Name: Sunjun Gu(sgu59@wisc.edu)

Campus ID: 9081574338

Assignment: hw01

Partner: Jiaxin Li(jli2274@wisc.edu)

1. (a)

We need to argue that FindDiameterAndDepth and FindDiameter correctly implement their specifications.

For FINDDIAMETER algorithm, it specifies that when given a nonempty rooted tree, the algorithm would return the diameter of the tree. And this algorithm is based on the FINDDIAMETERANDDEPTH algorithm. Therefore, we should first prove the correctness of the second algorithm, then the correctness of the first algorithm is automatically demonstrated.

For FINDDIAMETERANDDEPTH algorithm, its specification is that when given a nonempty rooted tree, the algorithm would return the diameter and the depth of the tree. The procedure says that if the tree’s root has no children then the diameter and depth of the tree are both 0; if the if the tree’s root has children, let the children become the root of the subtrees and call the FINDDIAMETERANDDEPTH algorithm recursively on every subtrees and get the subtrees’ diameters and depths. The depth(**d**) of the original tree is the largest depth of the subtree plus one. Define d’ as the second largest depth which is 0 when the tree only has one subtree and d’ is one plus the second largest depth of the subtree plus one when the tree has more than one subtrees. The diameter(**D**) of the original tree is the maximum of all the subtrees’ diameters and the sum of d and d’.

**Proof of Correctness**: The proof is by induction on the depth(**d**) and diameter(**D**) of the tree:

Definition:

The depth(**d**) of a nonempty rooted tree T is the largest distance between any vertex and the root;

The diameter(**D**) of T is the largest distance between any pair of vertices.

*Base case:* The base case corresponds to a nonempty rooted tree whose root has no children. Because the tree has no children, the largest distance between any vertex and the root is the distance between root and itself by definition of depth, therefore, the depth is 0; And the largest distance between any pair of vertices is the distance between root and itself by definition of diameter, therefore, the diameter of the tree is 0 as well. In this case the algorithm correctly returns 0 of both the depth and diameter.

*Inductive Step:* Assume the input tree T’s root has children. And for all the subtree’s(**T1,T2…Tk**), the algorithm returns the diameter and depth of the subtree(**Di, di; i=1,2…k**). Because the depth of a nonempty rooted tree T is the largest distance between any vertex and the root. Equally, the depth is one plus the largest distance between any vertex and the children of the root. And the largest distance between any vertex and the children of the root is also maximum of the depth of the all the subtree(**T1,T2…Tk**) whose root is the child of T’s root. By the induction hypothesis, the recursive call correctly returns (**Di, di**) for the subtree input. That is, maximum of the depth of the all the subtree is the largest depth of all the subtrees(**T1,T2…Tk**). The algorithm returns d=1+largest di. To complete the proof, we need to show that the algorithm also returns the diameter of the tree. Let **d’** be the second largest depth of trees. We have two cases:

Case 1: the tree only has one subtree(T1). The only depth of subtrees is the depth of T1(d1). Therefore, the second largest depth of subtrees is 0(d’ = 0). For diameter of T, It is the largest distance between any pair of vertices. We have two cases of the root of T.

(i) If the root of T is concluded, because T has only one subtree, the root must be either the beginning or the end node of the diameter. That is, the diameter is the largest distance between any vertex and the root. It is the same as the definition of the depth. Therefore, in this case, D = d = d+d’(d’ = 0)

(ii) If the root of T is not concluded, the subtrees are separated. Therefore, the diameter of T must be the largest diameter of subtrees(D1,D2…Dk). In this case, only one subtree, D = D1 = max(D1,D2…Dk) (D2=…=Dk = 0)

Combine this two case, D= max(D1,D2…Dk, d+d’).

Case 2: the tree has more than one subtree(T1,T2…Tk). d’ equals to one plus the second largest depth among all the subtrees(T1, T2,…Tk). For diameter of T, it is the largest distance between any pair of vertices. We have two cases of the root of T.

(i) If the root of T is concluded. The diameter must be the largest relatively left part distance from root plus the largest relatively right part distance from root. By the definition of the depth which is the largest distance between any vertex and the root. Therefore, these two part must be the largest and second depth of the T which could be returned by this algorithm and proved before. That is D = d+d’.

(ii) If the root of T is not concluded, and the subtrees are separated. Therefore, the diameter of T must be the largest diameter of subtrees(D1,D2…Dk). In this case, D = max(D1,D2…Dk)

Combine this two case, D= max(D1,D2…Dk, d+d’).

These completes the proof of correctness. That is the FINDDIAMETERANDDEPTH algorithm would return the diameter and the depth of the tree when given a nonempty rooted tree.

And the FINDDIAMETER algorithm applies the FINDDIAMETERANDDEPTH algorithm to output the diameter of the tree. Therefore, both algorithm correctly implement their specifications.

1.(b)

**Running time:** We need to argue that FindDiameterAndDepth and FindDiameter algorithms run in linear time.

Using recursion tree to calculate the FindDiameterAndDepth algorithm running time. The recursion tree is structurally identical to the tree T because we visit each node on the tree once. We go from the root then recursively calls on the subtrees of the root, so every node would be visited once. And for the pseudocode, line 5,6,11,12,14 all in constant time O(1). So only O(1) work would be done for the leaf node. And for the non-leaf node, O(1) + O(number of the node’s children) work would be done. That is, for the tree T’s recursive calls, O(number of the parent-children relation) work would be done. And the total number of the parent-children relation is equal to the number of edges of T (number of edges = number of nodes-1). Therefore, Running time = O(1)\*(number of nodes of T) + O(number of edges of T) = O(number of nodes), which is linear time.

By applies the FindDiameterAndDepth algorithm, the FindDiameter algorithm output the diameter in the same runtime as the FindDiameterAndDepth algorithm which is linear time.

Therefore, both FindDiameterAndDepth and FindDiameter algorithms correctly implement their specifications and run in linear time.

3.(a)

**Algorithm**

**Input:** A perfect binary tree T with n = leaves, where each leaf contains an integer value. The binary tree is stored in an array of length n, A[0, … , n-1]

//应该不是binary tree,应该是n leaves

\*The input is an array of length n, A[0, … , n-1] storing n leaves.

**Output:** (c, B), where c is the smallest number of inversions in the sequence of integers, B is a sorted copy of A.

1: **procedure** Smallest-Inversion-And-Sort(A)

2: **if** n = 1 **then**

3: return (0, A)

4: **else**

5: m

6: (

7: (

//应该是从m—(n-1)

(

8:

9:

10: **if**

11:

12: **else**

13:

14: B

15: **return** (c, B)

where Merge

Input: sorted arrays L[1, . . . ,n] and R[1, . . . ,m] with n, m≥1.

Output: Sort(LR)

where Count-Cross

Input: sorted arrays L[1, . . . ,n] and R[1, . . . ,m] with n, m≥1.

Output: Inv(LR)

We design a recursive algorithm Smallest-Inversion-And-Sort for 3a. The algorithm first checks whether the root is the leaf node. If not, separate the perfect binary tree to the left subtree and right subtree. The algorithm recursively calls itself on the left subtree and right subtree at the root’s left and right and input the relevant sequence of integers. The key to the problem is to compare the number of inversions of the subtrees after swap and the number of inversions not been swapped, and choose the method(swap two subtrees or not) with a smallest number of inversions. Below is the proof of correctness and runtime analysis.

**Proof of correctness:**

We need to prove that when given an array of length n, A[0, … , n-1] storing n leaves of the perfect binary tree, the algorithm could return the smallest number of inversions in the sequence of integers. The proof is by induction.

Definition : An **inversion** in an array A[1, . . . ,n] is a pair (i,j)∈[n]×[n] with i<j and A[i]>A[j].

*Base case:*

The base case corresponds to a one-node tree, that is, the leaf node. There is no inversion and no need to sort one node. In this case, the algorithm correctly returns the smallest inversion number 0 and the original array.

*Inductive Step:*

Assume for a perfect binary tree T with n = (n>1) leaves and the input array A[0,…,n- 1], the algorithm correctly output the smallest number of inversions and a sorted copy of A. Now, Suppose the perfect binary tree T with n = leaves and the input array A[0,…,n]. Divide the tree T to its left and right tree whose root it T’s root’s left and right children. Then, recursively calls on left subtree array L[0,…,(n/2 -1)] (the left subtree has leaves) and right subtree array L[n/2,…,(n-1)]( the right tree has leaves). By the assumption before, the recursively calls return and separately. Then, to get the inversion number we need to use the Count-Cross algorithm to calculate the inversion between the left subtree array and right subtree array, that is Count- Cross(L,R). Next, we need to decide whether or not swap the two subtrees to get the smallest inversion number. Compare the number Count-Cross(L,R) that the subtrees not swap and the number Count-Cross(R,L) that the subtrees swap. Therefore, the smallest inversion number of T is. Because Count-Cross algorithm needs two sorted array, we use Merge algorithm to get the sorted array. (This would not impact our result, because we first calculate the left and right subtrees’ inversion number then sort the array. And for Count-Cross algorithm, the order of the two sorted arrays does not matter). Therefore, for the perfect binary tree with n = leaves and the input array A[0,…,n], the algorithm correctly returns the smallest inversion number and the sorted array of A. This completes the proof of correctness.

**Running time:** We prove the algorithm running time is O(nlogn) using recurrence relation .

According to 9/10 lecture, the Count-Cross and Merge algorithm both run in linear time. And the Smallest-Inversion-And-Sort algorithm have two recursive calls for the half of the original input. The other lines on the algorithm run in constant time. The recurrence relation could be written as T(n)=2\*T(n/2) + O(n) + O(1) and T(1) = O(1). So, the algorithm’s running time is O(nlogn).

3.(b)

**Algorithm**

**Input:** an array of n integers that are leaf values of a perfect binary tree T (A[0, …, n-1]), The undesirability on those internal nodes are stored in the linked notes, in each notes N there is an integer indicating the value of undesirability (**U**), the reference of its left child (**leftChild**) and the reference of its right child (**rightChild**). And the root note R of this whole undesirability tree is the input of this algorithm

**Output:** (c, B), where c is the smallest undesirability in this tree, B is a sorted copy of A.

1: **procedure** Smallest-Undesirability (A, R)

2: **if** n = 1 **then**

3: **return** (0, A)

4: **else**

5: m

6:

7:

8:

9:

10: **if**

11: **if** **then**

12: c

// 这里用一个括号会不会容易看一点（）

13: **else**

14: c

15: **if**

16: **if** **then**

17: c

18: **else**

19: c

20: B

12: **return** (c, B)

