Algorithm Assignment 3

201828018670050 王纪宝

2018 年 11 月 14 日

目录

1 Problem 1: 度序列可简单图化 **1**
1.1 Basic idea and Pseudo-code . . . 1
1.1.1 Basic idea . . . 1
1.1.2 Pseudo-code . . . 2
1.2 Greedy-choice Property and Optimal Substructure . . . 2
1.3 Prove the Correctness . . . 2
1.4 The complexity of this algorithm . . . 3

2 Problem 4: 剪绳子得最大乘积 **3**
2.1 Basic idea and Pseudo-code . . . 3
2.1.1 Basic idea . . . 3
2.1.2 Pseudo-code . . . 3
2.2 Greedy-choice Property and Optimal Substructure . . . 3
2.3 Prove the Correctness . . . 4
2.4 The complexity of this algorithm . . . 4

1 Problem 1: 度序列可简单图化

1.1 Basic idea and Pseudo-code

1.1.1 Basic idea

给定一个非负整数序列 $\{d_n\}$，若存在一个无向图使得图中各点的度与此序列一一相应, 则称此序列可图化。进一步, 若图为简单图，则称此序列可简单图化。至于能不能依据这个序列构造一个图，就须要依据 Havel-Hakimi 定理中的方法来构图。

可图化的判定：$d_1 + d_2 + \ldots\ldots + d_n \equiv 0(mod 2)$。关于详细图的构造, 我们能够简单地把奇数度的点配对，剩下的所有搞成自环。

可简单图化的判定（Havel-Hakimi 定理）：把序列排成不增序，即 $d_1 \geq d2 \geq \ldots\ldots \geq d_n$，则 d 可简单图化当且仅当 $d' = \{d_2 - 1, d_3 - 1, \ldots\ldots, d_{d_1+1} - 1, d_{d_1+2}, d_{d_1+3}, \ldots\ldots, d_n\}$ 可简单图化。简单的说，把 d 排序后，找出度最大的点 (设度为 d_1)，把它与度次大的 d_1 个点之间连边。然后这个点就能够排除了，一直继续这个过程，直到建出完整的图。

当然构图过程中也会出现不合理的情况。

1：某次对剩下序列排序后，最大的度数比剩下的顶点数还要多。

2：度数减 1 后，出现负数。

上面两种情况都是无法构成图的。

1.1.2 Pseudo-code

Algorithm 1: 度序列可否简单图化判定 (Havel-Hamiki 定理)

```
Input: 度序列 {d_n}
Output: True or False
if sum({d_n})%2 != 0 then
    return False
end
for i = 0 → n − 1 do
    sort(d_i.....d_n);
    if d_i > n − i − 1 then
        return False
    end
    for j = i + 1 → i + d_i do
        if d_j == 0 then
            return False
        end
        d_j − −
    end
end
return True
```

1.2 Greedy-choice Property and Optimal Substructure

贪心选择特性：先将度序列按照非增序进行排序，每次贪心的选择当前度最大的节点，依次与后面的次大度点之间连接一条边。

最优子结构：当前最大度的点处理完成后，将其删除，接下来只考虑剩下的点如何进行图化即可。

1.3 Prove the Correctness

算法每次都选取最大度数的点是为了后面有足够多的点可以抵消该点的度数，首先考虑该算法排除的情况是否确实不能构成简单图，其排除条件包括度数总和不为偶数、最大的度数超过节点数以及连接过程出现了负数度数的点，第一个是能构成图的充要条件，而第二个条件说明该节点必然存在环或平行边，第三个条件表示连接了孤立节点，这是不允许的，所以图中必然出现了环或平行边。所以该算法排除的情况都是不能构成简单图的情况。

那么是否存在该算法成功确定的序列却不能构成简单图的情况呢？这显然是不可能，因为任意的一个简单图都能转换成每个节点连接的都是度数不小于它的节点的简单图，例如，首先随机找出一个节点，连接其他 d_1 个节点，之后就不再考虑该节点，然后在剩下的节点中找度数最大的节点，给它分配第二大的度数，即与 d_2 个节点连接，如此类推，直到分配完所有度数。

综上所述，该算法是正确的。

1.4 The complexity of this algorithm

一个遍历中嵌套一个排序，所以算法的时间复杂度为 $O(n^2 \log n)$。

2 Problem 4: 剪绳子得最大乘积

2.1 Basic idea and Pseudo-code

2.1.1 Basic idea

使用贪心策略解决本问题的想法：为了使各段绳子的乘积最大，当 $n \geq 5$ 时，我们尽可能多地剪长度为 3 的绳子；当剩下的绳子长度为 4 时，把绳子剪成两段长度为 2 的绳子，这种思路可保证最后的乘积最大。

2.1.2 Pseudo-code

```
int maxProductAfterCutting(int length){
    if( length<2 )
        return 0;
    else if( length==2 )
        return 1;
    else if( length==3 )
        return 2;
    //尽可能多地剪去长度为的绳子段3
    int timesOf3 = length/3;

    //当绳子最后剩下的长度为的时候，不能再剪去长度为的绳子段43
    //此时更好的方法是把绳子剪成长度为的两段，因为2 2*2 > 3*1
    if( length-timesOf3*3 == 1 )
        timesOf3 -= 1;

    int timesOf2 = (length-timesOf3*3)/2;
    return (int)(pow(3,timesOf3)) * (int)(pow(2,timesOf2));
}
```

2.2 Greedy-choice Property and Optimal Substructure

贪心选择特性：当 $n \geq 5$ 的时候，我们贪心地每次剪掉一段长为 3 的绳子段，不需考虑到全局最优，只是保证当前乘积变大，最后会得出最终乘积最大。

最优子结构：从当前绳子上剪掉一段长为 3 的绳子段后，我们接着处理剩下的 $n-3$ 的绳子段即可。

2.3 Prove the Correctness

首先，当 $n \geq 5$ 的时候，我们可以证明 $2(n-2) > n$ 并且 $3(n-3) > n$。也就是说，当绳子剩下的长度大于或者等于 5 的时候，我们就把它剪成长度为 3 或者 2 的绳子段，另外，当 $n \geq 5$ 时，$3(n-3) \geq 2(n-2)$，因此我们应该尽可能地多剪长度为 3 的绳子段。

前面证明的前提是 $n \geq 5$，那么当绳子的长度为 4 呢？在长度为 4 的绳子上剪一刀，有两种可能得结果：剪成长度分别为 1 和 3 的两根绳子，或者两根长度都为 2 的绳子。注意到 $2 \times 2 > 1 \times 3$，同时 $2 \times 2 = 4$，也就是说，当绳子长度为 4 时其实没有必要剪，只是题目要求是至少要剪一刀。

当绳子长度为 2 或者 3 时，其中一段绳子的长度必为 1，做一下特判即可。

综上所述，该算法是正确的。

2.4 The complexity of this algorithm

整个贪心算法中没有使用任何循环语句，只用到了几个条件判断语句，故算法的时间复杂度为 $O(1)$。