Istanbul Technical University

Analysis of Algorithm II

BLG 336E

BFS - DFS

Homework 1

Tevfik Özgü

150180082

03 April 2021

CONTENT OF TABLE

TABLES

FIGURES

1 INTRODUCTION

In this homework, it will be tried to explain how tree is created and how node assignments are done. Also, pseudocode will be given and the complexity of algorithms will be showed. In the second part the number of visited nodes, maximum number of nodes kept in memory and running times will be given and will be compared. And finally, it will be discussed that why discovered nodes should be maintained.

2 Question 1

In this question, node assignments will be explained, pseudocode will be given and complexity of algorithm will be given.

1.1 Node Assignments

When I was trying to create a tree, which will be worked on, I thought to work on maps which are from STL. So, my decision is to create a root and child nodes map. For better visualization a binary tree is given on Figure 1.

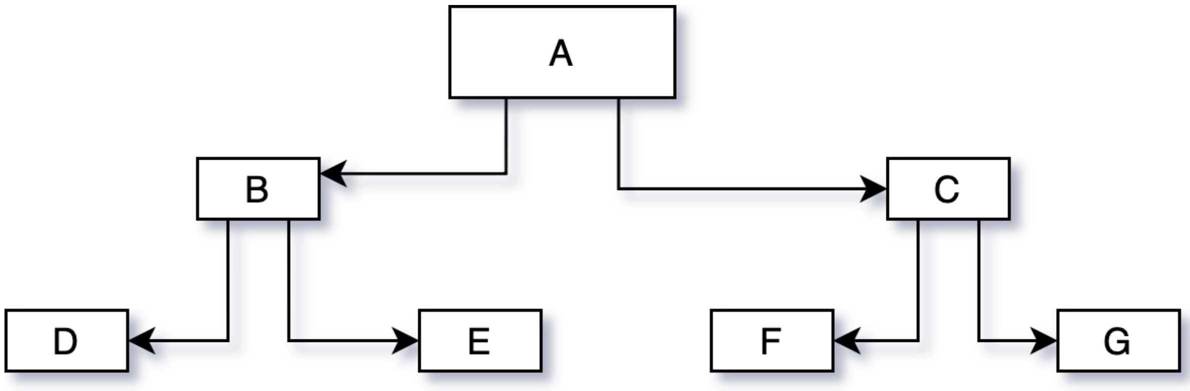

Figure 1. Example Tree

In my assignment, map structure is as given below.

A: [B, C]

B: [D, E]

C: [F, G]

So, as it will be understood, root becomes the key and children become the values of this key. In our question, every node except leaves have 10 children. Every node keeps its node number which will be used as key, its level, its addressed character and its value in its layer.

For example, when inputs and output are typed, I concatted them and remove duplicate characters. So, number of different characters becomes the depth-1 of the tree. First root does not have any number. In the first layer of the tree, values from 0 to 9 is kept which are the possible number of the first character of concatted string. For example, when TWO TWO FOUR is typed, they are concatted and became as TWOFUR and in the first layer T has a chance to be from 0 to 9. In the second layer, there are possible numbers for TW. Children of the root which has value 0 has values from 0 to 9. Children of values which is 1 has values from 10 to 19 and this goes to 99 but duplicate numbers are not taken into account. In the third layer, possible values which are from 0 to 999 is assigned to TWO characters but here also no value has same characters. So, for example 999 is not in the tree.

It is hard to explain with whole tree but a little portion is given at Figure 2. As it can be seen, in the last layer, values are the possible values for TWOFUR.

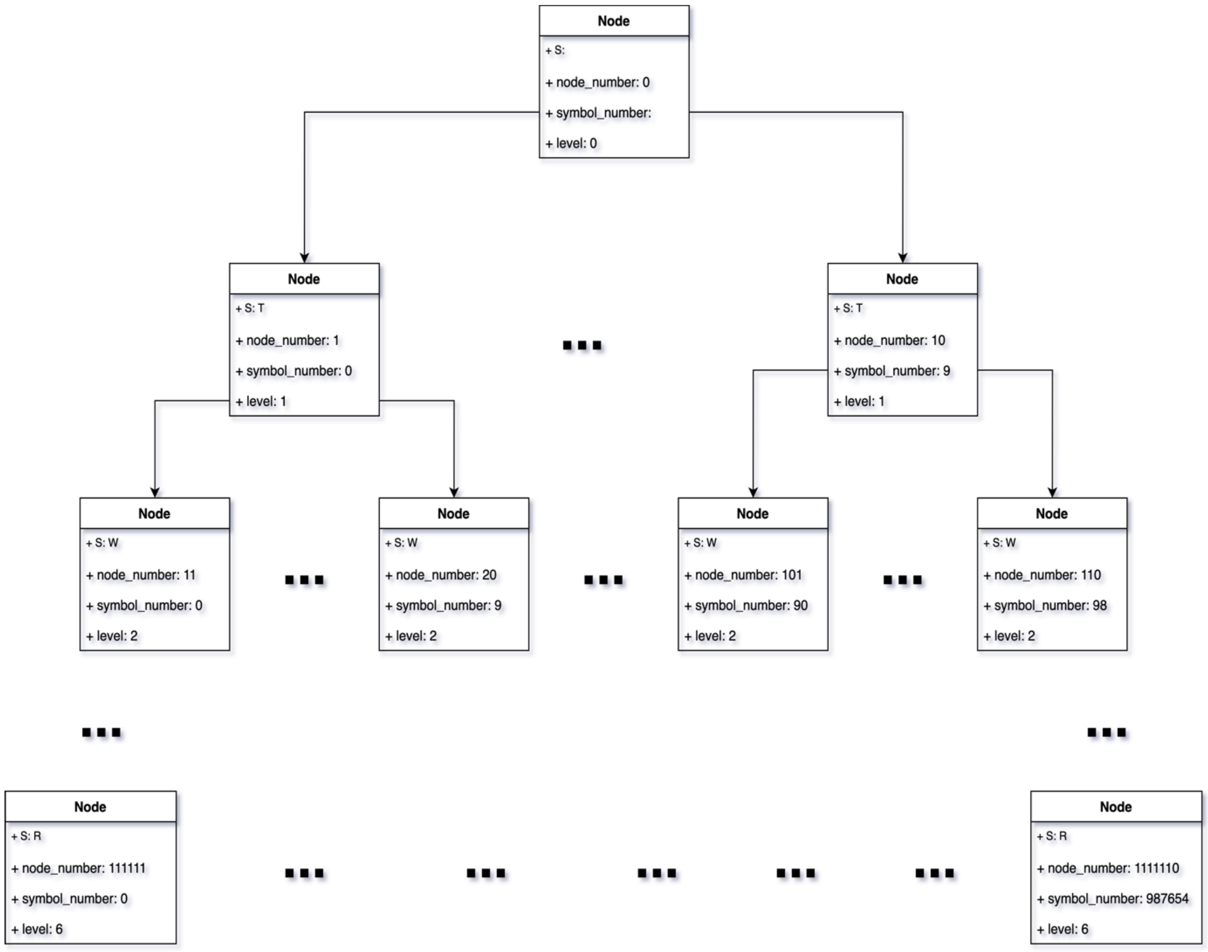

Figure 2. Little Portion of Tree Structure

When this tree is created, real values are searched by using Depth First Search and Breadth First Search. When the symbolic numbers are found according to constraints, they are written into text file and program terminated.

1.2 Pseudo-code

In this part, creating the tree, BFS, DFS and condition checking pseudocodes will be given.

1.2.1 Creating Tree

Table 1. Pseudocode of Creating Tree

```
total_letters_amount <- concatted_string.length()
current_node_number <- 0
curr_Val_Asl <- 0
for i <- 0 to total_letter_amount
  number_of_nodes_in_layer <- pow(10,i)
  for j <- 0 to number_of_nodes_in_layer
    isUnique <- false
    for j <- 0 to number_of_nodes_in_layer
      val = to_string(curr_Val_Asl)
      isUnique <- uniqueCharacters(val)
      if isUnique
        addEdge(current_node_number,new_node)
      curr_Val_Asl <- curr_Val_Asl +1
    if isUnique
      current_node_number<- current_node_number +1
  curr_Val_Asl <- 0
```

1.2.2 DFS (v)

Table 2. Pseudocode of Depth First Search (DFS)

```
visited_node_amount <- visited_node_amount + 1
visited[v.node_number] = true
for i <- adj[node.node_number].begin to adj[node.node_number].end
  if i.level == total_level
    check_condition(i)
  if !visited[i.node_number]
    DFS(i)
```

1.2.3 BFS (v[ ])

Table 3. Pseudocode of Breadth First Search (BFS)

```
queue
for a <- 0 to v.size
  visited[v[a].node_number] = true
  queue.push_back(v[a])
while not queue.empty
  x = queue.front
  x.pop_front
  if queue.size >max_siz
    queue.set_size(max_size)
  if x.level == level_total
    check_cond(x)
  adj[x.node_number].begin to adj[x.node_number].end
    queue.push_back(i)
    if !visited[i.node_number]
      visited[i.node_number] = true
```

1.2.4 check_cond (candidate)

Table 4. Pseudocode of Check Condition Function

```
num = candidate.symbol_number
uu = num.len
for i<-0 to level_total - uu
  num = '0' + num
is_diff = uniq(num)
if is_diff
  for i<-0 to level_total
    letters_map[concatted_String[i]] = num[i]
  for i<-0 to first_input.length()
    let_val = letters_map[first_input[first_input.len - i -1]]
  for i<-0 to second_input.length()
    let_val2= letters_map[second _input[second _input.len - i -1]]
  for i<-0 to output.length()
    out_let= letters_map[output.[output.len - i -1]]
  if let_1 + let_2 == out_let
    write_file(letters_map)
```

1.3 Complexity

For Depth First Search, as seen here, initially we are trying to reach to leaf. When we reach leaf, bad worse scenario is to visiting all the nodes. So, time complexity is **O(n).** For Breadth First Search, worst case scenario is visiting all nodes in the layer. So, like Depth First Search worst case scenario is **O(n).**

As a conclusion, as seen from BFS and DFS pseudocodes, it will be tried to visit all the nodes. So, complexity is O(n) for both BFS and DFS where n is the number of nodes.

3 Question 2

In this question it was asked to compare BFS and DFS according to the number of visited, maximum number of nodes kept in the memory and the running time. Results are given at Table 1.

	Number of Visited Nodes		**Maximum Number of Nodes**		**Running Time**	
SEND MORE MONEY	3199749	2405933	3265919	3265919	20.628384 seconds	18.083672 seconds
TWO TWO FOUR	167536	71808	207177	207177	0.915291 seconds	0.447076 seconds
DOWN WWW ERROR	198243	105466	207177	207177	1.193891 seconds	0.723905 seconds
ALGORITHM:	**BFS**	**DFS**	**BFS**	**DFS**	**BFS**	**DFS**

Table 5. Results of BFS and DFS

As given here, for all examples BFS takes more time because for all examples, as it is known BFS moves depth by depth. When it is in the first layer, it looks all nodes but there is no need to look that nodes since there is not enough values for all characters. So, until it reaches to leaves it actually spends time. But it is not same for DFS. Because it works from leaves to root and it works initially on last layer where we are looking for characters. As it can be seen from Table 1, when DFS is used, it visits less nodes for all examples. Maximum number of nodes is changes according to depth.

4 Question 3

In this question it was asked why discovered nodes should be maintained. The reason is that when a node is visited, if it is the root of any child, it will be checked one more time whether it satisfies conditions or not. So, it will affect the performance of application. Suppose we are at the fifth layer and we are iterating on the child of any node in fifth layer. If we cannot find any node that satisfies conditions, we will recursively look for the root node again and this is not needed. But there is a disadvantage of maintaining discovered nodes. This is because when discovered nodes are kept, it will consume some memory and there is a cost to save discovered nodes. But for graphs which has cycles, discovered nodes MUST be maintained to not iterate on same cycle.

As a conclusion, it is good to maintain discovered nodes and it will increase the performance of program. But moreover, it will be more important to use when the structure is graph. But for the trees, it can affect good or worse since it is good to not turning back to discovered nodes but there is a disadvantage because it will cost some memory and it will be time consuming to keep these children.