Algorithms - Problem Set 2 Solutions

by Michael Allen

February 20, 2001

1. **Practice with Red-Black Trees**

   (a) The successive Red-Black trees that are constructed by adding the keys 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 into an initially empty tree are shown in figure 1.

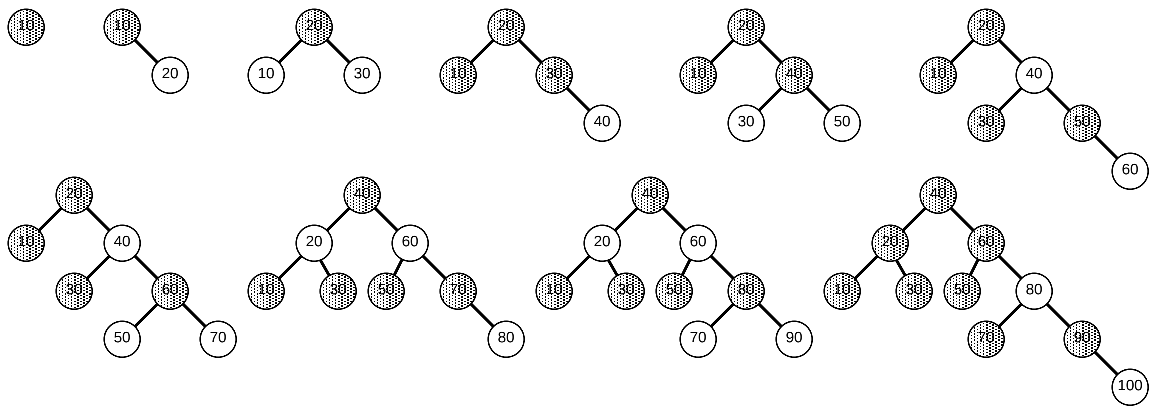


Figure 1: building a red-black tree

   (b) The results of deleting the values 60, 70, and 90 are shown in figure 2. On the left are the results if we do not maintain the red-black property, and the right shows if we do.

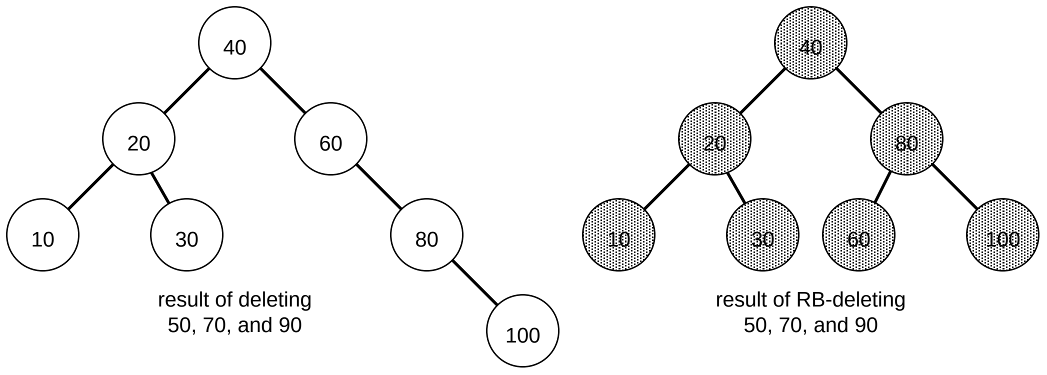


Figure 2: after elements are deleted

2. **Practice with Graph Algorithms**

   (a) The graph shown in figure 3 will yield incorrect results using Dijkstra's algorithm.

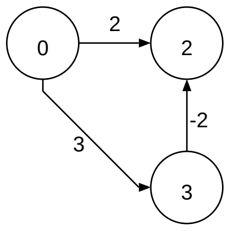


Figure 3: Dijkstra can't handle this graph

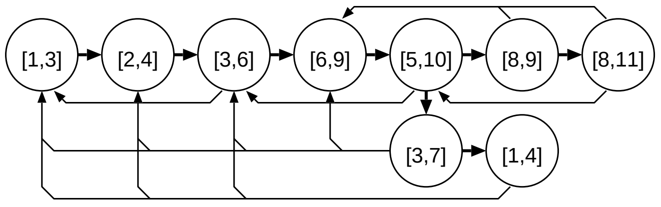


Figure 4: interval graph DFS tree

(b) The depth-first search tree with back edges for the interval graph containing {[1,3], [2,4], [6,9], [5,10], [8,9], [8,11], [3,6], [1,4], [3,7] } is shown in figure 4.

(c) The shortest path tree using Dijkstra's algorithm, and the breadth first search tree for the graph on page 499 of the text are shown in figure 5.

3. **Shortest Path Algorithm**

The BFS-scanning shortest path algorithm will go into an infinite loop on a graph with a negative cost cycle. Nodes that are a part of the cycle will be repeatedly explored as the distances are relaxed more and more.

4. **Topological Sorting**

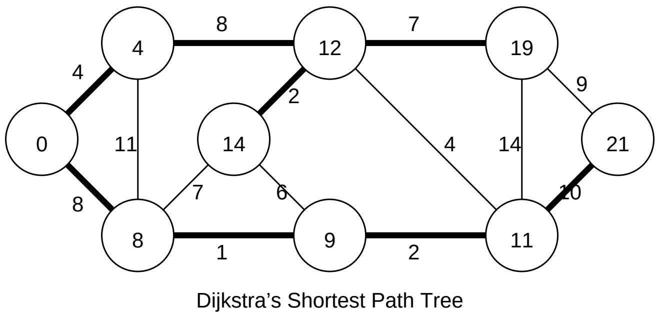
no code yet

5. **Finding a Cycle in an Undirected Graph**

The algorithm below can be used to find and print a cycle in an undirected graph. Works very much like DFS, except that when it encounters a node it has already searched (which only happens when a cycle is present), it prints out the cycle.

```
FindCycle(G)
1. for each vertex u in V[G]
2.   do color[u] <- white
3.      parent[u] <- nil
3. depth <- 0
4. for each vertex u in V[G]
5.   do if color[u] = white
6.        then FC-Visit(u)

FC-Visit(u)
1. color[u] = gray
2. for each v in Adj[u]
3.   do if color[v] = gray and v != parent[u]
4.        then parent[v] = u
5.             FC-PrintOut(v)
```



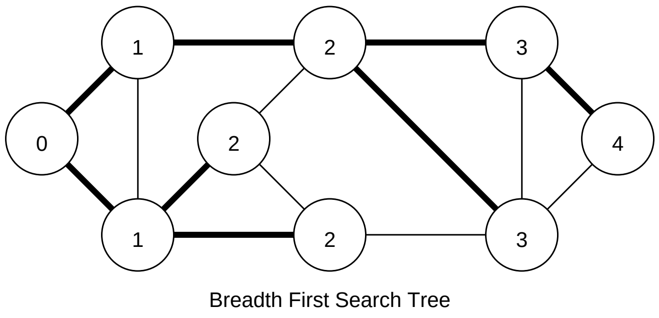


Figure 5: shortest path and search trees

```
6.      if color[v] = white
7.        then parent[v] = u
8.             FC-Visit(v)
9. color[u] = black

FC-PrintOut(v)
1. w <- v
2. do w <- parent[w]
3.    print w
3.   while w != v
4. end
```

6. **Depth First Search**

   no code yet

7. **The Circus Problem**

   Given the heights and weights of a number of performers, the following algorithm will find the tallest stack that can be built with no performer standing on the shoulders of another who is lighter or shorter.

   (a) Add a node to the graph for each performer.

   (b) Examine each pair of nodes and add a directed edge from A to B if A can stand of B's shoulders.

   (c) Add a "start" node with a directed edge to every other node in the graph.

   (d) Find the longest path starting at the start node. (This can be accomplished by assign each edge a weight of -1 and finding the shortest path, since we are guaranteed not to get a negative cycle.)

   (e) Find the farthest node and rebuild the path by following the appropriate edges backwards.