COT5405 Algorithm Midterm Solution Key

Yichi Zhang

November 2021

Question 1

Pseudo code (15 pts):

Algorithm 1: Find Change

```
1 FindChange
      Input: target value $N$ in cents, face value of currency
             denominations V[1..m]
      Output: number of currency used for each face value C[1..m]
2     for $i = m \cdots 1$ do
3         $n = \lfloor N/V[i] \rfloor$
4         $C[i] = C[i] + n$
5         $N = N - n * V[i]$
6     end
```

Proof of correctness (15 pts):
Use proof by contradiction. Observe the nature of specific V given. For best solution C^*, if $C^*[1] \geq 5$, then we can always replace 5 \$0.01 coin with a \$0.05 coin and optimise the solution. So we have $C^* \leq 4$; if $C^*[2] \geq 2$, replace 2 \$0.05 coin with a \$0.10 coin; if $C^[3] \geq 3$, replace 3 \$0.10 coin with a \$0.25 coin and a \$0.05 coin. Similar argument can be applied to all but last entry of C^*. If assume the greedy approach above does not generate C^*, then there exists some better solution $C'[1..m]$ s.t. $\Sigma C > \Sigma C'$. Let i be the entry with largest subscript s.t. $C[i] \neq C'[i]$. $C'[i] > C[i]$ is impossible since $\Sigma_{[i..m]}(VC') > N$ due to line 3 of the pseudo code, so $C'[i] < C[i]$. Then $C'[1..(i-1)]$ uses less coin than $C[1..(i-1)]$ for a value $k \cdot V[i]$ larger. This is impossible due to abovementioned arguments on upper bound for each entry of C^*.
Time complexity (10 pts):
$O(N)$. Line 3 will run $O(N)$ times and line 2 constant times. Nested loop altogether runs $O(N)$ times.

Question 2

Pseudo code (15 pts):

Algorithm 2: Find Longest Path of a Binary Tree

```
FindLongestPath
    Input: Binary tree $T$
    Output: Length $l$, Sequence $S[]$ where $\forall S[i] \in L, R$
    if $T.left = null$ and $T.right \neq null$ then
        S.push(R)
        $l = l + 1$
        FLP(T.right)
    end
    if $T.left \neq null$ and $T.right = null$ then
        S.push(L)
        $l = l + 1$
        FLP(T.left)
    end
    if $T.left \neq null$ and $T.right \neq null$ then
        $length_l, sequence_l = FLP(T.left)$
        $length_r, sequence_r = FLP(T.right)$
        if $length_l > length_r$ then
            $S = sequence_l$
            S.push(L)
        else
            $S = sequence_r$
            S.push(R)
        end
        $l = l + 1$
    else
        return l, S[]
    end
```

Proof of correctness (10 pts):
Proof by induction.
Base case: tree with 1 node. Trivial.
Induction step: For simplicity we denote a tree with its root node. Assume some node N in tree T with the longest path from N to a leaf found, and such leaf is L. Then there does not exist a path from T to any of subtree N's leaves that are longer than the path from T to N and then to L. So if L is one of the deepest leaf of tree T, then N must be on the longest path found by such algorithm.
Time complexity (5 pts):
$O(N)$ where N is the size of input tree. $T(N) = 2T(\frac{N}{2}) + O(1)$, by master theorem $T(N) = O(N)$.
For tree with weighted edges, change line 4, 9, and 22 to $l = l + edge.weight$

Question 3

Pseudo code (10 pts):

Algorithm 3: Find Largest Number

```
1  FindLargestNumber
       Input: Array $A[1..n]$
       Output: $Max$ of $A[1..n]$
2      if $A.size = 1$ then
3          return $A[1]$
4      end
5      if $A.size = 2$ then
6          return $max(A[1], A[2])$
7      end
8      if $A[1] < A[n]$ then
9          return $A[n]$
10     else
11         $mid = floor((1 + n)/2)$
12         if $A[1] < A[mid]$ then
13             FindLargestNumber(A[(mid+1)..n])
14         else
15             FindLargestNumber(A[1..mid])
16         end
17     end
```

Proof of correctness (5 pts):
Line 2 through 7 are two base cases. Observe that input is circularly shifted. If shifted 0 positions (no shift at all), then the last one is the largest; if shifted, $A[n]$ should be the direct predecessor of $A[1]$ before shifting, so comparing $A[1]$ and $A[n]$ is sufficient to check if shifted, i.e. $k = 0$.
If shift happened, then the largest one should directly precede the smallest one in A. By doing a binary search for the only place where $A[i] > A[i + 1]$ we can locate the largest item in A.
Time complexity (5 pts):
$T(n) = T(\frac{n}{2}) + O(1)$, by master theorem $T(n) = O(logn)$

Notes

There are several common mistakes in your solution. Please refer to the following entries.

- For Q1, you need to show why this specific set of face values would make greedy strategy work. Many of you knew this question required proof by contradiction, some mentioned other face values may not work with greedy, but that is not quite enough.
- We specifically mentioned before the exam that DP would not be covered

in the midterm. Those using DP on Q1 got points off due to giving an algorithm not efficient enough.

- For Q2, we generally accepted both BFS and DFS.
- If your proof is not a proof by induction, don't call it one, especially for a problem like Q3.
- For Q4, we are generally giving 5 pts on low hanging fruits such as geometric sequence and Fibonacci sequence. Further discussions are awarded more than that. However, just searching and copying the word "canonical" from research articles to your answer would not count.
- Quoting the rubric of the course, "A grade of zero for the entire assignment will be given to all cases of plagiarism and cheating". Please contact Dr. Dobra on that topic.