Homework3 for EECS 340

Yu Mi,yxm319

February 26, 2018

1 Sorting Leftover Elements

Question: Suppose that you are given an array consisting of n sorted elements followed by $f(n)$ elements in an arbitrary order, where $f(n) \in O(n^{1-\epsilon})$ for some $\epsilon \in (0, 1)$. Describe a method to sort the array in $O(n)$ time.
Answer: This question describes a scenario similar to the intermediate steps of a merge sort, so that we need to solve this problem similar to the approach of a merge sort. First, we need to sort the leftover elements with merge sort, which takes $O(n^{1-\epsilon} \cdot \log n^{1-\epsilon})$ time. After that, we will need to merge the two sequence (original sorted one and the left over part) into a whole sorted sequence, which takes $O(n)$ time. To show that $O(n^{1-\epsilon} \cdot \log n^{1-\epsilon})$ takes less time than $O(n)$, we assign $g(n) = n^{1-\epsilon} \cdot \log n^{1-\epsilon}$, $h(n) = n$. Thus we have:

$$\lim_{n\to\infty} \frac{g(n)}{h(n)} = \lim_{n\to\infty} \frac{(1-\epsilon) \cdot n^{1-\epsilon} \cdot \ln n}{n \cdot \ln 2} = \lim_{n\to\infty} \frac{(1-\epsilon)^2}{\ln 2} \cdot (n^{-\epsilon} \cdot \ln n + n^{-\epsilon}) = \lim_{n\to\infty} \frac{(1-\epsilon)^2}{\ln 2} \cdot \frac{\ln n + 1}{n^{\epsilon}} = \lim_{n\to\infty} \frac{(1-\epsilon)^2}{\ln 2} \cdot \frac{1}{\epsilon n^{\epsilon}}$$

L'Hospital's rule is used in the second and forth step. The equation above will approach 0 when $n \to \infty$, so that $O(n^{1-\epsilon} \cdot \log n^{1-\epsilon})$ takes less time than $O(n)$, we can conclude that $O(n^{1-\epsilon} \cdot \log n^{1-\epsilon}) + O(n)$ is $O(n)$.

2 Theory: Sorting Algorithm Run-times

2.1 Give a tight asymptotic bound on $f(n)$

Answer: Since the total amount of permutation od n elements is $n!$, and binary encoding will use $\log(n!)$ bits to encode such permutations, our tight asymptotic bound should be

$$\log(n!) = \sum_{i=1}^{n} \log i$$

Such time bound make sense because it is always smaller than $n \log n$, which is the time of fastest sorting algorithm (at least I know).

2.2 Why doesn't the derived lower bound, from the previous part hold for non-comparison-based sorting algorithms like radix sort.

Answer: Since non-comparison-based is not based on comparisons to make a sort, they cannot be viewed as a comparison-based approach where each comparison can be treated as searching for a bit in the permutation. The lower bound derived above only stands for the comparison based sorting algorithm where we use each comparison to compose a 'bit' for the ultimate answer.

3 Post Office Placement

Question: Suppose you are the postmaster in charge of putting a new post office in a small town, where all the houses are along one street, where the new post office should go as well. Let us view this street as a line and the houses on it as a set of real numbers, $\{x_1, x_2, ..., x_n\}$, corresponding to points on this line. To make everyone in town as happy as possible, the location, p, for the new post office should minimize the sum,

$$\sum_{i=1}^{n} |p - x_i|.$$

Describe an efficient algorithm for finding the optimal location for the new post office, show that your algorithm is correct, and analyze its running time.
Answer: To solve this problem, we can first consider two houses, where we can simply conclude that the best post office position for this two houses is the middle of their address. The we can extend this conclusion to a three house scenario, where the middle point calculated can be seen as having a weight of 2 and the third house have a weight of 1, then we can conclude that the weighted average point is the optimal position for the new post office. Another notable fact is that we can calculate the average point recursively, which will achieve a time complexity of $O(\log n)$. The algorithm can be shown as follow:

Algorithm FindPostOffice(S, n)
Data: An address sequence S, and its length n
Result: The best position for a new post office.
if $n == 1$ **then**
  return $S[n]$
end if
if $n == 2$ **then**
  return $(S[0] + S[1])/2$
end if
if $n\%2 == 0$ **then**
  return $FindPostOffice(S, n/2) + FindPostOffice(S + n/2, n/2)$
else
  $Middle \leftarrow (n-1)/2$
  $LeftPart \leftarrow (Middle/n) \times FindPostOffice(S, Middle)$
  $RightPart \leftarrow (n - Middle/n) \times FindPostOffice(S + Middle, n - Middle)$
  return $LeftPart + RightPart$
end if

4 Sorting Sequences

Question: Suppose we are given a sequence S of n elements, each of which is an integer in the range $[0, n^2 - 1]$. Describe a simple method for sorting S in $O(n)$ time.
Hint: Think of alternate ways of viewing the elements. *Answer*: The approach is based on Radix sort:

Algorithm Radix-sort(S, n)
Data: A unsorted sequence S, and its length n
Result: The sorted sequence S'
buckets $\leftarrow n$ lists
for $i \leftarrow 0$ to n **do**
  digit $\leftarrow$**int**$((S[n])/n)$
  buckets[*digit*].append($S[n]$)
end for
for $i \leftarrow 0$ to n **do**

```
        result ← n lists
        for j ← 0 to length(buckets[i]) do
            result[buckets[i][j]%n].append(buckets[i][j])
        end for
        bucket ← empty list
        for j ← 0 to n do
            bucket.append(result[j])
        end for
        buckets[i] ← bucket
    end for
    result ← empty list
    for i ← 0 to n do
        for number in bucket[i] do
            result.append(number)
        end for
    end for
```

5 Median From Two Lists

Question: Suppose you are given two sorted lists, A and B, of n elements each all of which are distinct. Describe a method that runs in $O(\log n)$ time for finding the median in the set defined by the union of A and B. Note that merging or concatenating the arrays would take $O(n)$ time.
Answer: The approach is based on recursive and dividing:

```
Algorithm GetMedian(A, B, n)
Data: Two sorted list A and B, and their length are both n
Result: The median of the union of A and B.
if n == 1 then
    return (A[0] + B[0])/2
end if
if n%2 == 0 then                                   ▷ Get the median of A and B
    m1 ← (A[n/2] + A[n/2 − 1])/2
    m2 ← (B[n/2] + B[n/2 − 1])/2
else
    m1 ← A[n/2]
    m2 ← B[n/2]
end if
if m1 == m2 then                    ▷ If the median are equal, return either m1 or m2
    return m1
end if
if ( thenm1 < m2)
    if n%2 == 0 then
        return GetMedian(A + n/2 − 1, B, n − n/2 + 1)  ▷ The add operation to a list name is same as
C language which means to shift the pointer to the next n/2 − 1 address.
    else
        return GetMedian(A + n/2, B, n − n/2)
    end if
else
    if n%2 == 0 then
        return GetMedian(B + n/2 − 1, A, n − n/2 + 1)
    else
```

return $GetMedian(B + n/2, A, n - n/2)$
end if
end if

To achieve the $O(\log n)$ time complexity, we need to try divide and recursively solve the problem. First we need to calculate the medians of A and B, and then compare them. If $m1$ and $m2$ are equal, we are simply done. Otherwise, if $m1 < m2$, we can conclude that the median is in the two list, one is from $m1$ to the last element of A and the other one is from the first element of B to $m2$. It is the similar process when $m1 > m2$. When the two arrays have a size of 1, we can simply calculate their average as the result.

6 Warm-up: Graphs

6.1 R-7.1

Question: Suppose we have a social network with members A, B, C, D, E, F, and G, and the set of friendship ties,

$$\{(A, B), (B, C), (C, A), (D, E), (F, G)\}.$$

Where are the connected components?
Answer: The connection relationship can be represented by the Fig.1.

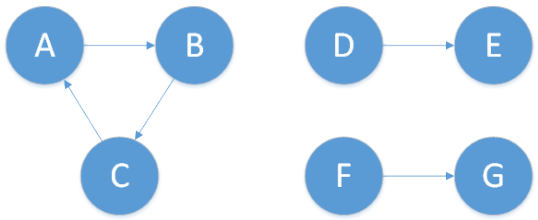

Figure 1: Relation graph

From the figure we can conclude that the connect components are (A, B, C), (D, E) and (F, G).

6.2 R-13.2

Question: Let G be a simple connected graph with n vertices and m edges. Explain why $O(\log m)$ is $O(\log n)$.
Answer: When we have m vertices, the maximum number of edges could be $m < n(n-1) < n^2$, which means $O(\log m) < O(\log n^2) = O(2 \log n) = O(\log n)$. To sum up, $O(\log m)$ is $O(\log n)$.

6.3 Given the graph

6.3.1 Draw the graph where all nodes are annotated with the order that they are visited in a depth-first search from A.

Answer: The result is shown in Fig. 2.

6.3.2 Draw a representation of the tree generated by the DFS annotated over the previous part.

Answer: The result is shown in Fig. 3.

7 Application: Chess

Answer:

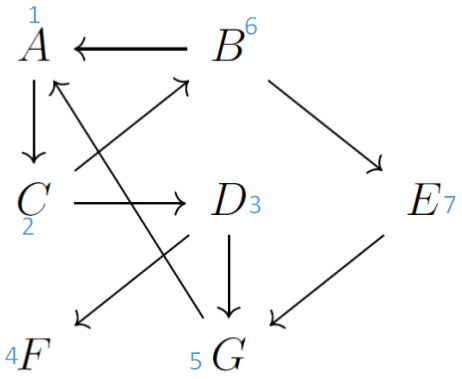

Figure 2: Annotated graph

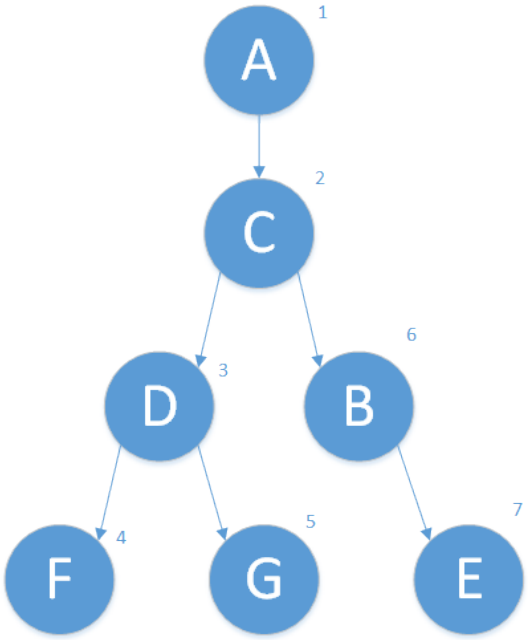

Figure 3: Tree graph

8 Application: A World of Voxels

Answer:

9 Practice: Topological Sorting

Answer:

10 EECS 454 only

10.1 C-17.4

Question: Consider the problem **DNF-SAT**, which takes a Boolean formula S in disjunctive normal form (DNF) as input and asks whether $S\wedge$ is satisfiable. Describe a deterministic polynomial-time algorithm

for **DNF-SAT**.

10.2 C-17.5

Question: Consider the problem **DNF-DISSAT**, which takes a Boolean formula S in disjunctive normal form (**DNF**) as input and asks whether S is dissatisfiable, that is, there is an assignment of Boolean values to the variables of S so that it evaluates to 0. Show that **DNF-DISSAT** is *NP*-complete.