Assignment 4

EECS 2011 – Fundamentals of data structures.

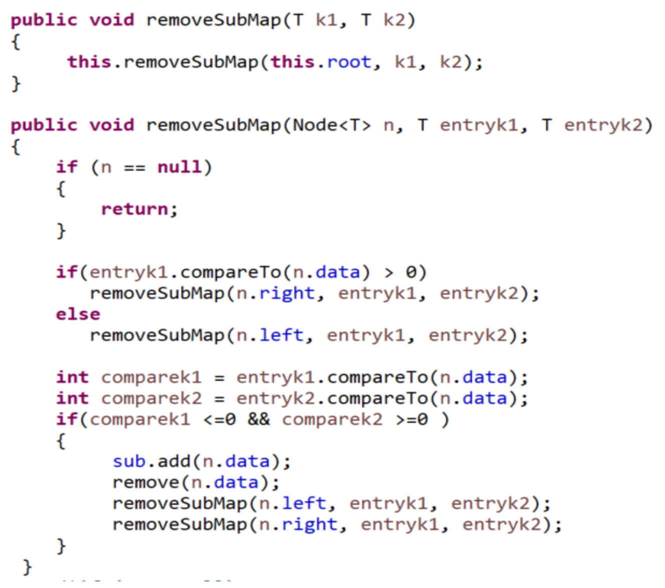
Rajkumar Balakrishnan Lakshmi - 213141197

5/12/17

Problem 1: Remove Sub Map (k1, k2).

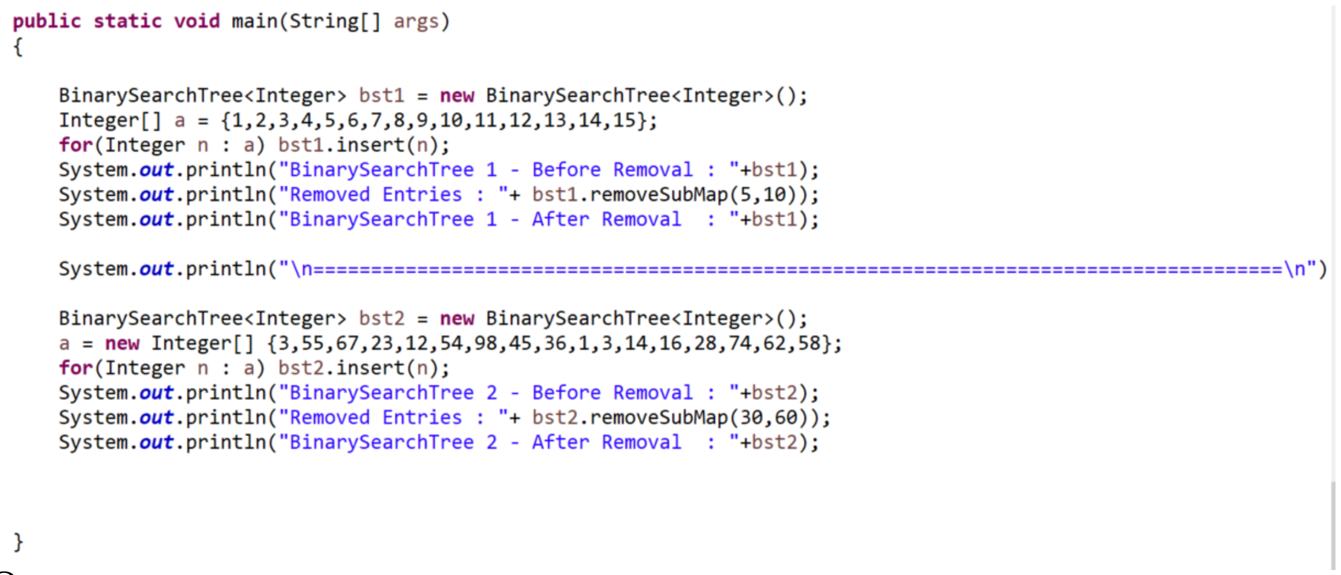
For this question, we are supposed to remove all entries that fall in the range between k1 to k2. I have used an removeSubMap(k1, k2) method that removes the required nodes, in worst-case 0(s+h) where s is the number of entries removed and h is the height of the tree. This method calls an overloaded version (`removeSubMap(Node<T> n, T entryk1, T entryk2)`) of itself with an additional parameter for the root of the tree to be passed to it. We first check if the value of n is null, if it is null we return null otherwise we proceed to the next step. As a next step we compare the value of k1 to the root (n). If the value of k1 is higher than that of the root, we discard the left sub tree and start traversing through the right tree to check for the range (k1 – k2). We do so because of the BST property as the value in the left-hand side of a binary search tree is always smaller than that of the parent or root where as the right-hand side is always greater than that of the parent or root. If the value of k1 is not higher than that of the root, we proceed to check for the range (k1 – k2) in the left sub tree first. In either case if the value of n falls within the range (k1 <= n >= k2) we remove the node. If the tree is unbalanced after the removal, we rebalance it to maintain bst and continue to do this process recursively until we remove all the entries of the tree map that are in the range. Since we compare the range values to the values of the node and skip the unwanted subtrees, the algorithm takes at most of **O (s + h)** of the time. A rough java implementation of the algorithm is attached with this assignment as: RemoveSubRange.java

Code Snippet:



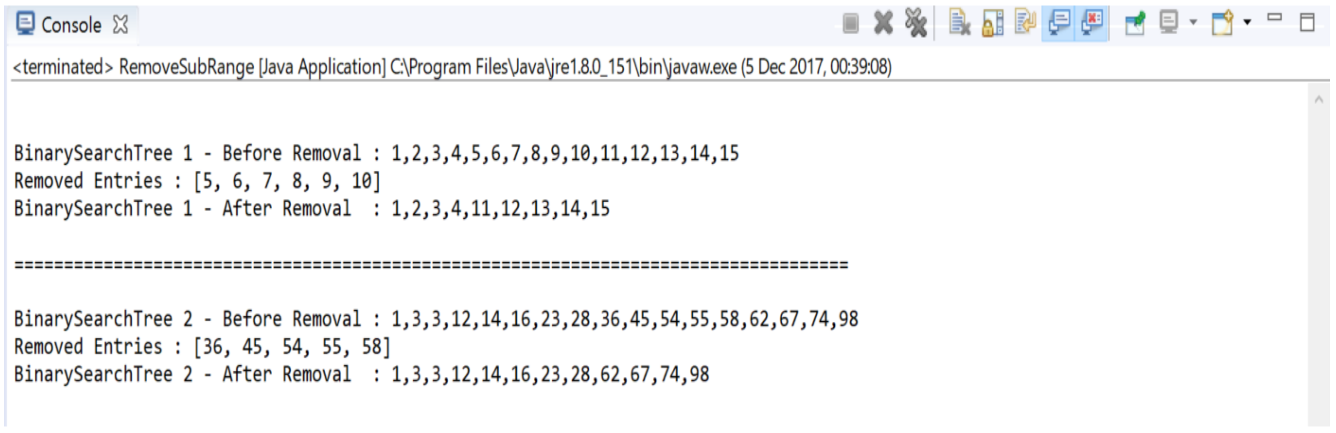
```
public void removeSubMap(T k1, T k2)
{
     this.removeSubMap(this.root, k1, k2);
}

public void removeSubMap(Node<T> n, T entryk1, T entryk2)
{
    if (n == null)
    {
        return;
    }

    if(entryk1.compareTo(n.data) > 0)
       removeSubMap(n.right, entryk1, entryk2);
    else
       removeSubMap(n.left, entryk1, entryk2);

    int comparek1 = entryk1.compareTo(n.data);
    int comparek2 = entryk2.compareTo(n.data);
    if(comparek1 <=0 && comparek2 >=0 )
    {
         sub.add(n.data);
         remove(n.data);
         removeSubMap(n.left, entryk1, entryk2);
         removeSubMap(n.right, entryk1, entryk2);
    }
 }
```

Input:



```
public static void main(String[] args)
{

    BinarySearchTree<Integer> bst1 = new BinarySearchTree<Integer>();
    Integer[] a = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
    for(Integer n : a) bst1.insert(n);
    System.out.println("BinarySearchTree 1 - Before Removal : "+bst1);
    System.out.println("Removed Entries : "+ bst1.removeSubMap(5,10));
    System.out.println("BinarySearchTree 1 - After Removal  : "+bst1);

    System.out.println("\n===================================================================================\n");

    BinarySearchTree<Integer> bst2 = new BinarySearchTree<Integer>();
    a = new Integer[] {3,55,67,23,12,54,98,45,36,1,3,14,16,28,74,62,58};
    for(Integer n : a) bst2.insert(n);
    System.out.println("BinarySearchTree 2 - Before Removal : "+bst2);
    System.out.println("Removed Entries : "+ bst2.removeSubMap(30,60));
    System.out.println("BinarySearchTree 2 - After Removal  : "+bst2);


}
```

Output:



Console

```
<terminated> RemoveSubRange [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (5 Dec 2017, 00:39:08)


BinarySearchTree 1 - Before Removal : 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
Removed Entries : [5, 6, 7, 8, 9, 10]
BinarySearchTree 1 - After Removal  : 1,2,3,4,11,12,13,14,15

===================================================================================

BinarySearchTree 2 - Before Removal : 1,3,3,12,14,16,23,28,36,45,54,55,58,62,67,74,98
Removed Entries : [36, 45, 54, 55, 58]
BinarySearchTree 2 - After Removal  : 1,3,3,12,14,16,23,28,62,67,74,98
```

Problem 2: Sets of Nuts and Bolts.

For this question, we are supposed to find the match pairs of nuts and bolts that belongs to two different sets. This can be achieved through the divide and conquer methodology using the quick sort procedure. The pivot value here is chosen randomly. We choose a random nut and use it to partition the remaining bolts around it. We find the matching bolt for the chosen nut by sorting the partitioned bolts. Then we take the matching bolt and use it to partition and sort the nuts around it into two groups, those with threads larger or smaller than that of the bolt.

Now we are left with sets groups of nuts and two groups of bolts. We repeat this process for each matching pair of nuts and bolts until all the bolts and nuts match up each other. We thus achieved unique pairs of nuts and bolts. Since we are using the divide and conquer methodology using the randomized quick sorting algorithm, the expected running time of the algorithm will be **O (n log n).**

Problem 3: Center of a Graph.

For this question, we are supposed to prove that for every graph g: $radius \leq diameter \leq 2\ radius$ and find the eccentricity, diameter, radius and the center of undirected connected graph G.

a) Prove that for every graph G: $\boldsymbol{radius \leq diameter \leq 2\ radius}$

The radius is the smallest eccentricity of any vertex in graph G and the diameter is the maximum eccentricity of any vertex in G. A center is a vertex whose eccentricity is the radius. Therefore, there must be vertex u whose radius is from a central vertex. Since diameter is the maximum eccentricity of any vertex in G, there can be no two pairs of vertices with their distances greater than that. Hence $\boldsymbol{radius \leq diameter}$**.**

Let’s consider three vertices u, v, c where c is the center vertex of G, such that diameter (u, v) = diameter(G) = |d|. Where d is the shortest path between u an v. Now let d1, d2 be the shortest path from u to c and v to c respectively. Since diameter(G) is the shortest path from d1 to d2, any other path between d1 and d2 is longer. Therefore diameter(G) ≤ d1+d2 and d1+d2 ≤ radius(G) + radius(G) = 2 radius(G) i.e., $\boldsymbol{diameter \leq 2\ radius.}$

Hence $\boldsymbol{radius\ \leq\ diameter \leq\ 2\ radius.}$

b) Find the eccentricity, diameter, radius and the center of a graph.

For this part of the question I have used the BFS algorithm from the lecture slides on graphs.

```
procedure  $BFS(G, s)$
    $i \leftarrow 0$
    $L_i \leftarrow$ new empty queue
$L_i . enque(s)$
$setLabel(s, VISITED)$
while  $\neg L_i . isEmpty()$
    $L_{i+1} \leftarrow$ new empty queue    // next level
    for all  $v \in L_i . elements()$
        for all  $e \in G.incidentEdges(v)$
            if  $getLabel(e) = UNEXPLORED$
                $w \leftarrow opposite(v,e)$
                if  $getLabel(w) = UNEXPLORED$
                    $setLabel(e, DISCOVERY)$
                    $setLabel(w, VISITED)$
                    $L_{i+1} . enque(w)$
                else  $setLabel(e, CROSS)$
    $i \leftarrow i + 1$     // start next level exploration
end-while
```

Now to find the eccentricity, diameter, radius and center of the graph, we first find the eccentricity of all the vertices of the graph G and the rest can be obtained from there. For that purpose, we compute the eccentricity of each vertex in the graph by finding the shortest path between the vertex v and the vertex farthest from v using the BFS (Breadth First Search). Thus, we store the eccentricity of all vertices in an array ADT to their respective indexes. With this information being stored the diameter (maximum eccentricity of any vertex), radius (minimum eccentricity of any vertex) and center (eccentricity is radius) can be obtained. A rough java implementation of the algorithm is given below.

Code Snippet:

```
for (int i = 0; i < G.V(); i++)
        eccentricities[i] = eccentricity(i);

      eccentricity(v)
       {
         BFS(G,v);
         ec = 0;
         for (int i = 0; i < BFS.distTo.length(); i++){
             if(BFS.hasPathTo(i)){
                 if( BFS.distTo(i) > ec) ec = BFS.distTo(i);
             }
         }
         return ec;
      }

     diameter()
      {
        diameter = eccentricities[0];
        for (int i = 1; i < eccentricities.length ; i++)
            if( eccentricities[i] > diameter ) diameter = eccentricities[i];
        return diameter;
      }

     radius()
      {
        radius = eccentricities[0];
        for (int i = 1; i < eccentricities.length ; i++)
            if( exc[i] < radius ) radius = eccentricities[i];
        return radius;
      }

     center()
      {
        center = radius();
        for (int i = 0; i < eccentricities.length ; i++)
            if( radius == eccentricities[i]) return i;
      }
```

Problem 4: Maximum bandwidth

For this question, we are supposed to find the maximum bandwidth between the vertices a and b in graph G. We can achieve this by making few changes to the

Dijkstra's algorithm. Here we use a max priority queue to access it in constant amount of time. And in Dijkstra's algorithm `D[a] = 0 and D[u] = ∞`, in where as here it is the opposite way around. Instead of representing shortest path, D[u] represents the maximum bandwidth of any path from a to u. The maximum bandwidth for path from a through u to v that is adjacent to u is the maximum of D[v] or min(D[u],w(u,v)). The running time of this algorithm is like that of Dijkstra's algorithm since there is only very few changes made to it and the changes made does not affect the running time. So this algorithm runs in $\boldsymbol{O}((\boldsymbol{n} + \boldsymbol{m}) \log \boldsymbol{n})$ time.

```
Algorithm maximimBanwidth(G,a,b)
 {
   Input : A weighted graph G with non-negative edge weights and two distinguishes
vertexes a and b of G.
   Output: Maximum bandwidth between a and b.
   Intialize D[a] = ∞ and D[u] = 0 for each vertex v≠a.
   Let a max priority queue Q contain all the vertices of G using D labels as keys.
   while Q is not empty do
   u = value returned by Q.removeMax()
   if( u = b) return D[u]
   else
   for each vertex v adjecent to u such that v is in Q do
   if min(D[u],w(u,v)) > D[v] then
   D[v] = min(D[u],w(u,v))
 }
```