159.172 Computational Thinking

Tutorial 9: Search Trees

In the lectures, we discussed how to recursively search for a key in a binary search tree by comparing the target key with the key of the node that is currently examined, and depending on the result, either returning the node if the key is found there or recursively search in one of the subtrees if this is not the case:

```
def _bstSearch( self, subtree, target ):
  if subtree is None:
    return None
  elif target < subtree.key:
    return self._bstSearch( subtree.left )
  elif target > subtree.key:
    return self._bstSearch( subtree.right )
  else:
    return subtree
```

Once the correct node is found, we can then extract the value from that node:

```
def valueOf( self, key ):
  node = self._bstSearch( self._root, key )
  assert node is not None, "Invalid map key."
  return node.value
```

If the keys are numeric values, then it sometimes makes sense to look for keys in a certain range. For example, if the keys were prices for items on sale at a garage sale, then one might rather be interested in all items between $1 and $10, rather than items that are on sale for exactly $3.75.

Your task in this tutorial is to extend the binary search method so that it looks for keys in a given range. The range is specified by a lower bound (called **`min`**) and an upper bound (called **`max`**). These are used instead of **`target`** as parameter of **`bstSearch`**. Whenever a node is found with a key in this range, it is added to the list of nodes that are returned by the binary search method.

You are also supposed to change the **`valueOf`** function so that it more adequately deals with the result of **`bstSearch`**. Instead of a single value, the modified function is supposed to return a dictionary with the keys found in the given range and their corresponding values.

Submit your work via Stream by the end of Sunday 11 October.