4005-800 Algorithms

Homework 3

Christopher Wood

April 10, 2012

PROBLEM 1. *CLRS 22.1-1*

Solution.

Given an adjacency list representation of a *directed* graph, the only way to determine the adjacent vertices of each vertex $v \in V$ is to traverse the entire adjacency list of v. Using this fact, we can easily determine the time complexity of computing out-degree and in-degree of every vertex as follows.

1. To compute the out-degree for a single vertex $u \in V$, we must count the total amount of vertices contained within the adjacency list of u. To do this, we must traverse the the adjacency list for u, which amounts to traversing all outgoing edges starting from u as well. Therefore, in order to compute the out-degree of every vertex in a directed graph, we must repeat this procedure for every vertex, which means that we will traverse over every vertex and every edge in the graph. Thus, the time complexity is to compute the out-degree of every vertex is $\Theta(V + E)$.

2. To compute the in-degree for a single vertex $uinV$, we must inspect all adjacency lists for every vertex $v \in V$ to determine if u is adjacent to v. Only after a complete traversal of the entire adjacency list representation can we be certain that we have examined all possible edges leading to u, and thus can compute the in-degree. A naive approach to extend this to all vertices would be to repeat this search procedure V times, amounting in a time complexity of $O(V(V + E))$. However, if we use an auxiliary data structure to keep track of the in-degree of every vertex $u \in V$, we need only perform this adjacency list traversal once, incrementing the in-degree of each vertex v that is visited in the traversal. Therefore, just as with the out-degree calculation, the time complexity is simply $\Theta(V + E)$.

PROBLEM 2-a. *CLRS 22.2-2*

Solution.

After running the breadth-first search on the undirected graph shown in Figure 22.3 in the textbook (using vertex u as the source) we arrive at the following values for d and π.

Vertex	d	π
r	4	s
s	3	w
t	1	u
u	0	NIL
v	5	r
w	2	t
x	1	u
y	1	y

PROBLEM 2-b. *CLRS 22.2-3*

Solution.

The purpose of the gray color in BFS is to indicate vertices that have been discovered in the traversal. Without this color, we are guaranteed that the only vertices added to the queue Q are those that are white when visited during the traversal of a vertex adjacency list. Furthermore, the only way to avoid being added to the queue Q is for the vertex under consideration to be colored black. Therefore, without the intermediate gray color, it is possible for a vertex v to be added to Q multiple times before its adjacency list is completely traversed and it is assigned a color black. We now show that duplicate entries in Q do not change the behavior or result of the BFS procedure.

Since the queue is still modified as normal, we are guaranteed that each vertex enqueued into Q will be done so in a breadth-first manner, and that upon dequeuing a vertex v from Q we are guaranteed to enqueue all white-colored vertices adjacent to v before dequeuing the next vertex from Q. Therefore, if a vertex v is duplicated in Q, it is guaranteed that all vertices processed between the first instance of v and the second instance of v will have their neighbors enqueued and then be colored black, so upon the second visit to v, all black-colored adjacent vertices will not be re-added to the queue and all white-colored adjacent vertices will be re-added again (making more duplicates). However, these vertices have already been added in the appropriate order to generate a breadth-first tree, so the duplicates will not change the resulting breadth-first tree. Furthermore, since we are always guaranteed to color a vertex black after enqueuing its neighbors into Q, we will not run into an infinite loop enqueuing and dequeuing duplicate vertices from Q.

PROBLEM 2-c. *CLRS 22.2-5*

Solution.

In the correctness proof provided in the textbook, it is shown that $u.d = \delta(s, u)$, meaning that $u.d$ is always equal to the length of the shortest path between s and u upon termination of the BFS routine. Furthermore, the proof goes on to show that the BFS routine will always produce the shortest path lengths between a start vertex s and all other vertices $u \in V$ for any graph G

without assuming any particular order of the adjacency lists. This is intuitively true since the order of vertices in the adjacency list representation of a graph G does not have any effect on the topolgy of G (i.e. the actual edges that exist in the graph). Therefore, we can conclude that the value $u.d$ assigned to a vertex u is independent of the order in which the vertices appear in each adjacency list for G.

In Figure 22.3 from the textbook, we see that t must precede u in the adjacency list for w. However, if we swap the position of t and x in the adjacency list for w, a BFS traversal will yield the edge (x, u), rather than (t, u), which is a different different breadth-first tree. This difference occurs because the vertices adjacent to x will be enqueued in Q before the vertices adjacent to t because x is visited first in the adjacency list. Therefore, the predecessor of u will be x, not t.

PROBLEM 3. *CLRS 22.3-7*

Solution.

The code for the DFS algorithm that uses a stack for its depth-first traversal is shown below:

ALGORITHM 1: StackDFS

```
1:  for each vertex u ∈ G.V do
2:      u.color = WHITE
3:      u.π = NIL
4:  end for
5:  time = 0
6:  S = makeStack()
7:  for each vertex u ∈ G.V do
8:      if u.color == WHITE then
9:          S.push(u)
10:         while S.notEmpty() do
11:             time = time + 1
12:             v = S.top()
13:             S.pop()
14:             v.d = time
15:             v.color = GRAY
16:             for each vertex w ∈ G.Adj[v] do
17:                 if w.color == WHITE then
18:                     S.push(w)
19:                 end if
20:             end for
21:             v.color = BLACK
```

22: $time = time + 1$
23: $v.f = time$
24: **end while**
25: **end if**
26: **end for**

PROBLEM 4-a. $T(1) = 1, T(n) = aT(n-1) + bn$

Solution.

To solve this recurrence relation using the iteration method, we first expand the recursive calls in order to identify a pattern, as shown below:

$$\begin{aligned} T(n) &= aT(n-1) + bn \\ &= a(aT(n-2) + b(n-1)) + bn \\ &= a^2T(n-2) + ab(n-1) + bn \\ &= a^2(aT(n-3) + b(n-2)) + ab(n-1) + bn \\ &= a^3T(n-3) + a^2b(n-2) + ab(n-1) + bn \\ &= ... \\ &= a^kT(n-k) + a^{k-1}b(n-(k-1)) + ... + ab(n-1) + bn \end{aligned}$$

Now, if we let $k = (n-1)$, we will reach the end of these recursive calls and end up the following result:

$$\begin{aligned} T(n) &= a^{n-1}T(n-(n-1)) + a^{n-2}b(n-(n-2)) + ... + ab(n-1) + bn \\ &= a^{n-1} + b\sum_{i=0}^{n-2} a^i(n-i) \end{aligned}$$

Now we make the observation that $a^n(\frac{1}{a})^{n-i} = a^i$, so we can re-write the summation above as $a^n \sum_{i=0}^{n-2}(\frac{1}{a})^{n-i}(n-i)$, which is the same as $a^n\Big((\sum_{i=0}^{n} i(\frac{1}{a})^i) - (\frac{1}{a})\Big)$. We now have the following:

$$\begin{aligned} T(n) &= a^{n-1} + a^n b\Big((\sum_{i=0}^{n} i(\frac{1}{a})^i) - (\frac{1}{a})\Big) \\ &= a^{n-1} + a^n b\Big(\frac{n(\frac{1}{a})^{n+2} - (n+1)(\frac{1}{a})^{n+1} + (\frac{1}{a})}{((\frac{1}{a})-1)^2} - (\frac{1}{a})\Big) \end{aligned}$$

$$
\begin{aligned}
&= a^{n-1} + \frac{a^n bn(\frac{1}{a})^{n+2} - a^n b(n+1)(\frac{1}{a})^{n+1} + a^n b(\frac{1}{a})}{((\frac{1}{a})-1)^2} - (\frac{1}{a}) \\
&= a^{n-1} + \frac{bn(\frac{1}{a^2}) - b(n+1)(\frac{1}{a}) + a^{n-1}b}{((\frac{1}{a})-1)^2} - (\frac{1}{a})
\end{aligned}
$$

By another observation we can see that the $((\frac{1}{a})-1)^2$ term in the demoniator of the expression above can be discarded to obtain an upper bound on T(n). Thus, we have the following:

$$
T(n) < a^{n-1} + bn(\frac{1}{a^2}) - b(n+1)(\frac{1}{a}) + a^{n-1}b - (\frac{1}{a}) = (b+1)a^{n-1} + bn(\frac{1}{a^2}) - b(n+1)(\frac{1}{a}) - (\frac{1}{a})
$$

Therefore, by discarding the lowest terms in this expression for $T(n)$, we can conclude that $T(n) = O(a^n)$.

PROBLEM 4-b. $T(1) = 1, T(n) = aT(n-1) + bn\ log(n)$

Solution.

To solve this recurrence relation using the iteration method, we first expand the recursive calls in order to identify a pattern, as shown below:

$$
\begin{aligned}
T(n) &= aT(n-1) + bn\ \log(n) \\
&= a(aT(n-2) + b(n-1)\ \log(n-1)) + bn\ \log(n) \\
&= a^2T(n-2) + ab(n-1)\ \log(n-1) + bn\ \log(n) \\
&= a^2(aT(n-3) + b(n-2)\ \log(n-2)) + ab(n-1)\ \log(n-1) + bn\ \log(n) \\
&= a^3T(n-3) + a^2b(n-2)\ \log(n-2) + ab(n-1)\ \log(n-1) + bn\ \log(n) \\
&= ... \\
&= a^kT(n-k) + a^{k-1}b(n-(k-1))\ \log(n-(k-1)) + ... + ab(n-1)\ \log(n-1) + bn\ \log(n)
\end{aligned}
$$

Now, if we let $k = (n-1)$, we will reach the end of these recursive calls and end up the following result:

$$
\begin{aligned}
T(n) &= a^{n-1}T(n-(n-1)) + a^{n-2}b(n-(n-2))\ \log(n-(n-2)) + ... + ab(n-1)\ \log(n-1) + bn\ \log(n) \\
&= a^{n-1}T(1) + a^{n-2}b(n-(n-2))\ \log(n-(n-2)) + ... + ab(n-1)\ \log(n-1) + bn\ \log(n) \\
&= a^{n-1} + b\sum_{i=0}^{n-2} a^i(n-i)\ \log(n-i)
\end{aligned}
$$

Now we make the observation that $a^n(\frac{1}{a})^{n-i} = a^i$, so we can re-write the summation above as

$a^n \sum_{i=0}^{n-2} \frac{1}{a}^{n-i}(n-i)\ \log(n-i)$, which is less than $a^n \sum_{i=0}^{n-2}(n-i)\ \log(n-i)$. Furthermore, we make the observation that $\sum_{i=0}^{n-2}\ \log(n-i) < \sum_{i=0}^{n-2}\ \log(n)$, which means we that $a^n \sum_{i=0}^{n-2}(n-i)\ \log(n-i) < a^n\ \log(n) \sum_{i=0}^{n-2}(n-i)$. We now have the following:

$$\begin{aligned} T(n) < a^{n-1} + a^n b\ \log(n) \sum_{i=0}^{n-2}(n-i) &= a^{n-1} + a^n b\ \log(n)\Big(\frac{1}{2}(n-1)(n+2)\Big) \\ &= a^{n-1} + \frac{a^n b\ \log(n)}{2}(n^2+n-2) \end{aligned}$$

Therefore, by discarding the lowest terms in this expression for $T(n)$, we can conclude that $T(n) = O(a^n\ \log(n) n^2)$.

PROBLEM 4-c. $T(1) = 1, T(n) = aT(n-1) + bn^c$

Solution.

To solve this recurrence relation using the iteration method, we first expand the recursive calls in order to identify a pattern, as shown below:

$$\begin{aligned} T(n) &= aT(n-1) + bn^c \\ &= a(aT(n-2) + b(n-1)^c) + bn^c \\ &= a^2T(n-2) + ab(n-1)^c + bn^c \\ &= a^2(aT(n-3) + b(n-2)^c) + ab(n-1)^c + bn^c \\ &= a^3T(n-3) + a^2b(n-2)^c + ab(n-1)^c + bn^c \\ &= ... \\ &= a^kT(n-k) + a^{k-1}b(n-(k-1))^c + ... + ab(n-1)^c + bn^c \end{aligned}$$

Now, if we let $k = (n-1)$, we will reach the end of these recursive calls and end up the following result:

$$\begin{aligned} T(n) &= a^{n-1}T(n-(n-1)) + a^{n-2}b(n-(n-2))^c + ... + ab(n-1)^c + bn^c \\ &= a^{n-1}T(1) + a^{n-2}b(n-(n-2))^c + ... + ab(n-1)^c + bn^c \\ &= a^{n-1} + b\sum_{i=0}^{n-2} a^i(n-i)^c \end{aligned}$$

Now we make the observation that $a^n(\frac{1}{a})^{n-i} = a^i$, so we can re-write the summation above as $a^n \sum_{i=0}^{n-2} \frac{1}{a}^{n-i}(n-i)^c$, which is less than $a^n \sum_{i=0}^{n-2}(n-i)^c$. Furthermore, we make the observation

that $\sum_{i=0}^{n-2}(n-i)^c < \sum_{i=0}^{n-2} n^c$, which means we that $a^n \sum_{i=0}^{n-2}(n-i)^c < a^n \sum_{i=0}^{n-2} n^c$. We now have the following:

$$\begin{aligned} T(n) < a^{n-1} + a^n b \sum_{i=0}^{n-2}(n)^c &= a^{n-1} + a^n b\Big((n-1)n^c\Big) \\ &= a^{n-1} + a^n b(n^{c+1} - n^c) \end{aligned}$$

Therefore, by discarding the lowest terms in this expression for $T(n)$, we can conclude that $T(n) = O(a^n n^{c+1})$.

PROBLEM 4-d. $T(n) = aT(n/2) + bn^c$

Solution.

To solve this recurrence relation using the iteration method, we assume that $n = 2^k$ and then continue to expand the recursive calls in order to identify a pattern, as shown below:

$$\begin{aligned} T(n) &= aT(n/2) + bn^c \\ &= a(aT(n/2^2) + b(n/2)^c) + bn^c \\ &= a^2T(n/2^2) + ab(n/2)^c + bn^c \\ &= a^2(aT(n/2^3) + b(n/2^2)^c) + ab(n/2)^c + bn^c \\ &= a^3T(n/2^3) + a^2b(n/2^2)^c + ab(n/2)^c + bn^c \\ &= ... \\ &= a^iT(n/2^i) + a^{i-1}b(n/2^{i-1})^c + ... + ab(n/2)^c + bn^c \end{aligned}$$

Now, if we let $i = k$, we will reach the end of these recursive calls and end up the following result:

$$\begin{aligned} T(n) &= a^kT(n/2^k) + a^{k-1}b(n/2^{k-1})^c + ... + ab(n/2)^c + bn^c \\ &= a^kT(1) + a^{k-1}b(n/2^{k-1})^c + ... + ab(n/2)^c + bn^c \\ &= a^k + b\sum_{j=0}^{k-1} a^j(n/2^j)^c \end{aligned}$$

Now we make the observation that $a^j(\frac{n}{2^j})^c = a^j n^c(\frac{1}{2^c})^j = a^j n^c(\frac{1}{2^j})^c = n^c(\frac{a}{2^c})^j$, so we can re-write the summation above as $n^c \sum_{j=0}^{k-1}(\frac{a}{2^c})^j$. We now have the following:

$$\begin{aligned} T(n) &= a^k + n^c b \sum_{j=0}^{k-1} (\frac{a}{2^c})^j \\ &= a^k + n^c b \left(\frac{(\frac{a}{2^c})^k - 1}{(\frac{a}{2^c}) - 1} \right) \\ &= a^k + \frac{bn^c(\frac{a}{2^c})^k}{(\frac{a}{2^c}) - 1} - \frac{bn^c}{(\frac{a}{2^c}) - 1} \end{aligned}$$

Now, we can observe that $a^k + \frac{bn^c(\frac{a}{2^c})^k}{(\frac{a}{2^c})-1} - \frac{bn^c}{(\frac{a}{2^c})-1} < a^k + bn^c(\frac{a}{2^c})^k$. Now, by replacing k with $\lg(n)$, we have the following:

$$\begin{aligned} T(n) < a^{\lg(n)} + bn^c(\frac{a}{2^c})^{\lg(n)} &= n^{\lg(a)} + bn^c n^{\lg(\frac{a}{2^c})} \\ &= n^{\lg(a)} + bn^c n^{\lg(a) - \lg(2^c)} \\ &= n^{\lg(a)} + bn^c n^{\lg(a)-c} \\ &= n^{\lg(a)} + bn^{\lg(a)-c+c} \\ &= (b+1)n^{\lg(a)} \end{aligned}$$

Therefore, we can conclude that $T(n) = O(n^{\lg(a)})$.

PROBLEM 5.

Solution.

The maximum element out of a set of 5 numbers can be found by iteratively applying the following equation.

$$\max(a, b) = \frac{a+b}{2} + |\frac{a-b}{2}| = \frac{a+b+|a-b|}{2}$$

The source code for the $max5$ routine that relies on this equation is shown below.

```
1  def max5(x1, x2, x3, x4, x5):
2          max1 = (x1 + x2 + abs(x1 - x2)) / 2
3          max2 = (max1 + x3 + abs(max1 - x3)) / 2
4          max3 = (max2 + x4 + abs(max2 - x4)) / 2
5          max4 = (max3 + x5 + abs(max3 - x5)) / 2
6          return max4
```