tags: Algorithms

Algorithms HW4 Handwritten

2020 Spring | 90899201Y tony20715 黃悟淳

1. Square

1. $\Theta(\log(min(i,j)))$. 若每一次針對 $i*j$ 長方形切割時，皆得出 $j=a*i+b$ 的形式，其中 a 為正整數、 b 為零或正整數，若在 a 大於 1 狀況下，可在一次迴圈執行過程中切下多個 (事實上即 a 個) 邊長為 i 的正方形，則此貪婪演算法等於 Euclid's Algorithm 取 (i,j) 之最大公因數過程：利用 $gcd(i,j)=gcd(i,j-a*i)|a$**為正整數** 的性質，輾轉相除，其中 $j-a*i=b$。本題所求 (最少個正方形個數) 為每次迴圈執行中的 a 係數之加總。由 [http://www.csie.ntnu.edu.tw/~u91029/Divisor.html] (http://www.csie.ntnu.edu.tw/~u91029/Divisor.html%5D) ，了解 Euclid's Algorithm 的時間複雜度為 $\log(\min(i,j))$ 。
2. No. For example, given a 30x36 rectangle, you will obtain one 30x30 square and five 6x6 squares by the previous method, causing a total square count of six (6). However, you can cut the same rectangle into two 18x18 squares and three 12x12 squares, leading to five (5) squares in total and thus making such situation a counterexample of the "least amounts of squares" claim. Idea gained from [https://www.geeksforgeeks.org/paper-cut-minimum-number-squares/] (https://www.geeksforgeeks.org/paper-cut-minimum-number-squares/%5D) .

---

2. Lexicographical order

1.
   - card < cast
   - ntu < ntust
   - ntu > ntnu
2. 由題意可知，數列產生是取決於到訪 (課本稱 discovered) 順序，因此，DFS traversal開始的點 (vertex) 也就是數列中的第一項。為使數列的字典順序最小，第一項應該最小，因此應取key最小的點，在此例便是`key==0`的點。
3.
   1. 由於本題使用無向無循環圖 (undirected acyclic graph)，可以確保兩點之間只有唯一路徑 (path) 連接，則節點數量 $|V|$ 和邊數量 $|E|$ 可用下列關係表示: $|E|=|V|-1$ 。根據題意 (有 n 個節點)，可知 $|V|=n$ 。
   2. 題中 $N(v)$ 即是節點 v 接觸的邊數。由於一個邊必與兩個節點連接，若將所有節點 v的 $N(v)$ 加總，每個邊正好被算兩次，即 $\sum_{i=1}^{n}N(i)=2*|E|$ 。
   3. $N(v)$ 為一個節點接到的邊數，故任一 $N(v)$ 都不會超過整個圖的總邊數 $|E|$，結合(1)可知 $N(v)\leq|E|=|V|-1=n-1$ ，故針對任一 $N(v)$，$\log(N(v))\leq\log(n-1)<\log(n)$。
   4. 綜合上述，可得 $\sum_{i=1}^{n}N(i)\log(N(i))<\sum_{i=1}^{n}N(i)\log(n)=\log(n)\sum_{i=1}^{n}N(i)=2|E|\log(n)=2(n-1)\log(n)\in O(n\log(n))$
4. **演算法:**

   從`key==0`的節點開始執行"Print_and_Sort"：1. 每次輸出(print)自己的key進入結果數列。 2.將與自己相鄰之非父節點根據key由小到大排列(sort)。 3.根據排列順序，針對各相鄰之非父節點執行"Print_and_Sort"。

```
Print_and_Sort (thisNode, parentNode)
    // Put this node into the result, O(1).
    Print (thisNode)
    // If adjacency info is stored by a list,
    // the summation of adjacent nodes of all nodes will be 2n,
    // so the total time complexity of getAdjacentNodes
    // for all nodes is O(2n) = O(n).
    NodeArray array = getAdjacentNodes (thisNode) - parentNode
    // According to the presumption from (3)(3),
    // T(Sort) = O(array.length * log(array.length)).
    // array.length == N(thisNode) - 1.
    Sort (array)
    // Will iterate until all nodes have been
    // executed "Print_and_Sort" exactly once.
    for i=1 to array.length
        Print_and_Sort (array[i], thisNode)

//Execute from Node(1), the smallest node.
Print_and_Sort (Node(1), NULL)
```

正確性:

- key最小的節點放在第一位的數列，其字典序小於任何第一位不是key最小的節點的數列。
- 根據DFS traversal規則，數列的第二位必須是第一個節點的鄰居。
- 選擇第一個節點的鄰居中key最小的節點作為數列的第二位，其字典序將小於任何第二位為其他鄰居的數列。
- 依此類推，每次選擇節點的最小key鄰居，將可獲得最小字典序數列。

時間複雜度:

- 整個程式執行完會使每個節點都執行過一次 Print_and_Sort。
- 每一次執行中，有執行Print: O(1)；Sort: $O(N(v)\log(N(v)))$；和getAdjacentNodes。
- Print的時間複雜度小於Sort，可以忽略。
- 由(3)知道每個節點Sort加總的時間複雜度為 $O(n\log(n))$，大於 getAdjacentNodes 加總的時間複雜度 $O(n)$。
- 故整個演算法的時間複雜度為 $O(n\log(n))$。

---

3. Arithmetic

a. 演算法

Assume the $[a_1..a_n]$ array is named A.

```
sort(A) // Sort the array from the largest to the smallest value, O(nlogn).
cost = 0
for i=1 to n // n iterations
    cost = cost + i * A[i] // O(1)
cost = cost - A[n]
```

cost即為所求。

b. 時間複雜度

- Sort
  - 由課本 Theorem 8.1 及 Corollary 8.2 知，任何比較排序屬於 $\Omega(n\log(n))$，而heapsort和merge sort屬於 $O(n\log(n))$。
  - Sort若使用 heapsort 或 merge sort，其時間複雜度屬於 $\Theta(n\log(n))$。
- cost加總
  - for迴圈執行n次，每次為 $O(1)$，故整體時間複雜度屬於 $O(n)$。
- **總時間複雜度**
  - 取上兩個大者，即 $T(n)=\Theta(n\log(n))$。

(2) 證明為最佳解

由觀察可以發現，總和花費 (total cost) 為step 1的兩個數相加乘以 $(n-1)$ ，加上step 2選的新數字乘以 $(n-2)$，加上step 3選的新數字乘以 $(n-3)$...，加上最後一步step $(n-1)$ 選的新數字乘以 1。 若我們將 $[a_1..a_n]$ 由小到大排序為 $[b_1..b_n]$ ，使得當 $i<j$ 時，可以確保 $b_i \leq b_j$ ，則前述演算法就是將 b_1 和 b_2 先相加，再加 b_3 、 b_4 ...，最後加上 b_n 的總和花費。可以數學式表示為：

$$total\ cost=(b_1+b_2)(n-1)+b_3(n-2)+...+b_n(1)=(b_1+b_2)(n-1)+\sum_{i=3}^{n}b_i(n-i+1)$$

假使此演算法不是最佳解，則我們可以找到至少一組 b_i 和 b_j $(i<j)$ 在上式位子交換，使總和花費差 (denoted as Δ) ，定義為交換後的總和花費減交換前的總和花費，小於 0。反之，若能證明所有交換都無法導致$\Delta<0$，則表示此演算法可以得到最佳解。由總花費差定義可得：

- $\Delta=0$ ，若 $i=1,j=2$ (式1)；
- $\Delta=(b_j-b_1)(j-2)$ ，若 $i=1,j\geq 3$ (式2)；
- $\Delta=(b_j-b_2)(j-2)$ ，若 $i=2,j\geq 3$ (式3)；
- $\Delta=b_i((n-j+1)-(n-i+1))+b_j((n-i+1)-(n-j+1))$ ，若 $i\geq 3,j>i$ (式4)。

式2、3：

- 由定義， $j\geq 3>2>1$ 可得 $b_j\geq b_2\geq b_1$ ，得 $b_j-b_1\geq 0$ 及 $b_j-b_2\geq 0$。
- 因 $j\geq 3$，$j-2>0$。
- 因此，$\Delta\geq 0$。

式4：

- 化簡得 $\Delta=b_j(j-i)+b_i(-(j-i))=(b_j-b_i)(j-i)$。
- 由於 $j>i$ 使得 $b_j\geq b_i$，$(b_j-b_i)\geq 0$ 且 $(j-i)>0$。
- 因此，$\Delta\geq 0$。

沒有任何其況可以使 $\Delta<0$，得證此演算法之解為最佳解。