Project 3 Graph modeling and graph algorithms
COT 4400, Spring 2019

Julie Reyes | u76631122

Introduction:

For this project, we were required to determine the amount of water that takes the most moves to make given a set of buckets of different and unique sizes. The amount of water the set of buckets can hold ranges from 0 to the sum of the sizes of all the buckets in the given set. Initially, the problem was approached conceptually to determine a method to solve the problem from which an algorithm could be developed.

Modeling the problem:

Prior to developing an algorithm to solve the problem presented in this project, the problem was approached for a simple example since inputs and outputs were provided. This was done in order to understand how the program should be processed in a conceptual manner. The first observation that was made while examining the simple example was that the data structure to model the graph would be an adjacency list. Next, the states were determined to be the amounts of water the buckets hold in an instance. Lastly, the transitions were determined to be the moves from one state to another by filling a bucket, emptying a bucket, or pouring water between buckets. Since the states represented the amounts of water each bucket in the set had, the graph was determined to be vertex-labeled. The edges of the graph did not require weights as moves from one instance of the state of the buckets was produced by only one of the three transition options at a time, therefore all the edges were of equal value. Lastly, since the transitions were determined as one of three options from the current state, the graph edges were directed from the current state to the next state that could be reached.

To compute the algorithm for this project, the first obstacle was to create all the possible vertices or states of buckets that could exist. For the algorithm to create all the states of the buckets, a function was developed that built the buckets state by the vertex number. That formula takes in a unique number to identify a vertex and a vector of the sizes of all buckets in the problem. Then a vector that contained the state of each bucket was returned. The Pseudocode for determining the state of the buckets corresponding to a particular vertex is as follows:

Input: vector of max size of buckets, vertex identifier as an integer

```
Output: vector that corresponds to the unique vertex id
        for i = 0 to number of buckets - 1
            build_state.push_back(vertex id % (bucket_size.at(i)+1));
            vertex id /= bucket_size.at(i)+1;
        return build;
```

Note the since all the possible vertices were created for the graph, each vertex could be uniquely identified by an integer value. Once the states of the buckets were associated to a vertex number by the function created above, the edges were created.

To create the edges for the graph, each vertex was considered independently. For each vertex the edges to other vertices were made by determining the amounts of water in the buckets after 3 operations, emptying, filling, and pouring. The following is the pseudocode for adding edges to the graph:

```
For all the vertices in the graph
  For i=1 to all the buckets in vertex
        if can fill buckets.at(i) {
            fill bucket
            make edge corresponding to the state represented by filling at bucket i
        }

        if bucket at i has any water {
`       empty the bucket
      Add the edge to the graph that corresponds to the state with the empty bucket
        }
        For j = 0 to amount of buckets
            if(i == j)  continue;
      if the bucket at j is not full and the bucket at i is not empty
      add the contents of bucket at i to bucket j
            make bucket at i empty
            if bucket j has more than it can hold
      add the excess back to bucket i
       add edge to graph that corresponds to the state from pouring one
```

bucket into another

}

Lastly, to determine the amount of water that takes the most moves to make given a set of buckets a breadth-first search was performed on the graph created. To determined whether a vertex had been visited an array of Boolean values was created, initialized as false, and named visited[]. The index of the array corresponds with a vertex in the graph. Also, an array, named sums[], of all the possible sums for the given set buckets was created and initialized to -1. Lastly, an array of the levels, aptly named bfs_level[], for each vertex was also initialized to -1 and tracked the level each vertex was found at. The following is the pseudocode for the breath first search procedure, where u is the start vertex that corresponds to all the buckets being empty:

```
int u = 0;
int max_moves = 0;
int max_vertex = 0;
bfs_level[u] = 0;
visited[u] = true;
//begin BFS at vertex u
queue = queue();
queue.push(u);
while (!queue.empty()) {
    u = queue.front();
    queue.pop();
    for i = 0 to all the neighbors of u -1 {
        int temp = graph[u].at(i);
        if (visited[temp] == false) {
            queue.push(temp);
            bfs_level[temp] = bfs_level[u] + 1;
            visited[temp] = true;
            if((bfs_level[temp] > max_moves) && sums[amount in vertex temp] == -1){
                max_moves = bfs_level[temp];
                max_vertex = temp;
                sums[amount in vertex temp] = max_moves;
```

```
                }
                else if(sums[amount in vertex temp] == -1)
                        sums[amount in vertex temp] = bfs_level[u] + 1;
        }
    }
```

Once breath first search was completed and the max moves of the corresponding vertex had been found, the sums array was checked in order to ensure that if there was a tie the smallest amount of water for a number of moves it took was chosen. To find this vertex a for loop was used and if an element at the array matched the max moves that were determined during the breath first search, it was updated to the smaller amount of water. Lastly to end the program all the values required were output to “output.txt” file.