

BLG 336E 1st Project Report

Analysis of Algorithms II



Tuğba Özkal

150120053

18.03.2018

BLG 336E 1st Project Report

Analysis of Algorithms II

1. Problem formulation, state and action representations in detail.

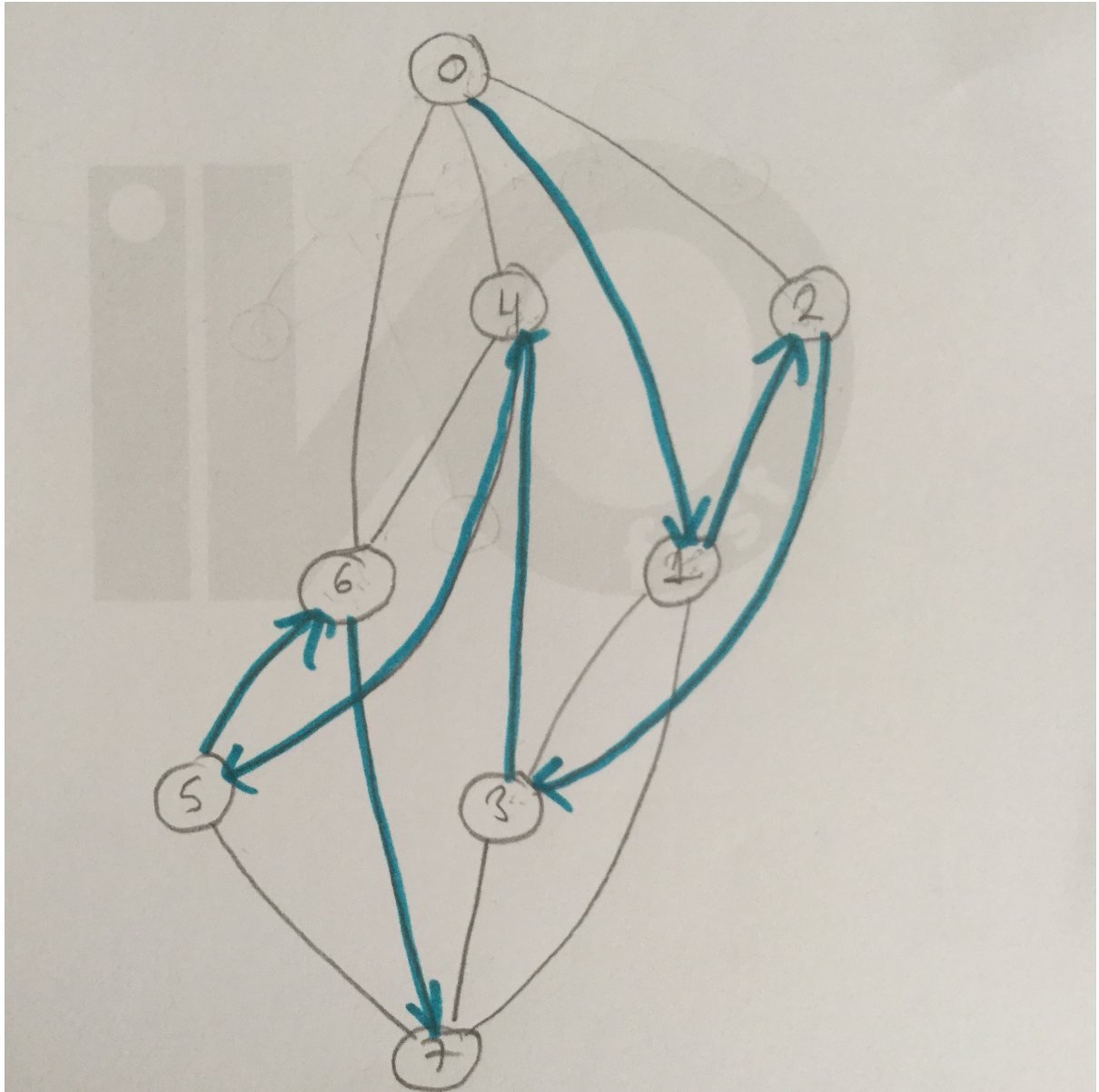
There is 8 nodes in total. This nodes can be explained as:

0. node = (Farmer, Carrot, Rabbit, Fox | | ...)
1. node = (Carrot, Fox | | Farmer, Rabbit)
2. node = (Farmer, Carrot, Fox | | Rabbit)
3. node = (Fox | | Farmer, Carrot, Rabbit)
4. node = (Farmer, Rabbit, Fox | | Carrot)
5. node = (Rabbit | | Farmer, Fox, Carrot)
6. node = (Farmer, Rabbit | | Fox, Carrot)
7. node = (... | | Farmer, Rabbit, Carrot, Fox)

Edge list:

Nodes	Connections			
0	1	2	4	6
1	0	2	3	7
2	0	1	3	
3	1	2	4	7
4	0	3	5	6
5	4	6	7	
6	0	4	5	7
7	1	3	5	6

Farmer should be in the bot in all travel.



8. How does your algorithms work?

```
vector <bool> discovered;
vector <map <int, string> > graph(NODE_COUNT);
vector <string> node;
```

"discovered" vector is used to keep the nodes' visit information as boolean. If the i^{th} node is visited, `discovered[i]` is true, else it is false.

"node" vector is used to keep the node names as string such as "(Farmer, Carrot, Rabbit, Fox || ...)"

"graph" vector is the simulation the real graph. It keeps a map and that map keeps the edges between nodes and those edges' name. For example, node 1 have 4 edge connection to other nodes (0, 2, 3, 7).

```
graph[1][0] = "(Farmer, Rabbit <)"
```

```
graph[1][2] = "(Farmer <)"
```

These connections and vectors are created in "void setup()" function.

DFS algorithm pseudo code:

```
void DFS (int i)
    visited_node_count++
    if node[i] visited
        return
    else
        set node[i] visited
        if (Stack is not empty)
            int j = Stack.pop()
            move_count++
        Stack.push(i)
        for int it = graph[i]->begin; graph[i]->end; it++
            DFS(it)
    return
```

Here, DFS is a recursive function. It calls itself with the node numbers which has connection with i^{th} node. If the node was visited before, visited node count is increased and then function returned. If the node has not been visited before, it is set as visited. The edge between the current node and the previous visited node is found by using stack. Current node number is pushed to stack.

BFS algorithm pseudo code:

```
void BFS (int i)
    visited_node_count++
    set node[i] visited
    for int it = graph[i]->begin; graph[i]->end; it++
        if node[i] is not visited
            move_count++
            DFS(it)
    return
```


Here, BFS is a recursive function. It calls itself with the node numbers which has connection with unvisited i^{th} node. `visited_node_count` is increased in each calls. `move_count` is increased when function called it self.

9. In DFS algorithm, why a list of discovered nodes are maintained?

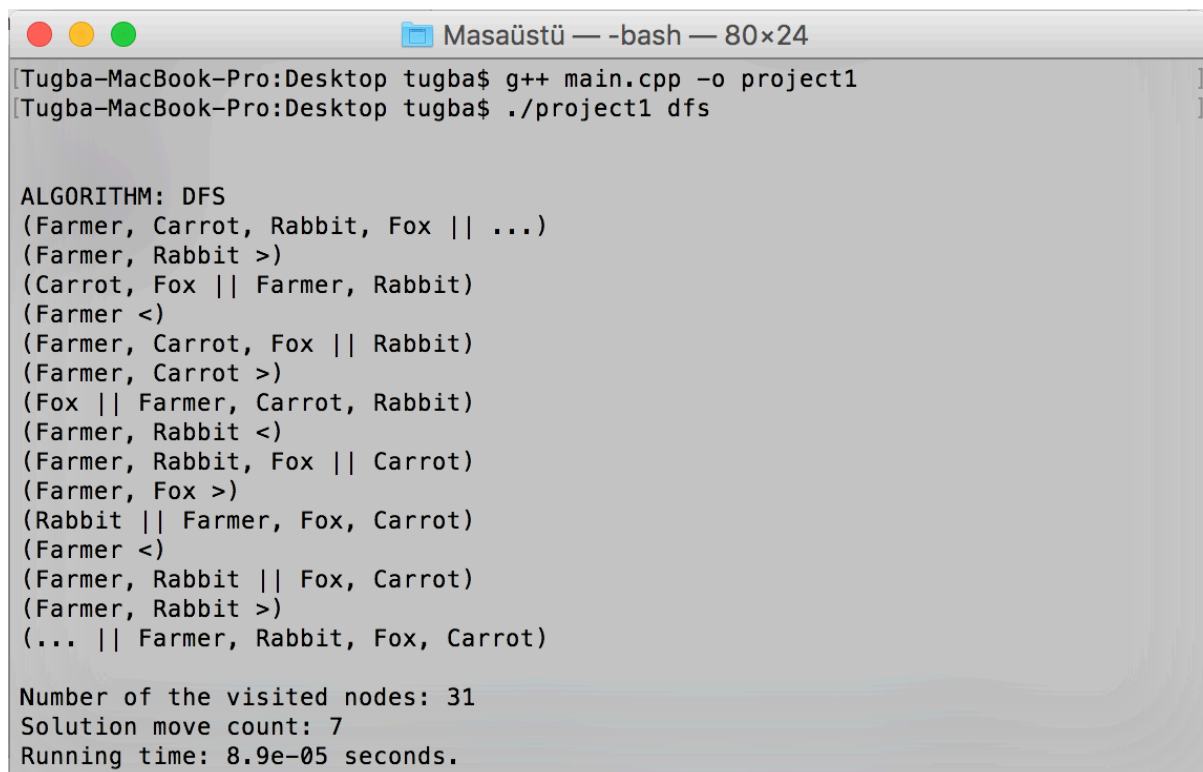
Graph is undirected so, edges have 2 dimensions. So, discovered nodes list should be kept for not visit the discovered nodes again.

10. Is the graph constructed by the given problem formulation a bipartite graph? Why or why not?

Graph is bipartite graph because the farmer should turn back. Farmer makes several trips in both two dimension.

11. Analyze

Number of visited nodes mean trying to visit both visited and unvisited nodes. But move count keeps only the visiting number of unvisited nodes.



```
Masaüstü — -bash — 80x24
[Tugba-MacBook-Pro:Desktop tugba$ g++ main.cpp -o project1
[Tugba-MacBook-Pro:Desktop tugba$ ./project1 dfs

ALGORITHM: DFS
(Farmer, Carrot, Rabbit, Fox || ...)
(Farmer, Rabbit >)
(Carrot, Fox || Farmer, Rabbit)
(Farmer <)
(Farmer, Carrot, Fox || Rabbit)
(Farmer, Carrot >)
(Fox || Farmer, Carrot, Rabbit)
(Farmer, Rabbit <)
(Farmer, Rabbit, Fox || Carrot)
(Farmer, Fox >)
(Rabbit || Farmer, Fox, Carrot)
(Farmer <)
(Farmer, Rabbit || Fox, Carrot)
(Farmer, Rabbit >)
(... || Farmer, Rabbit, Fox, Carrot)

Number of the visited nodes: 31
Solution move count: 7
Running time: 8.9e-05 seconds.
```

```
Masaüstü — -bash — 80×24

[Tugba-MacBook-Pro:Desktop tugba$ ./project1 bfs

ALGORITHM: BFS
(Farmer, Carrot, Rabbit, Fox || ...)
(Farmer, Rabbit >)
(Carrot, Fox || Farmer, Rabbit)
(Farmer <)
(Farmer, Carrot, Fox || Rabbit)
(Farmer, Carrot >)
(Fox || Farmer, Carrot, Rabbit)
(Farmer, Rabbit <)
(Farmer, Rabbit, Fox || Carrot)
(Farmer, Fox >)
(Rabbit || Farmer, Fox, Carrot)
(Farmer <)
(Farmer, Rabbit || Fox, Carrot)
(Farmer, Rabbit >)
(... || Farmer, Rabbit, Fox, Carrot)

Number of the visited nodes: 8
Solution move count: 7
Running time: 7.2e-05 seconds.
```

DFS algorithm takes time about $8.9e-05$ seconds. BFS algorithm takes time about $7.2e-05$ seconds. BFS is faster than DFS but there is no big difference between them.