BLG336E -Analysis of Algorithms- HW1

Emir Kaan Erdogan

March 27, 2023

Report Structure
Within the scope of the project report, I basically refer to the solution I provide, mention pseudo-codes for the algorithms I utilized . Having completed this task, I give brief information about the complexity analysis as well. At the end, there are 2 questions to answer which are:
1) Why should you maintain a list of discovered nodes? How does this affects the outcome of the algorithms?
2)How does increasing the number of the kids affects the memory complexity of the algorithms? Prove it.

1 Section

A) The code can be compiled and run with the following instructions !
Program Compile Command: g++ -std=c++11 -Wall -Werror main.cpp -o main
./main <input.txt>
In the following the pseudocode will be given and the time complexity of the algorithms is to analyze.
In order to implement the project, first I need to take the input and create an adjacency matrix. To read the input file I utilized a function named "read_file" that creates a vector of pointers to Kid objects, which I need to create the adjacency matrix and take as argument in "create_grid" function. In the following, I provide the pseudo-code for the"**create_grid**" function:
Pseudo-Code for the function creating the Graph

```
create_grid(vector<vector<int»my_grid)
    my_grid <-(kids.size(), kids.size())
    for i in 0 to my_grid.size() - 1 do
     for j in 0 to my_grid[0].size() - 1 do
      if i != j then
       if is_reachable(kids[i], kids[j]) then
        my_grid[i][j] <- 1
        my_grid[j][i] <- 1
write_graph(my_grid)
return my_grid
```

Time Complexity
We are confronting with a nested loop within the function, the outer loop iterates kids.size() times and the inner loop also iterates kids.size() times for each iteration of the outer loop. Therefore, the total number of iterations is kids.size() kids.size()

which gives a time complexity of O(n^2)

BFS Code Explanation
The function named "min_passes" takes grid, source and target as argument and aims to return the number of minimum passes from source to target while obeying the pre-given rules. For this purpose it uses BFS algorithm implemented with the help queue and unordered set data structures. To find the shortest path a queue for kids is utilized as usual and to avoid ambiguous search an unordered set should be added up to the code. In my design, I utilized unordered set data structure instead of boolean array. The pseudo-code is provided below:

Pseudo-Code for the BFS function
```
int min_passes(vector<vector<int>grid,int src,int target)
    search_queue
    visited
    dist <- a new vector of integers with size
    grid.size() initialized to -1
    parent <- a new vector of integers with size
    grid.size() initialized to -1
    search_queue.push(src)
    visited.insert(src)
    dist[src] <- 0
    while search_queue is not empty do
      current_kid <- search_queue.front()
      search_queue.pop()
      if current_kid == target then
         print_parent_path(parent, src, target)<-Print path
         return dist[target]
      for i in 0 to grid[current_kid].size() - 1 do
        (if grid[current_kid][i] == 1 and i is not in visited)
         visited.insert(i)
         search_queue.push(i)
         dist[i] <- dist[current_kid] + 1
         parent[i] <- current_kid
    return -1
```

Time Complexity:
Since the utilized data structures "queue and unordered set" have a constant time complexity for insertion,deletion and access operations their effect on overall time complexity is negligible in comparison with the BFS implementation. Overall, we have O(N^2)$timecomplexitysincewedealwithanadjacency$

Pseudo-Code for the DFS function
```
bool dfs(current,source,grid,visited,path)
    path.push_back(current)
    visited.insert(current)
    if current node is source path.size>2:
        return true
     for i=0 to grid.size()-1
    if current node is source and size>2:
      return true
    if current node is not visited then
      Call DFS recursively.
      if(dfs)
        return true
  path.pop_back()
  return false
```

Time Complexity
Since we visit each cell at most once thanks to the unordered set utilized to keep track of the visited kids, in worst case we visited each cell which results in overall time complexity O(n^2)

2 Questions

2.1 Why should you maintain a list of discovered nodes? How does this affects the outcome of the algorithms?

The reason for maintaining this list is to prevent visiting the same node multiple times. Without maintaining a list of discovered nodes, the program gets stuck in an infinite loop, and visits repeatedly the same nodes, and never finds a solution. In addition, by reducing the number of the redundant traversals it increases the efficiency of the algorithm.

2.2 How does increasing the number of the kids affects the memory complexity of the algorithms?

2.2.1 Show the space complexity of your algorithm on the pseudo-code.

BFS SPACE COMPLEXITY
For BFS, the memory complexity is directly proportional to the number of nodes in the graph or tree. While implementing BFS a queue data structure is utilized to keep track of the nodes to be explored. In the worst case, all the nodes in the graph or tree may need to be stored in the queue at some point during the traversal.

Therefore, the memory required for the queue can be as much as O(N), So, the increasing number of the kids results in increasing memory complexity.

In my algorithm, I use o(n**2) memory for adjacency matrix, and since I do not append the unvisited kids to visited set, it cannot be larger than n which is negligible. And overall it has space complexity O(N^2).

DFS SPACE COMPLEXITY
In a recursive implementation of DFS (depth-first search), the memory complexity depends on the maximum depth of the recursion stack. The depth of the recursion stack is determined by the maximum depth of the tree or graph being traversed. Since the increasing number of kids also increases the maximum depth ,it leads to higher memory requirement.
In my algorithm I use (n*n=n**2) memory for adjacency matrix, and since the number of kids

equals n and we do not traverse through the same nodes again, longest path can only contain n kids in worst case which is negligible. The overall space complexity is O(N^2).

2.2.2 Show run-time of all cases in a graph with number of nodes in x-axis and run-time in y-axis.

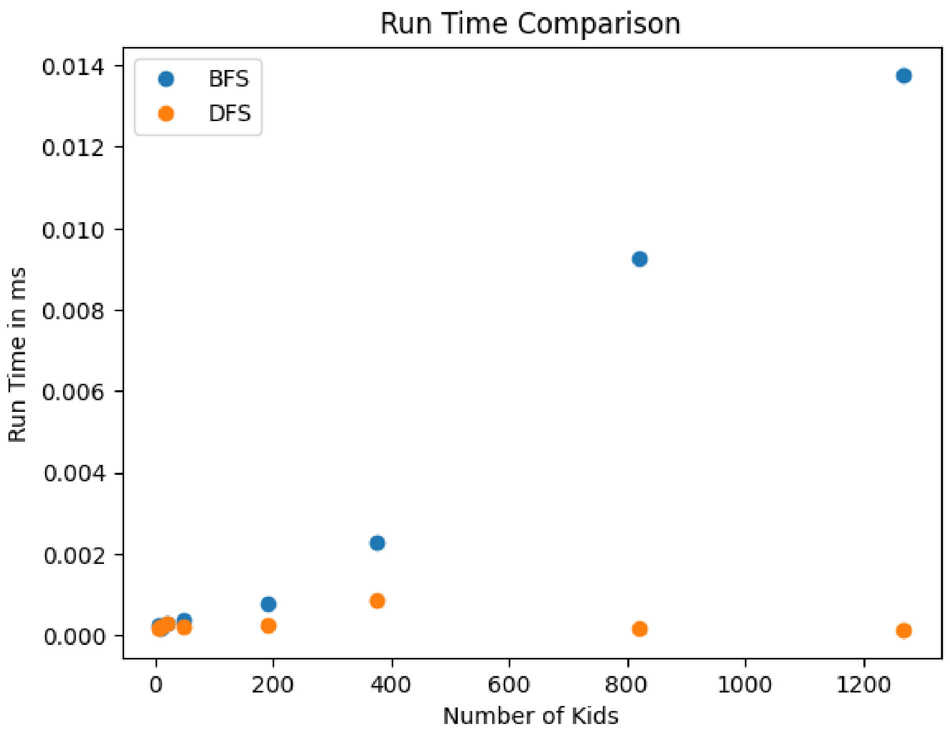

Figure 1: Run Time Graph.

3 References

- Lecture Slides and Recitations
- https://www.youtube.com/watch?v=6ZRhq2oFCuot=338s
- https://www.programiz.com/dsa/graph-dfs