ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 223E
DATA STRUCTURES

HOMEWORK NO : 2

HOMEWORK DATE : 12.05.2024

GROUP MEMBERS:

150210100 : Selin Yılmaz

SPRING 2024

Contents

1 INTRODUCTION

In this homework, I am working on states of a Pokemon battle. These Pokemons have different *attacks*, and this different attacks have different properties (Table 1 and 2). Our aim is first, implementing a graph until a given level. Second, finding the easiest way to end this battle.

Pikachu Attacks				
name	power point	accuracy	damage	first usage
thundershock	-10	100	40	0
skullbash	-15	70	50	0
slam	-20	80	60	0
pskip	100	100	0	3

Table 1

Blastoise Attacks				
name	power point	accuracy	damage	first usage
tackle	-10	100	30	0
watergun	-20	100	40	0
bite	-25	100	60	0
bskip	100	100	0	3

Table 2

2 GRAPH IMPLEMENTATION

To implement graph structure, first, I added constructor and addChild functions, and a *parent* member to *node*. *addChild* is used for adding a new child node to child DoublyList of a node. In addChild, this⟶ node assigned as childNode's (parameter which is taken by addChild) parent.

Then, I added root, max_level, and num variables to graph. Graph constructor takes node* root, and int max_level as parameters.

Graph creation is done using the *buildGraph* function of the graph. It takes node* root as parameter. Inside of it, there is a *DoublyList node* q* which works similarly to queue structure. The root is pushed into it (addBack). Then, in a loop, I popped the first element (removeFront), checked if it is a leaf node. If it is a leaf node, it is added to *DoublyList node* leafNodes* in order to use later. If it is not a leaf node, addChild function of graph is called.

graph::addChild takes a parent node as parameter and creates nodes for all of its children and adds them to parent's child DoublyList (using *node::addChild*). It provides necessary initializations (e.g. hp, pp, the probability of pokemons) to *child node*s' constructors, sends these created nodes to node::addChild.

This is how I found probability:

$$probability = (parent \longrightarrow prob/performable\_attacks) * (current\_attack \longrightarrow get\_accuracy()/100)$$

After addChild finishes its work, graph::buildGraph continues. In a loop, all of the children of the node (same node, that was given to addChild as parameter) are pushed to q (using addBack). These steps are repeated until q becomes empty.

In this way, I implemented my graph using **breadth first logic** (BFS). For the output of this part, I implemented *graph::printLevel* function. It takes root and level as parameters. By recursion, it finds the proper level and and prints the necessary information.

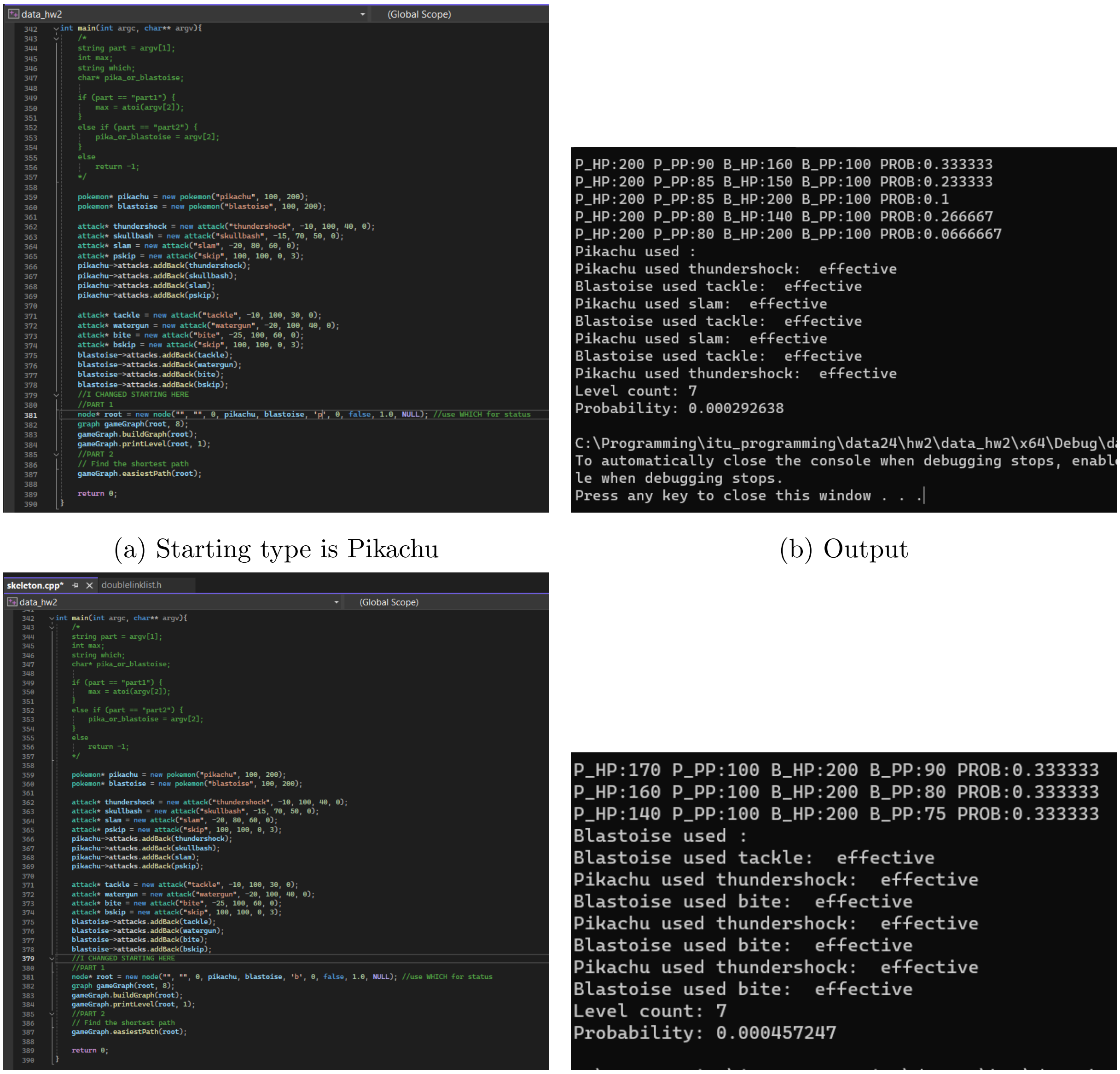
3 EASIEST PATH

In order to end the table in the most quickly with the highest probability possible, I implemented the *graph::easiestPath* function. It takes a starting node as a parameter. To determine which Pokemon starts first, I use the start node's status.

First, the function finds the leaf node (using *leafNodes* doublyList we created during graph building) with the lowest and the highest probability. Then, this leaf node and parents of it (current⟶parent is done in a loop) added to path (addFront). Finally, necessary outputs are printed iterating through the path.

4 OUTPUTS

I could not manage to build SSH environment on my computer. So I checked if my code is working properly and giving desired outputs on the program I use directly (Visual Studio). After I decided everything works properly, I tried if it is working with SSH on someone else's computer (this person is not taking this course). There were no problem with the first part. But, second part took 3 minutes (or more because we gave up after at some point) to compile. But, it was giving proper outputs when I built it on my computer. So here are the outputs and my main for the second part of assignment.



```cpp
int main(int argc, char** argv){
    /*
    string part = argv[1];
    int max;
    string which;
    char* pika_or_blastoise;

    if (part == "part1") {
        max = atoi(argv[2]);
    }
    else if (part == "part2") {
        pika_or_blastoise = argv[2];
    }
    else
        return -1;
    */

    pokemon* pikachu = new pokemon("pikachu", 100, 200);
    pokemon* blastoise = new pokemon("blastoise", 100, 200);

    attack* thundershock = new attack("thundershock", -10, 100, 40, 0);
    attack* skullbash = new attack("skullbash", -15, 70, 50, 0);
    attack* slam = new attack("slam", -20, 80, 60, 0);
    attack* pskip = new attack("skip", 100, 100, 0, 3);
    pikachu->attacks.addBack(thundershock);
    pikachu->attacks.addBack(skullbash);
    pikachu->attacks.addBack(slam);
    pikachu->attacks.addBack(pskip);

    attack* tackle = new attack("tackle", -10, 100, 30, 0);
    attack* watergun = new attack("watergun", -20, 100, 40, 0);
    attack* bite = new attack("bite", -25, 100, 60, 0);
    attack* bskip = new attack("skip", 100, 100, 0, 3);
    blastoise->attacks.addBack(tackle);
    blastoise->attacks.addBack(watergun);
    blastoise->attacks.addBack(bite);
    blastoise->attacks.addBack(bskip);
    //I CHANGED STARTING HERE
    //PART 1
    node* root = new node("", "", 0, pikachu, blastoise, 'p', 0, false, 1.0, NULL); //use WHICH for status
    graph gameGraph(root, 8);
    gameGraph.buildGraph(root);
    gameGraph.printLevel(root, 1);
    //PART 2
    // Find the shortest path
    gameGraph.easiestPath(root);

    return 0;
}
```

(a) Starting type is Pikachu

```
P_HP:200 P_PP:90 B_HP:160 B_PP:100 PROB:0.333333
P_HP:200 P_PP:85 B_HP:150 B_PP:100 PROB:0.233333
P_HP:200 P_PP:85 B_HP:200 B_PP:100 PROB:0.1
P_HP:200 P_PP:80 B_HP:140 B_PP:100 PROB:0.266667
P_HP:200 P_PP:80 B_HP:200 B_PP:100 PROB:0.0666667
Pikachu used :
Pikachu used thundershock:  effective
Blastoise used tackle:  effective
Pikachu used slam:  effective
Blastoise used tackle:  effective
Pikachu used slam:  effective
Blastoise used tackle:  effective
Pikachu used thundershock:  effective
Level count: 7
Probability: 0.000292638

C:\Programming\itu_programming\data24\hw2\data_hw2\x64\Debug\d
To automatically close the console when debugging stops, enabl
le when debugging stops.
Press any key to close this window . . .
```

(b) Output

```cpp
int main(int argc, char** argv){
    /*
    string part = argv[1];
    int max;
    string which;
    char* pika_or_blastoise;

    if (part == "part1") {
        max = atoi(argv[2]);
    }
    else if (part == "part2") {
        pika_or_blastoise = argv[2];
    }
    else
        return -1;
    */

    pokemon* pikachu = new pokemon("pikachu", 100, 200);
    pokemon* blastoise = new pokemon("blastoise", 100, 200);

    attack* thundershock = new attack("thundershock", -10, 100, 40, 0);
    attack* skullbash = new attack("skullbash", -15, 70, 50, 0);
    attack* slam = new attack("slam", -20, 80, 60, 0);
    attack* pskip = new attack("skip", 100, 100, 0, 3);
    pikachu->attacks.addBack(thundershock);
    pikachu->attacks.addBack(skullbash);
    pikachu->attacks.addBack(slam);
    pikachu->attacks.addBack(pskip);

    attack* tackle = new attack("tackle", -10, 100, 30, 0);
    attack* watergun = new attack("watergun", -20, 100, 40, 0);
    attack* bite = new attack("bite", -25, 100, 60, 0);
    attack* bskip = new attack("skip", 100, 100, 0, 3);
    blastoise->attacks.addBack(tackle);
    blastoise->attacks.addBack(watergun);
    blastoise->attacks.addBack(bite);
    blastoise->attacks.addBack(bskip);
    //I CHANGED STARTING HERE
    //PART 1
    node* root = new node("", "", 0, pikachu, blastoise, 'b', 0, false, 1.0, NULL); //use WHICH for status
    graph gameGraph(root, 8);
    gameGraph.buildGraph(root);
    gameGraph.printLevel(root, 1);
    //PART 2
    // Find the shortest path
    gameGraph.easiestPath(root);

    return 0;
}
```

(c) Starting type is Blastoise

```
P_HP:170 P_PP:100 B_HP:200 B_PP:90 PROB:0.333333
P_HP:160 P_PP:100 B_HP:200 B_PP:80 PROB:0.333333
P_HP:140 P_PP:100 B_HP:200 B_PP:75 PROB:0.333333
Blastoise used :
Blastoise used tackle:  effective
Pikachu used thundershock:  effective
Blastoise used bite:  effective
Pikachu used thundershock:  effective
Blastoise used bite:  effective
Pikachu used thundershock:  effective
Blastoise used bite:  effective
Level count: 7
Probability: 0.000457247
```

(d) Blastoise

Figure 1