Prof. Sergio A. Alvarez
21 Campanella Way, room 569
Computer Science Department
Boston College
Chestnut Hill, MA 02467 USA

http://www.cs.bc.edu/∼alvarez/
alvarez@cs.bc.edu
voice: (617) 552-4333
fax: (617) 552-6790

CS383, Algorithms
Spring 2009
HW5 Solutions

1. Manually perform depth-first search (DFS) on the graph shown in Fig. 3.2 of the textbook. Explore vertices in alphabetical order whenever there is a choice between two or more vertices. Draw the resulting DFS tree. Label each vertex with its pre and post numbers.

Solution

We discussed the construction of the DFS forest with pre/post labeling in class. The DFS tree for vertex A appears in Fig. 3.4 in the book. A second tree contains vertices K and L. Vertices are visited in the following order; (pre, post) numbers are shown:

$$A(1,20), B(2,13), E(3,8), I(4,7), J(5,6)$$

$$F(9,12), C(10,11)$$

$$D(14,19), G(15,18), H(16,17)$$

$$K(21,24), L(22,23)$$

2. Consider the specific form of the breadth first search algorithm shown in Algorithm 1.

Algorithm 1: Breadth First Search
Input: Graph G, start vertex s of G.
Output: For each vertex v of G that is reachable from s, the number of edges along the shortest path from s to v in G.

```
BFS(G)
(1)    foreach vertex v of G
(2)        dist(v) = ∞
(3)    dist(s) = 0
(4)    queue Q = {s}
(5)    while Q not empty
(6)        v = removeFront(Q)
(7)        foreach vertex w to which there is an edge from v
(8)            if dist(w) = ∞
(9)                insertRear(w, Q)
(10)               dist(w) = dist(v) + 1
```

(a) Provide a detailed running time analysis of Algorithm 1, assuming an incidence matrix representation for the graph G. Recall that the incidence matrix for a graph G with n vertices is an $n \times n$ matrix M (two–dimensional table with n rows and n columns) such that $M_{i,j} = 1$ whenever there is an edge in G from vertex i to vertex j, and $M_{i,j} = 0$ otherwise. You may assume that the queue insertion and removal operations run in time $O(1)$. Express the result of your running time analysis in asymptotic notation, as a function of the number of vertices of the graph and the number of edges.

Solution

The initialization in lines (1)–(2) takes time $O(|V|)$, where $|V|$ is the number of vertices in the graph G. Lines (3) and (4) only require time $O(1)$.

When a vertex w is inserted into the queue in line 9, its distance is immediately set to a finite value in line (10) (the finiteness follows by induction, since it is true initially when the queue contains only s, and line (10) provides the inductive step); hence, this vertex will never be placed in the queue again because of the boolean guard condition on line (8). Because of line (6), only one pass through the outer while loop is made for each vertex in the queue. Combining these facts, we see that the total number of iterations of the while loop is $O(|V|)$. The incidence matrix representation requires $O(|V|)$ checks over vertices w for each v on line (7) just to see whether an edge from v to w exists (the inner loop body (lines (8)-(10)) is actually only entered $O(|E|)$ times, but the damage has already been done by the loop condition). This implies that the total time spent in the loops is $O(|V|^2)$.

In conclusion, the overall running time is $O(|V|^2)$.

(b) Repeat the running time analysis of Algorithm 1, now assuming an adjacency list representation for the graph G. Recall that the adjacency list of a graph G with n vertices is an array L of length n, where the i-th entry $L[i]$ is a linked list whose elements are the vertices j such that there is an edge in G from vertex i to vertex j. Express the result of your running time analysis in terms of the number of vertices of the graph and the number of edges. Give the tightest possible bound in asymptotic notation.

Solution

Most of the running time analysis carries over from the incidence matrix case, except that checking the inner loop condition on line (7) requires only $O(|E|)$ steps combined over all iterations of the outer loop for an adjacency list representation. Thus, the overall running time is $O(|V| + |E|)$.

3. (a) Design an algorithm that takes a directed graph as the input and determines whether the graph is acyclic or not. Follow the description in section 3.3.2 of the book. Use pre and post numbers. Give detailed pseudocode.

(b) Carefully analyze the running time of your acyclicity detecting algorithm from the preceding subtask. Does the asymptotic running time depend on what graph representation is used (incidence matrix or adjacency list)? Explain.

Algorithm 2: Depth First Acyclicity Detection: Single Connected Component
Input: Graph $G = (V, E)$, start vertex v in V.
Output: returns *true* if the connected component of v in G is acyclic, *false* otherwise; sets visited(u) to *true* for each vertex u that is reachable from v in G.

```
isAcyclicComponent(G = (V, E))
(1)   visited(v) = true
(2)   acyclic(v) = true
(3)   foreach edge v → u in E
(4)      if not visited(u) then explore(u)
(5)                        else acyclic(v) = false
(6)   return acyclic(v)
```

Algorithm 3: Depth First Acyclicity Detection: Main Procedure
Input: Graph $G = (V, E)$.
Output: returns *true* if G is acyclic, *false* otherwise.

```
isAcyclic(G = (V, E))
(1)   foreach vertex v in V
(2)      visited(v) = false
(3)   foreach vertex v in V
(4)      if not visited(v)
(5)         if not isAcyclicComponent(v)
(6)            return false
(7)   return true
```

Solutions to (a) and (b)

Just modify the DFS procedures from Fig. 3.3 and Fig. 3.5 of the book as shown in Algorithms 2 and 3. The isAcyclic function is the one that should be called. isAcyclic calls the isAcyclicComponent function once on each connected component of the graph. The running time analysis is identical to that for standard DFS, as discussed in class.