**BLG336E**

**Homework-1 Report**

**24.03.2020**

Burak Bozdağ

150170110

**1 Overview**

In this homework I simulated a Pokémon battle where a Pikachu is fighting against a Blastoise.

**2 Graph Implementation**

I implemented the code which creates a graph according to the rules given in the Overview section. I created the graph according to the max-level value given in the graph. My code outputs the last layer’s node information. An example run is given below:

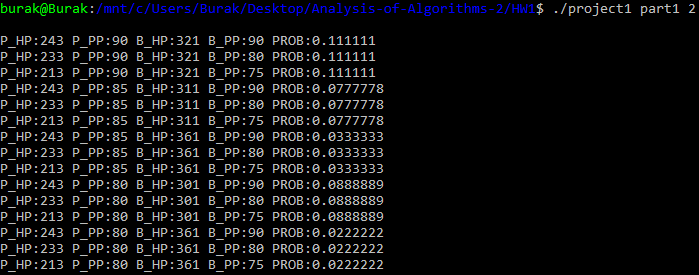


Figure 1: Graph Implementation

**3 BFS-DFS Implementation**

Using the graph generated by the functions in the previous part, I implemented BFS and DFS algorithms to traverse the graph. I ran both BFS and DFS algorithms and printed node count and running time.

An example run for BFS is given below:

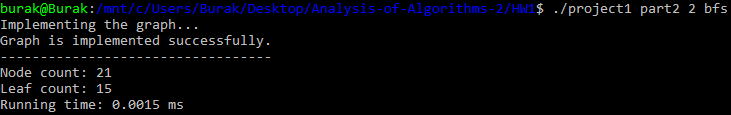


Figure 2: BFS Implementation

An example run for DFS is given below:

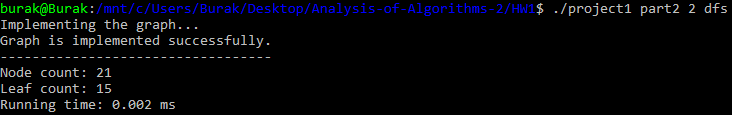


Figure 3: DFS Implementation

There is not much running time difference when maximum level is low. But when we are dealing with higher levels, these algorithms start to show some difference in terms of running time.

For example; when we run the program with max-level=12 parameter, the output of the program for both algorithms is as follows:

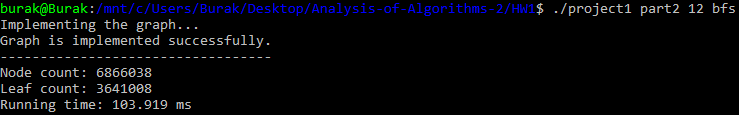


Figure 4: BFS, max-level=12

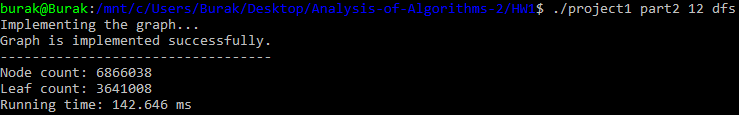


Figure 5: DFS, max-level=12

As seen above, even the time complexity of both algorithms is O(n), BFS is faster than DFS. Because in this scenario, the graph’s breadth (width) grows faster than graph’s depth (level). BFS algorithm allocates less memory when accessing to this kind of graphs, so that the latency of the program decreases significantly. In this example, BFS is nearly 27% faster than DFS.

**4 Probability of the Easiest Path**

For both Pikachu and Blastoise, I found out the probability of the easiest action sequence (containing minimum number of levels) to win the battle.

At first, I found a leaf node with the lowest level using BFS. Then, I used DFS to find the action sequence of this node to solve the problem. Example outputs for both Pikachu and Blastoise are given below:

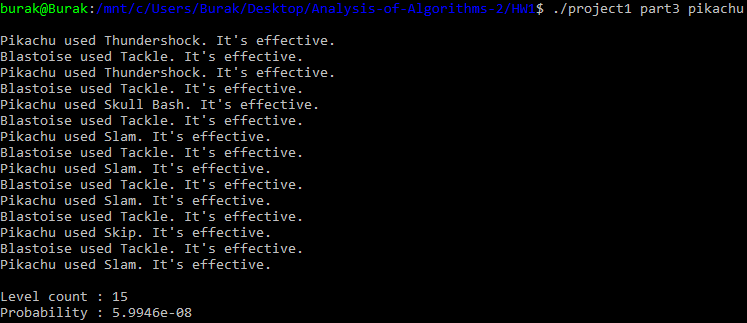


Figure 6: Easiest action sequence for Pikachu

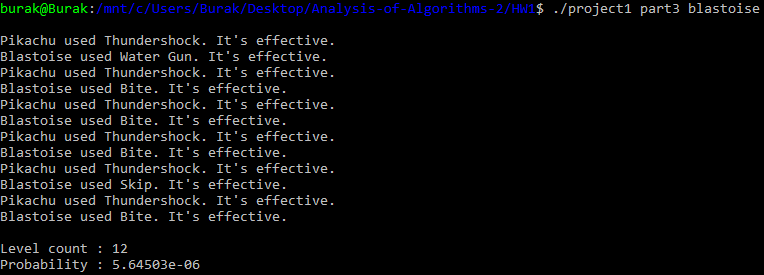


Figure 7: Easiest action sequence for Blastoise

***An important note****: When the program is finding the easiest action sequence for Pikachu, it allocates nearly 4GB of memory. If there is not enough memory for the program, you can get a* ***segmentation fault!*** *In my code at lines 9 and 10, you can easily define new HPs for both Pikachu and Blastoise in order to decrease memory allocation.*

**5 Compiling Instructions**

This C++ code uses C++11 standards. Please compile it using this command:

**g++ -std=c++11 main.cpp -o project1**

***The End of the Report***