Name: Thuan Tran

IMPORTANT: To compile my .cpp file in linux, type: g++ -std=c++11 -o Tran\_cannibals Tran\_cannibals.cpp

You do not need to type ./Tran\_cannibals.out to run

You only need: ./Tran\_cannibals

Assumption: The start will be on the left side of the river and the mission is to get to the right side

1. My approach for this problem is by using backtracking. Basically, I decided to try every possibility there are and then backtrack if I hit a dead end.

At every possibility, I marked that I have tried this possibility by saving the data into a set. The data here is the string representation of the number of cannibals and missionaries on both side of the river.

I also use a stack so that if I reach the base case, when the function return, I will push the string result into the stack so that I will pop the stack later and print out the result.

I defined 5 operations possible:

* Bring 2 missionaries
* Bring 2 cannibals
* Bring 1 cannibal
* Bring 1 missionary
* Bring 1 missionary and 1 cannibal

All I also defined a Boolean value that indicate which side of the river I am in: true for Left side and false for right side. A set data structures that will hold the unique string representation of each path and will return false if we already reach it. A stack that will be used to push in string that represent what path we chose.

At every point, I checked which side of the river I am currently at and use the 5 operations that I have (that involve minus and adding missionaries and cannibals on both side of the river).

My base case: If the number of missionaries and cannibals on the left = 3 and numbers on the right equal 0: return true

If the number of missionaries or cannibals on either side is negative: return false;

If the number of missionaries on either side of the river greater than 0 and less than the number of cannibals in that side of river: return false;

If we have visited this situation by checking the set, return false. Else add the string representation to the set

Recursive step: First, check which side of the river they are in:

If they are at the left side or right side, perform all 5 operations in if statement with variations. For the case in the left side, we want to take people from left to right, so I minus the number of people on the left and add to the right. For the case in the right side, we want to back people back, so I minus the number of people on the right and add to the left.

Pseudo code for entire function:

bool cross(int missLeft, int cannLeft, bool status, int missRight, int cannRight, set<string> & database, stack<string> &result )

{

Base case:

String represent= string form of all the first 5 parameters

Check if (represent in the set database)

Not in the set database => add it

Else => return false since have visited

Check if we reach the destination: return true if we did

Check if we are out of bound or missionaries got eaten: return false

Recursive case:

Check which side of river we in

Left side:

Call the recursive function for all 5 operations, first 2 parameters get decreased since we moving people to the right side, parameters for people on the right increased, negate the Boolean variable because we go to the other side

Else if we are in right side

The same for left side but first 2 parameter get increased while parameter for people on the right get decreased, negate the Boolean variable because we go to the other side

Return false at the end of the function

}

I have decided to use this approach for various reasons:

* This reminded me of backtracking that I used to did for the maze
* I find that the operations for this one is clear and concise

The complexity of my program:

Let’s represent the total possible state of the problem on either side of the river:

The possible state is: 4 \* 4 \*2. Number 4 come from how many states there are for both missionaries and cannibals. They can have 0,1,2,3 people. 2 means which side of the river the boat is in: Left or right.

We can represent the total state possible by this formula: 2\* (n+1)^2 (where n as this case is 3).

Let’s put 2 \* (n+1) ^2 = x to shorten the notation.

So, the total time: T(x) = 8 + T (x-1) (8 represent constant operations that we will always do at each recursion. Assuming that checking and adding to the set is constant and other base case operations is constant 1 as well).

So: Let T (1) = 1 (we reach the conclusion)

T(x) = 8 + T(x-1) ( x- 1 because at every state, we remember it for back tracking).

T(x) = 8 + 8 + T(x-2) = 8 + 8 + 8 + T(x-3) = 8\*k + T(x-k)

X – k =1 => k = x – 1

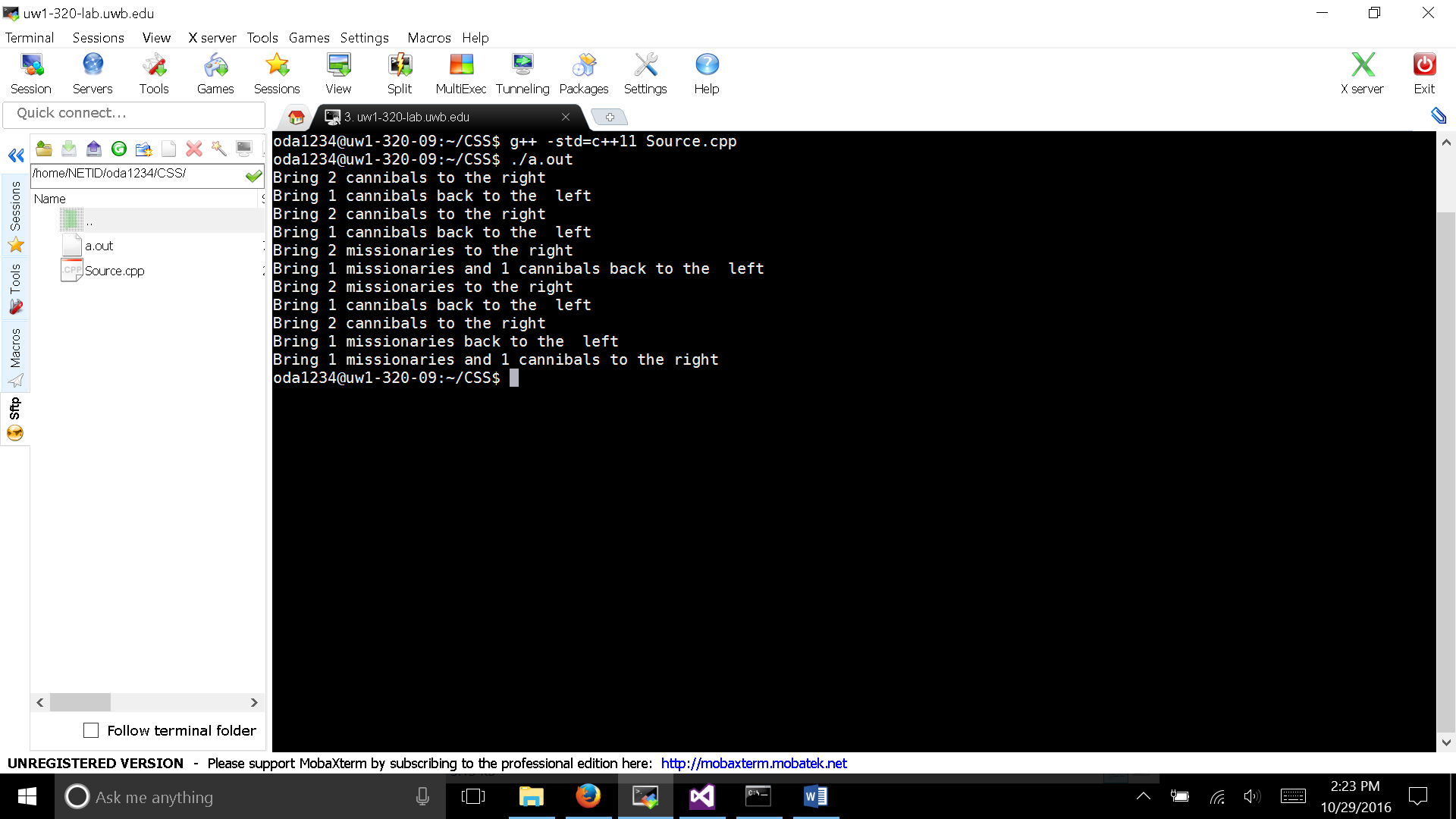
T(x) = 8 \* (x-1) + 1 = 8 \* ((2\* (n+1) ^2) – 1) + 1

T(x) = 16n^2 +32n + 25

T(n) = 16n^2 + 32n + 25

So the complexity of the program is O(n^2). It could not be log(n) because if we represent the operations as a graph, there are at some points that we will have connected nodes, traveling to where we used before.

**Output**



Explaination:

M will stand for missionary and C will be for Cannibals

Initial State: M M M C C C (Boat on the left)

After Step 1: M M M C C C (boat on right)

After step 2: M M M C C C (boat on left)

After step 3: M M M C C C (boat on right)

After step 4: M M MC C C (boat on left)

After step 5: M C M M C C (boat on right)

After step 6: M M C C M C (boat on left)

After step 7: C C M M M C (boat on right)

After step 8: C C C M M M (boat on left)

After step 9: C M M M C C boat on right)

After step 10: M C M M C C (boat on left)

After step 11: M M M C C C (boat on right)

External Reference: An explanation for Georgia Tech

https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwipy5P87vzPAhUKs1QKHdE4AtAQFggeMAA&url=http%3A%2F%2Fwww.cc.gatech.edu%2F~riedl%2Fclasses%2F2016%2Fcs3600%2Fexercises%2Fmissionaries-soln.pdf&usg=AFQjCNHmomWilQ05TSQmeTv4kXtGnQ-ALQ&sig2=QJc93WzOG70LjrqeJRayXg