BLG336E Project-1

Şahin Akkaya, 150170098 -Istanbul Technical University 24/03/2020

Part 1

Listing 1: Sample run of my program for part 1.

```
$ make
$ ./project1 part1 2
P_HP:170 P_PP:90 B_HP:160 B_PP:90 PROB:0.111111
P_HP:160 P_PP:90 B_HP:160 B_PP:80 PROB:0.111111
P_HP:140 P_PP:90 B_HP:160 B_PP:75 PROB:0.111111
P_HP:170 P_PP:85 B_HP:150 B_PP:90 PROB:0.077778
P_HP:160 P_PP:85 B_HP:150 B_PP:80 PROB:0.077778
P_HP:140 P_PP:85 B_HP:150 B_PP:75 PROB:0.077778
P_HP:170 P_PP:85 B_HP:200 B_PP:90 PROB:0.033333
P_HP:160 P_PP:85 B_HP:200 B_PP:80 PROB:0.033333
P_HP:140 P_PP:85 B_HP:200 B_PP:75 PROB:0.033333
P_HP:170 P_PP:80 B_HP:140 B_PP:90 PROB:0.088889
P_HP:160 P_PP:80 B_HP:140 B_PP:80 PROB:0.088889
P_HP:140 P_PP:80 B_HP:140 B_PP:75 PROB:0.088889
P_HP:170 P_PP:80 B_HP:200 B_PP:90 PROB:0.022222
P_HP:160 P_PP:80 B_HP:200 B_PP:80 PROB:0.022222
P_HP:140 P_PP:80 B_HP:200 B_PP:75 PROB:0.022222
```

Part 2

Listing 2: Sample runs for part 2.

```
$ ./project1 part2 7 bfs
node count is: 52596
It took 0.052432 seconds to execute
$ ./project1 part2 7 dfs
node count is: 52596
It took 0.057031 seconds to execute
$ ./project1 part2 9 bfs
node count is: 1118496
It took 1.11558 seconds to execute
$ ./project1 part2 9 dfs
node count is: 1118496
It took 1.1953 seconds to execute
```

As it can be seen from the results, running times for both algorithms are almost equal and they produce the same result.

Part 3

Listing 3: Solution for part 3.

```
$ ./project1 part3 pikachu
Pikachu used Thundershock. Effective (P:200, B:160)
Blastoise used Tackle. Effective (P:170, B:160)
Pikachu used Thundershock. Effective (P:170, B:120)
Blastoise used Tackle. Effective (P:140, B:120)
Pikachu used Slam. Effective (P:140, B:60)
Blastoise used Tackle. Effective (P:110, B:60)
Pikachu used Slam. Effective (P:110, B:0)
Level count: 7
Probability: 9.25926e-05

$ ./project1 part3 blastoise
Pikachu used Thundershock. Effective (P:200, B:160)
Blastoise used Tackle. Effective (P:170, B:160)
Pikachu used Thundershock. Effective (P:170, B:120)
Blastoise used Bite. Effective (P:110, B:120)
Pikachu used Thundershock. Effective (P:110, B:80)
Blastoise used Bite. Effective (P:50, B:80)
Pikachu used Thundershock. Effective (P:50, B:40)
Blastoise used Bite. Effective (P:0, B:40)
Level count: 8
Probability: 3.6169e-05
```

I have used breadth first search algorithm to find the shortest path. While creating the tree I was doing something similar to this:

Listing 4: Pseudo-code for creating the tree .

```
create_node(node, max_level){
    if(node.level == max_level)
        return
    node.children = find_children(node)
    for child in children
        create_node(child, max_level)
}
create_node(root, MAXLEVEL)
```

Following Listing 4... it is clear that this algorithm will create nodes in the order of we see them while doing depth first search. I tried to reuse this by setting MAXLEVEL to 999 and re-setting it back to the some lower value when I find a solution. As a matter of course, the first solution that I encounter with this method was way too late than it should. It was at 17^{th} level for

Pikachu. There were 2 choices: either I immediately return and perform a search on the tree and get a wrong result, 17 because tree is not complete, or I set MAXLEVEL to 17 and wait for the whole graph to complete which is horrible for both memory and time. So I rewrite my function to create tree level by level.