## 1 Inorder Binary Search Traversal

The basic inorder binary search traversal follows the following implementation:

- Traverse the left subtree by recursively calling the in-order function.
- Display the data part of the root (or current node).
- Traverse the right subtree by recursively calling the in-order function.

There are many ways to implement it such as by recursion, stack, etc. The recursive implementation is very trivial. Following is the method to implement using stacks without recursion:

- Create an empty stack S.
- Initialize current node as root.
- Push the current node to S and set current = current->left until current is NULL.
- If current is NULL and stack is not empty then
  a) Pop the top item from stack.
  b) Print the popped item, set current = popped item->right
  c) Go to step 3.
- If current is NULL and stack is empty then we are done.

[Knuth, 1973]

## 2 Shortest Path

**Dijkstra's algorithm** is an algorithm for finding the shortest paths between nodes in a graph. Textual description of the algorithm is as follows:

Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

- Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.
- Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.
- For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B (through A) will be $6 + 2 = 8$. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.
- When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
- If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
- Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new ćurrent node, and repeat.

[Dijkstra, 1959]

## 3 QuickSort

**Divide :** Partition (rearrange) the array A[p...r] into two (possibly empty) subarrays A[p...q-1] and A[q+1...r] such that each element of A[p...q-1] is less than or equal to A[q], which is, in turn, less than or equal to each element of A[q+1...r]. Compute the index q as part of this partitioning procedure.

**Conquer :** Sort the two subarrays A[p...q-1] and A[q+1...r] by recursive calls to quicksort.

**Combine :** Because the subarrays are already sorted, no work is needed to combine them: the entire array A[p...r] is now sorted.

The following procedure implements QuickSort. [Hoare, 1962]

```
QUICKSORT(A,p,r)
  if $p <$ r
    q = PARTITION(A,p,r)
    QUICKSORT(A,p,q-1)
    QUICKSORT(A,q+1,r)
```

To sort an entire array A, the initial call is QUICKSORT(A,1,A.length).

**Partitioning the array**

The key to the algorithm is the PARTITION procedure, which rearranges the subarray A[p...r] in place.

```
PARTITION(A,p,r)
  $x$ = A[$r$]
  $i = p$ - 1
  for $j = p$ to $r$ - 1 do
    if $A[j] \leq x$
      $i = i + 1$
      exchange A[$i$] with A[$j$]
  exchange A[$i$+1] with A[$r$]
  return $i + 1$
```

## References

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

[Hoare, 1962] Hoare, C. A. (1962). Quicksort. *The Computer Journal*, 5(1):10–16.

[Knuth, 1973] Knuth, D. (1973). Vol. 1: Fundamental algorithms. *World*.