BLG336E Analysis of Algorithms II, Spring 2022

Project 1

Battleship Game

Submission Deadline: 29.03.2022 23:59

Overview

Breadth-First Search (BFS) and Depth First Search (DFS) are well-known graph traverse algorithms. In this project, you will need to implement BFS and DFS algorithms to implement the Battleship game.

Battleship Game

The object of Battleship is to try to sink all of the other player's ships before they sink all of your ships. All of the other player's ships are somewhere on their board. You try and hit them by calling out the coordinates of one of the squares on the board. The other player also tries to hit your ships by calling out coordinates.

This game has additional game rules, unlike normal gameplay. There are two players named player1 and player2, and player1 always starts the game first. There is no need to place ships to board because the location of ships for both players is given as input. Every player has the same size board and ship number (total ship size are the same for all players). The size of the ship can be one unit. Players make their moves in their turns and select a location (x, y) to hit. Both players start to hit with given coordinates as input but continue hitting with BFS/DFS algorithm. Each cell has four neighbors in the coordinate system: top, left, bottom, and right cell. Thus, there are four directions to visit a neighbor (you can move only vertically or horizontally). Both for BFS and DFS, there are priorities for cells like that: Priority (top cell) > Priority (left cell) > Priority (bottom cell) > Priority (right cell). For example, it is given that the starting coordinate of player1 is (3, 3). When examining the neighbor cells for the second hit, you should first visit (2, 3) for BFS and DFS. Also, continue to select other cells according to priorities. The example game board is given below (see Figure 1). It is a size of 5x5 board. The coordinate system's starting point starts from the left-upper corner and increases to the right and down. As you can see in the figure, there is one ship on the board, and it is placed on coordinates (3, 1) – (3, 2) with the size of two units.

x/y	0	1	2	3	4
0					
1					
2					
3		x	x		
4					

Figure 1: Example board

The object of each player is to find all opponent's ships with the minimum number of hits. To do that, there are two strategies to traverse on coordinates system: BFS or DFS. So, the crucial implementation point is selecting the next coordinates according to BFS/DFS algorithms. So, the sequence of hit locations is generated by BFS or DFS except for the first location. After selecting the location to hit, the current player hits that location and checks whether it hits a ship or not, and then the next player continues to play with selecting location and hitting. It goes like that. Even a player hits a ship, the turn passes to the other player. So, each turn, there is one move. Each player continues hitting until all ships of one of the players are sunk, and at that point, the game finishes. A player that sunk all ships of its opponent before another player wins the game. You are expected to find which player wins the game with the given settings. Also, it is asked the number of visited nodes during the traverse, the number of nodes kept in the memory, and the running time. The game settings of a player are given as input. The format of the setting of one player is:

```
<Algorithm>
<xStart> <yStart>
<B> <N>
<x0> <y0> <x1> <y1>
<x2> <y2> <x3> <y3>
…
```

where, <Algorithm> is BFS or DFS, <xStart> is start position for hitting on opponent board on the x-axis, <yStart> is start position for hitting on opponent board on the y-axis, B is board size (Remember that board size is BxB and it must be a square and size of at least 2x2), N is the number of ships that player has (the minimum value of N is 1). After that, each N line represents the ships' coordinates in the pattern of starting x coordinates, starting y coordinates, finishing x coordinates, finishing y coordinates, respectively. For example, for figure 1, the input file for player1 is:

```
DFS
3 3
5 1
3 1 3 2
```

where, the algorithm is DFS, the start location on the opponent board is (3, 3), the game board size is 5x5. This player has one ship and its coordinates: (3, 1) – (3, 2).

Implementation (70 points)

1. Formalize this problem in a well-defined graph form and present your state and action representations in detail:
   a. Describe state and action representations in detail.
   b. You are not required to optimize the search procedure. Do not implement any forward checking mechanisms.
   c. Every visiting location on board should be represented by a node.

2. Run both BFS and DFS algorithms **(run program for game2 and game3)**, and print the following information like given outputs and analyze the results in terms of:
   a. the number of the visited nodes
   b. the number of nodes kept in the memory
   c. the running time

3. The algorithm (BFS or DFS) should be used for finding the hit sequence (visiting cells). The player board settings should be chosen by a command-line argument as given in the example.

4. You must use a graph representation for the model and solve it using DFS or BFS. Any other method will not be accepted. You can use graph representation for each player separately to find the hit sequence.

5. If your program crashes due to memory overload, you can use the **"third game settings (game4 and game5)"** for your report.

Example Scenarios

1. Game0 Run

```
>./project1 games/game0/player1.txt games/game0/player2.txt
The algorithm: Player1: DFS, Player2: DFS
The number of visited nodes: Player1: 2, Player2: 2
The number of nodes kept in the memory: Player1: 5, Player2: 5
The result: Player2 won!
The running time: 3ms
```

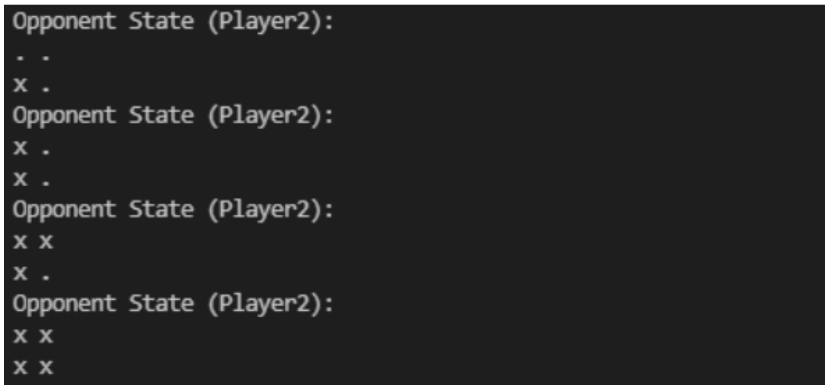

Figure 2: Hit sequence of player1

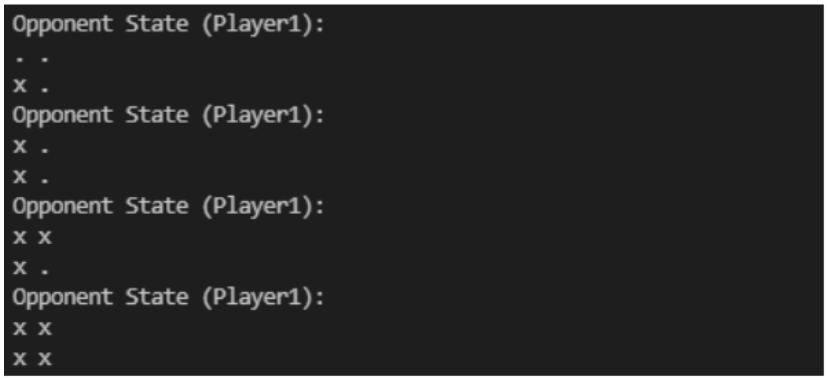

Figure 3: Hit sequence of player2

2. Game1 Settings

The input files:

player1.txt

```
DFS
3 3
7 2
2 0 2 4
5 0 6 0
```

player2.txt

```
DFS
3 3
7 2
2 0 2 4
6 0 6 1
```

The output:

```
>./project1 games/game1/player1.txt games/game1/player2.txt
The algorithm: Player1: DFS, Player2: DFS
The number of visited nodes: Player1: 31, Player2: 30
The number of nodes kept in the memory: Player1: 85, Player2: 85
The result: Player1 won!
The running time: 5ms
```

Note: The number of visited nodes and the number of nodes kept in the memory may differ for different implementations. Also, the running time may vary for other runs. However, the result of the game must be the same as given above.

Compiling and Running

Your program should compile and run using the following commands (if you have other commands for compilation, you should state them as a comment in your code). Compiling and running your codes on ITU servers is recommended via ssh.

```
g++ -std=c++11 -Wall -Werror project1.cpp -o project1

./project1  <player1-board> <player2-board>

./project1 games/game2/player1.txt games/game2/player2.txt
```

Report (30 points)

Please prepare a report and discuss the following items:

1. Present your problem formulation,

    a. Describe node and assignment representations in detail.
    b. Write your pseudo-code.
    c. Show the complexity of your algorithm on the pseudo-code.

2. Analyze and compare the algorithm results for "**game2 and game3**". They have the exact locations for ships but with different algorithms. Hence, you can compare BFS with DFS through game2's players vs. game3's players in terms of:

    a. the number of visited nodes
    b. the number of nodes kept in the memory
    c. the running time

3. Why should we maintain a list of discovered nodes? How does this affect the outcome of the algorithms?

4. How does increasing board size while keeping the same number of ships on the board affect the memory complexity of the algorithms?

Notes:

• Please submit your homework and its report only through Ninova. Late submissions will not be accepted.

• Please do not forget to write your name and student ID on the top of each document you upload. Also, **your code must have comments to explain your work. In case of insufficient comments, a point penalty (up to 10 points) will be applied.**

• You should write your code in C++ language and try to follow an object-oriented methodology with well-chosen variables, methods, and class names and comments where necessary. **You can use the C++ Standard Template Library (STL).** You can define multiple classes in a single cpp file or use multiple cpp files with header files.

• Your code should compile on Linux using g++. You can test your code through ITU's Linux Server (you can access it through SSH). Your code must compile without any errors.

• You may discuss the problem addressed in the homework at an abstract level with your classmates, but you should not share or copy code from your classmates or any web sources. You should submit your individual homework. In case of detection of an inappropriate exchange of information, disciplinary actions will be taken.

• If you have any questions, please contact T.A. Muhammed Raşit Erol via erolm15@itu.edu.tr.