David Olson

CS 312, Section 2

Project 3 Report

Included Code, Queue, PseudoCode and Complexity

Besides the form class, I've included three helper classes: BinaryHeap, HeapNode, and Node. BinaryHeap closely resembles the heap design in the exercises section of chapter 4 of our book. This design runs in O(logn) time for every action. Deleting from the front or inserting in the back call mirror sift down and bubble up functions. These travel through the array by doubling and halving the index respectively, hence O(logn) I've added just a few lines that deal with an array instantiated in the form class that keeps track of where every point is on the heap, so that the decrease key function is still in O(logn). I should also note that BinaryHeap has two constructors, one that puts the start node on the queue for the one path algorithm, and another that puts all the nodes on the queue for the all paths algorithm. The HeapNode class are the objects on the heap, simply a key and the point's index. The Node class is for objects in the array that keeps the pointers to the heap, and also keeps track of distance and edge pointers.

*|nodes|=|edges| according to the specs, so I will refer to just 'n' in this report

The form class contains two main functions:

OnePath:

```
Start a timer     O(1)
Initialize the pointer array     O(n)
Initialize the heap    O(1)
While the queue still has items and the stop node hasn't been reached:     O(worst case n)
    Delete the min from the queue     O(logn)
    If stop node:
        Halt
    Else:
        For each city in the node’s adjacency list:      O(1)
            If the node’s key + dist(node->city) < city’s dist:
                If the city is on the queue:
                    Decrease the city’s key    O(logn)
                Else:
                    Add city to queue with new key
            In both cases set the city’s previous pointer to the node     O(1)

If we never reached the stop index:
    It was unreachable    O(1)
Else:
    Draw the lines    O(1) usually
```

All Paths

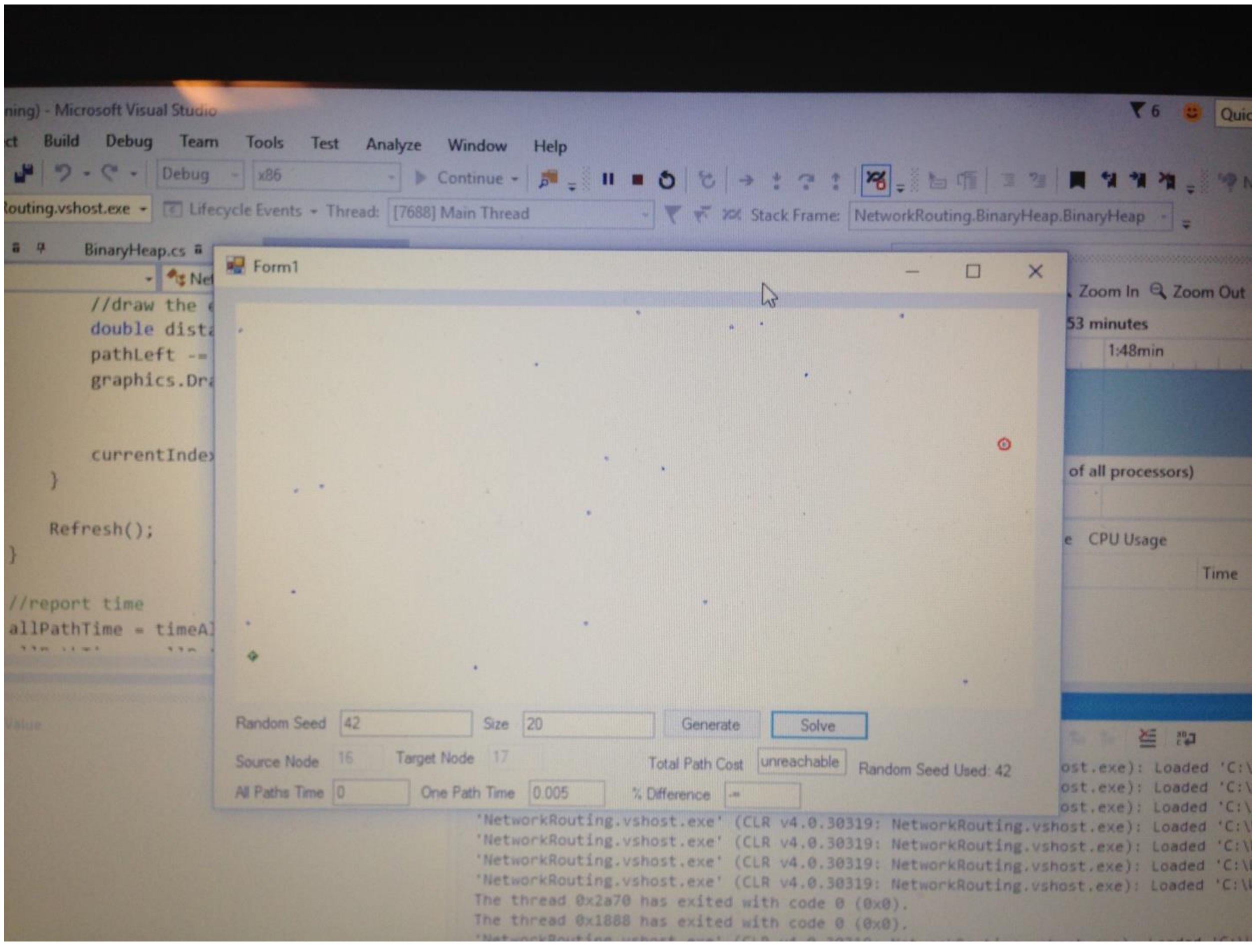
```
Start a timer     O(1)
Initialize the pointer array     O(n)
Initialize the heap    O(n) because the whole queue is initialized
While the queue still has items:     O(n)
    Delete the min from the queue     O(logn)
    For each city in the node’s adjacency list:      O(1)
        If the node’s key + dist(node->city) < city’s dist:
            Add city to queue with new key
            Set the city’s previous pointer to the node     O(1)

Draw the lines, watch out for an uncompleted path    O(1) usually
```

Both Functions:

The most dominant terms in both functions is the size n while loop. Inside this while loop the slowest operations that occur are O(logn). The theoretical final complexity of both onePath and allPaths is O(nlogn).

The space complexity of both algorithms is n. The points are stored in a size n array, there is a heap which uses a size n array, and a pointer heap that is also of size n.

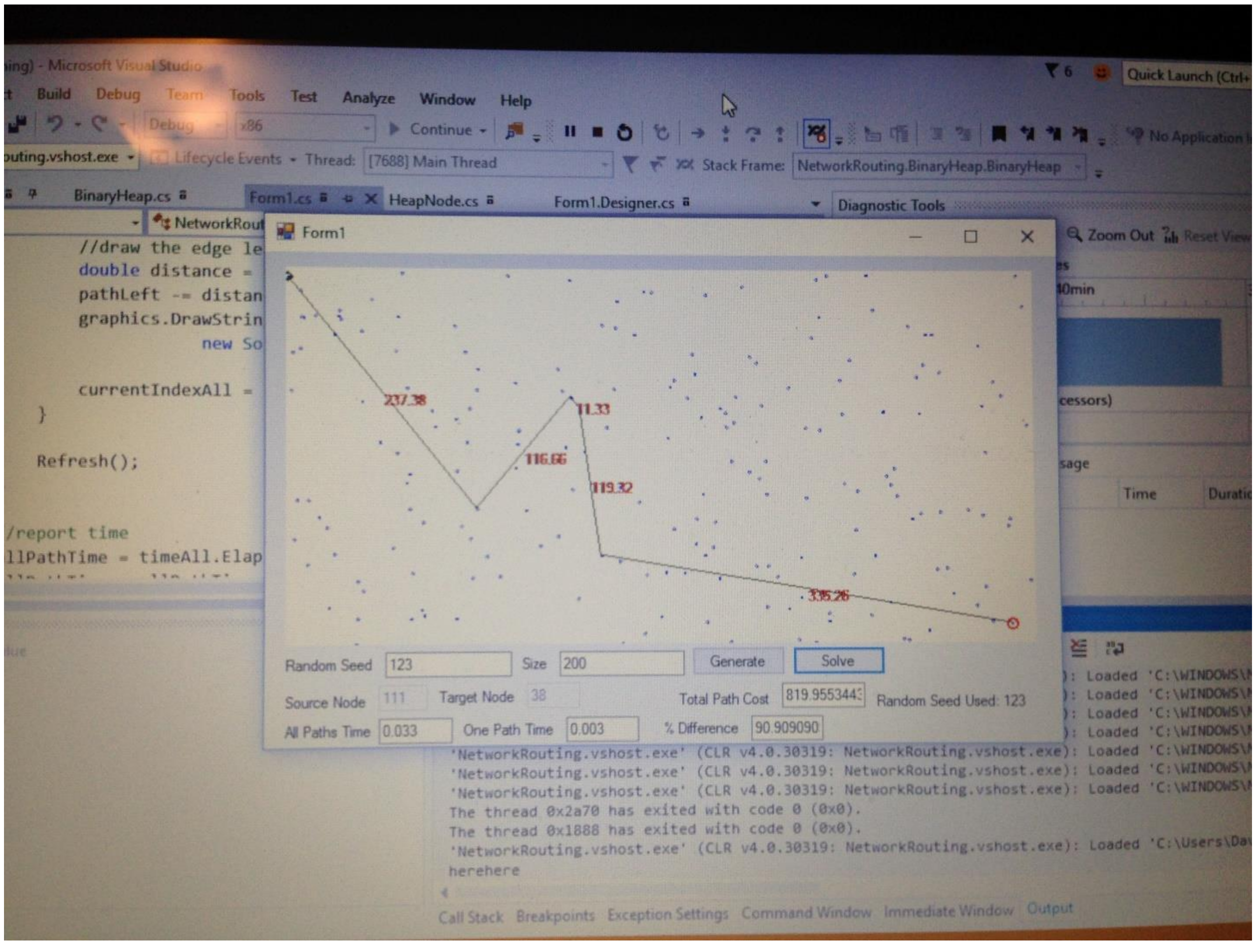
<u>ScreenShots</u>


ning) - Microsoft Visual Studio
Build Debug Team Tools Test Analyze Window Help
Debug x86 Continue
outing.vshost.exe Lifecycle Events Thread: [7688] Main Thread Stack Frame: NetworkRouting.BinaryHeap.BinaryHeap
BinaryHeap.cs
Form1
Zoom In Zoom Out
53 minutes
1:48min
of all processors)
CPU Usage
Time
Value

```
//draw the
double dist
pathLeft -=
graphics.Dr

currentInde
}

Refresh();
}

//report time
allPathTime = timeA
```

Random Seed: 42　Size: 20　Generate　Solve
Source Node: 16　Target Node: 17　Total Path Cost: unreachable　Random Seed Used: 42
All Paths Time: 0　One Path Time: 0.005　% Difference: -∞

```
ost.exe): Loaded 'C:\
ost.exe): Loaded 'C:\
ost.exe): Loaded 'C:\
'NetworkRouting.vshost.exe' (CLR v4.0.30319: NetworkRouting.vshost.exe): Loaded 'C:\
'NetworkRouting.vshost.exe' (CLR v4.0.30319: NetworkRouting.vshost.exe): Loaded 'C:\
'NetworkRouting.vshost.exe' (CLR v4.0.30319: NetworkRouting.vshost.exe): Loaded 'C:\
'NetworkRouting.vshost.exe' (CLR v4.0.30319: NetworkRouting.vshost.exe): Loaded 'C:\
The thread 0x2a70 has exited with code 0 (0x0).
The thread 0x1888 has exited with code 0 (0x0).
```


ing) - Microsoft Visual Studio
t Build Debug Team Tools Test Analyze Window Help
Debug x86 Continue
outing.vshost.exe Lifecycle Events Thread: [7688] Main Thread Stack Frame: NetworkRouting.BinaryHeap.BinaryHeap
BinaryHeap.cs Form1.cs HeapNode.cs Form1.Designer.cs Diagnostic Tools
NetworkRout
Form1
Zoom Out Reset View
237.38
11.33
116.66
119.32
335.26
Generate
Solve
Call Stack Breakpoints Exception Settings Command Window Immediate Window Output

```
        //draw the edge le
        double distance =
        pathLeft -= distan
        graphics.DrawStrin
                 new So

        currentIndexAll =
    }

    Refresh();


/report time
llPathTime = timeAll.Elap
```

Random Seed 123 Size 200

Source Node 111 Target Node 38 Total Path Cost 819.955344 Random Seed Used: 123

All Paths Time 0.033 One Path Time 0.003 % Difference 90.909090

```
): Loaded 'C:\WINDOWS\
): Loaded 'C:\WINDOWS\
): Loaded 'C:\WINDOWS\
): Loaded 'C:\WINDOWS\
'NetworkRouting.vshost.exe' (CLR v4.0.30319: NetworkRouting.vshost.exe): Loaded 'C:\WINDOWS\
'NetworkRouting.vshost.exe' (CLR v4.0.30319: NetworkRouting.vshost.exe): Loaded 'C:\WINDOWS\
'NetworkRouting.vshost.exe' (CLR v4.0.30319: NetworkRouting.vshost.exe): Loaded 'C:\WINDOWS\
The thread 0x2a70 has exited with code 0 (0x0).
The thread 0x1888 has exited with code 0 (0x0).
'NetworkRouting.vshost.exe' (CLR v4.0.30319: NetworkRouting.vshost.exe): Loaded 'C:\Users\Da
herehere
```

Empirical Data (two tables, but they represent the same tests)

One Path

	Test 1	Test 2	Test 3	Test 4	Test 5	Average
100	.001	.001	.001	.001	.001	.001
1000	.003	.003	.004	.007	.005	.0034
10000	.028	.034	.028	.032	.046	.0336
100000	.048	.189	.036	.342	.132	.1494
1000000	.038	.688	.857	.787	.291	.5322

All Paths

	Test 1	Test 2	Test 3	Test 4	Test 5	Average
100	.028	.027	.027	.03	.029	.0282
1000	.032	.03	.028	.028	.041	.0318
10000	.1	.097	.094	.079	.086	.0912
100000	.192	.281	.163	.186	.213	.207
1000000	.637	.497	.062	.093	.987	.4552

Constant Differences of All Path vs One Path

Size 100- 20x better; 1000-10x; 10000-3x; 100000-2x; 1000000-better than one path

Analysis

We know from our pseudocode analysis that the two algorithms are in the same complexity class, but will differ by a constant factor. It is hard to predict how much the times will differ, because the speed at which one path runs is highly affected by how quickly it encounters the stop node. My data showed that all paths was generally slower, but the constant by which it was better was anything but constant, and at the greatest size the two algorithms often switched superiority.

Other than the random factors of seeds, and start and stop node, the way I programmed the algorithm certainly had an effect on the constant time difference. When I first wrote the one path algorithm, I initialized the whole heap with dummy nodes, and all paths was performing better at size 100,000. I fixed that and retried the tests. My all paths algorithm is efficient because instead of literally inserting all n nodes (logn time) I hard code them into the array behind the heap, so O(n) time. So it's not too surprising to me, especially when corner nodes are chosen, that sometimes quickly setting up the all paths array and only doing deletions is faster than getting unlucky with one path and doing many insertions and deletions.