

2-2 The Efficiency of Algorithms¹

Hengfeng Wei (hfwei@nju.edu.cn)

16 March 2018 ~ March 25, 2018

The key points include:

- Correctness of the rotating caliper algorithm for the convex polygon diameter problem.
- Lower bound proof of comparison-based sorting.
- Depth-first and breadth-first traversal of trees.

The Convex Polygon Diameter Problem

Solution

Theorem 1 *For a convex polygon, a pair of vertices determine the diameter.*

Proof

Definition 1 (Line of Support)

Definition 2 (Antipodal)

Fact 1 *Not all vertex pairs are antipodal.*

Proof

Theorem 2 (Yaglom) *The diameter of a convex polygon is the greatest distance between parallel lines of support.*

Proof

Lower Bound for Comparison-based Sorting (UD Problem 6.13)

Prove a lower bound of $O(n \lg n)$ on the time complexity of any comparison-based sorting algorithm.

Understanding the Problem

First of all, this problem is *flawed*, if not wrong. It should be $\Omega(n \lg n)$, instead of $O(n \lg n)$.

Comparison-based Sorting.

Algorithm 1 Calculate the sum of contents of nodes of a tree T at each depth.

```

1: procedure SUM-AT-DEPTH( $r$ )           ▷  $r$ : root of the tree  $T$ 
2:    $r.depth \leftarrow 0$ 
3:    $Q \leftarrow \emptyset$ 
4:   ENQUEUE( $Q, r$ )
5:   while  $Q \neq \emptyset$  do
6:      $u \leftarrow$  DEQUEUE( $Q$ )
7:      $sumAtDepth[u.depth] += u.content$ 
8:     for all child vertex  $v$  of  $u$  do
9:        $v.depth \leftarrow u.depth + 1$ 
10:    ENQUEUE( $Q, v$ )

```

- We only consider the sorting algorithms in which comparison is the only way to obtain order information between elements; and
- Comparisons of elements are the critical operations we care about.

Time complexity $O(n \lg n)$. Algorithms on the inputs of size n .

Solution

Comments

BFS on Tree (UD Problem 4.3)

Write algorithms that solve the following problems by performing *breadth-first traversals* of the given trees. You may assume the availability of a queue Q . The operations on Q include adding an item to the rear, retrieving and removing an item from the front, and testing Q for emptiness.

- Given a tree T whose nodes contain integers, print a list consisting of the sum of contents of nodes at depth 0, the sum of contents of nodes at depth 1, etc.
- Given a tree T , find the depth K with the maximal number of nodes in T . If there are several such K s, return their maximum.

Solution

Tree Traversal (UD Problem 4.2)

- Write an algorithm which, given a tree T , calculates the sum of the depths of all the nodes of T .

Counting sort, radix sort, or bucket sort is not a comparison-based sorting algorithm.

In pancake sorting, we are interested in the minimum number of flips (or technically, prefix reversals).

William H. Gates and Christos H. Papadimitriou. Bounds for sorting by prefix reversal. *Discrete Math.*, 27(1): 47–57, January 1979. ISSN 0012-365X

Algorithm 2 Count the number of nodes of a tree T at each depth.

```

1: procedure NODES-AT-DEPTH( $r$ )           ▷  $r$ : root of the tree  $T$ 
2:    $r.depth \leftarrow 0$ 
3:    $Q \leftarrow \emptyset$ 
4:   ENQUEUE( $Q, r$ )
5:   while  $Q \neq \emptyset$  do
6:      $u \leftarrow$  DEQUEUE( $Q$ )
7:      $nodesAtDepth[u.depth] += 1$ 
8:     for all child vertex  $v$  of  $u$  do
9:        $v.depth \leftarrow u.depth + 1$ 
10:    ENQUEUE( $Q, v$ )

```

Algorithm 3 Calculate the sum of depths of all nodes of a tree T .

```

1: procedure SUM-OF-DEPTH()
2:   return SUM-OF-DEPTH( $T, 0$ )

3: procedure SUM-OF-DEPTH( $r, depth$ )           ▷  $r$ : root of a tree
4:   if  $T$  is a leaf then
5:     return  $depth$ 
6:   else
7:     for all child vertex  $v$  of  $r$  do
8:        $depth \leftarrow depth + \text{SUM-OF-DEPTH}(v, depth + 1)$ 
9:   return  $depth$ 

```

- (b) Write an algorithm which, given a tree T and a positive integer K , calculates the number of nodes in T at depth K .
- (c) Write an algorithm which, given a tree T , checks whether it has any leaf at an even depth.

Solution

References

William H. Gates and Christos H. Papadimitriou. Bounds for sorting by prefix reversal. *Discrete Math.*, 27(1):47–57, January 1979. ISSN 0012-365X.

Algorithm 4 Count the number of nodes in T at depth K .

```

1: procedure NODES-AT-DEPTH()
2:   return NODES-AT-DEPTH( $T, K$ )

3: procedure NODES-AT-DEPTH( $r, k$ )            $\triangleright r$ : root of a tree
4:   if  $k = 0$  then
5:     return 1
6:   else
7:      $num \leftarrow 0$ 
8:     for all child vertex  $v$  of  $r$  do
9:        $num \leftarrow num + \text{NODES-AT-DEPTH}(v, k - 1)$ 
10:    return  $sum$ 

```
