=k 时，n/m 的小数部分是大于 0.5 时，必存在 k-1 个 n/k 上取整的数和 1 个 n/k 下取整的数其总和为 n，因为 k-1 个上取整的数比 m/n 多出的部分加和会由最后一个下取整的数填补使总和仍为 n，因此即找到了和为 n 的最约等的一串数字，其乘积必为最大乘积。

综上所述，算法是正确的。

4.4 Complexity Analysis

因为算法只有计算步骤而与长度 n 和个数 m 无关，因此算法的时间复杂度为：

$$T(n) = O(1) \tag{12}$$