Assignment 3

Kyle Reagle and Yigit Gungor

November 26, 2017

(c) Optimizations:

- Changed the x and y coordinates in the Vertex class from int to short to lower the required memory of each vertex.
- Changed doubles to floats to halve the memory requirement of stored f values.
- f value added as a field to the Vertex class so that the fringe works as fast as intended. This significantly reduced run time.
- Changed closed list from ArrayList to Stack because pushing to a stack is constant time.

(d) Heuristics:

The best consistent heuristic we came up with was the euclidean distance from the current cell to the goal cell divided by 4 because this is the shortest possible distance, assuming that every cell is a highway cell. This is because h(n) at any cell n well never be greater than c(n,p) + h(p) where p is a successor of n. A diagonal distance heuristic divided by 4 should also be consistent, and it would be more appropriate for this grid world.

The other heuristics we tried were normal euclidean distance without the division, Manhattan distance, the example heuristic given in the PDF, diagonal distance (same distance as PDF example, different computation), and euclidean distance without the square root. None of these heuristics were consistent because they don't scale as well, but they required less computation time. These heuristics become more consistent with lower weight values in weighted A*.

(e) Regular A*:
Diagonal distance divided by 4 heuristic (consistent):

```
Average duration: 3723.44 milliseconds
Average path length: 183.00417
Average nodes expanded: 9984.06
Average memory usage: 20.29813632 megabytes
```

Normal diagonal distance heuristic:

```
Average duration: 123.98 milliseconds
Average path length: 147.58379
Average nodes expanded: 1519.58
Average memory usage: 18.54665552 megabytes
```

Euclidean distance heuristic:

```
Average duration: 232.04 milliseconds
Average path length: 154.28725
Average nodes expanded: 2131.04
Average memory usage: 19.95006704 megabytes
```

Manhattan distance heuristic:

```
Average duration: 37.36 milliseconds
Average path length: 149.55737
Average nodes expanded: 904.64
Average memory usage: 20.58225792 megabytes
```

Euclidean distance squared heuristic:

```
Average duration: 5.06 milliseconds
Average path length: 106.969086
Average nodes expanded: 120.96
Average memory usage: 17.7869728 megabytes
```

Weighted A* (w=1.5):

Diagonal distance divided by 4 heuristic (consistent):

```
Average duration: 2141.22 milliseconds
Average path length: 188.1233
Average nodes expanded: 8431.84
Average memory usage: 19.19475968 megabytes
```

Normal diagonal distance heuristic:

```
Average duration: 5.64 milliseconds
Average path length: 117.13381
Average nodes expanded: 157.52
Average memory usage: 17.48720464 megabytes
```

Euclidean distance heuristic:

```
Average duration: 5.82 milliseconds
Average path length: 120.37106
Average nodes expanded: 151.28
Average memory usage: 18.16818336 megabytes
```

Manhattan distance heuristic:

```
Average duration: 5.72 milliseconds
Average path length: 118.77044
Average nodes expanded: 166.04
Average memory usage: 18.33155264 megabytes
```

Euclidean distance squared heuristic:

```
Average duration: 4.7 milliseconds
Average path length: 111.8967
Average nodes expanded: 120.82
Average memory usage: 17.80062656 megabytes
```

Weighted A* (w=2):

Diagonal distance divided by 4 heuristic (consistent):

```
Average duration: 1755.14 milliseconds
Average path length: 179.31224
Average nodes expanded: 6882.54
Average memory usage: 20.34458368 megabytes
```

Normal diagonal distance heuristic:

```
Average duration: 4.82 milliseconds
Average path length: 115.04703
Average nodes expanded: 128.24
Average memory usage: 17.88627872 megabytes
```

Euclidean distance heuristic:

```
Average duration: 5.62 milliseconds
Average path length: 113.83597
Average nodes expanded: 126.6
Average memory usage: 17.81477408 megabytes
```

Manhattan distance heuristic:

```
Average duration: 4.92 milliseconds
Average path length: 116.99321
Average nodes expanded: 131.42
Average memory usage: 17.90921952 megabytes
```

Euclidean distance squared heuristic:

```
Average duration: 4.76 milliseconds
Average path length: 111.8967
Average nodes expanded: 120.82
Average memory usage: 17.80080256 megabytes
```

Uniform cost search:

```
Average duration: 5495.48 milliseconds
Average path length: 191.63365
Average nodes expanded: 12708.96
Average memory usage: 18.81405824 megabytes
```

(f) Right off the bat, it is apparent that weight values are disproportional to run time. Weighted A* runs much faster than A* on every heuristic when the weight value is greater than 1. This is also apparent with the first and second heuristics, which are both diagonal, but the second heuristic is 4 times the weight value of the first heuristic. Weight also seems to be proportional to path length, number of nodes expanded, and memory usage. This trend makes sense because the first heuristic assumes all cells are highways, and the rest of the heuristics assume that all cells are normal cells. In reality, even though most cells are normal, there is a good chance that path will pass through a hard to traverse region, and the cost of the path goes up. Therefore, higher weight values make the heuristic more proportional to the scale of g(n), which results in better path finding. This also explains why the euclidean squared heuristic did so well in regular A*, because it was more proportional to g(n).

Uniform cost search was much slower and more resourceful as intended because it is uninformed.

(g) Sequential Heuristic A*:

$w_1 = 1.25, w_2 = 2$

```
Average duration: 9.74 milliseconds
Average path length: 111.81703
Average nodes expanded: 120.82
Average memory usage: 23.33663136 megabytes
```

$w_1 = 1, w_2 = 1$

```
Average duration: 28.46 milliseconds
Average path length: 112.956505
Average nodes expanded: 425.36
Average memory usage: 21.0926704 megabytes
```

$w_1 = 2, w_2 = 2$

```
Average duration: 8.2 milliseconds
Average path length: 111.69824
Average nodes expanded: 120.52
Average memory usage: 22.16201776 megabytes
```

$w_1 = 1, w_2 = 1.5$

```
Average duration: 6.88 milliseconds
Average path length: 111.8967
Average nodes expanded: 120.96
Average memory usage: 22.64480832 megabytes
```

$w_1 = 2, w_2 = 1.25$

```
Average duration: 7.62 milliseconds
Average path length: 111.69824
Average nodes expanded: 120.52
Average memory usage: 22.14244624 megabytes
```

$w_1 = 2.5, w_2 = 1.6$

```
Average duration: 8.42 milliseconds
Average path length: 111.62102
Average nodes expanded: 120.42
Average memory usage: 22.2060832 megabytes
```

$w_1 = 3, w_2 = 2$

```
Average duration: 7.88 milliseconds
Average path length: 111.684044
Average nodes expanded: 120.34
Average memory usage: 22.2124696 megabytes
```

(h) For sequential heuristic search, it is once again apparent that higher weight values are preferable. When compared to A* and weighted A*, it is slightly slower and requires more memory, but seems to have shorter path lengths and less expanded nodes on average. Our implementation is efficient because it follows the pseudocode almost exactly, uses hashmaps for bp and g for linear time search, and uses a priority queue for the open list.

(i) Assuming $Key(s, 0) = w_1 * h_0(s) + g_0(s) > w_1 * g'(s_{goal})$, we pick a state s_i that is not expanded with the anchor search. We also know that state s_{goal} will never be expanded because the algorithm will terminate when a minkey for $OPEN_0$ exists.

When $i = 0$, which is the start state, $g(s_i) \leq w_1 * g(s_i)$. If $i \neq 0$, that means a previous state has already been expanded during the search. This we can calculate the following:

$$g(s_i) \leq g(s_{i-1}) + c(s_{i-1}, s_i)$$

$$g(s_i) \leq w_1 * (g(s_{i-1}) + c(s_{i-1}, s_i))$$

$$g(s_i) \leq w_1 * g'(s_i)$$

$$Key(s_i, 0) = w_1 * h_0(s_i) + g_0(s_i) \leq w_1 * g'(s_i) + w_1 * h_0(s_i) \leq w_1 * g'(s_i) + w_1 * c(s_i, s_{goal})$$

$$\text{Thus, } Key(s_i, 0) = w_1 * g'(s_{goal})$$

We have two contradicting statements. We have $key(s_i, 0) \leq w_1 * g'(s_{goal})$ and $w_1 * g'(s_{goal}) < key(s, 0)$ however h_0 should be consistent among all the calculations. So with contradiction we prove $key(s, 0) \leq w_1 * g'(s_{goal})$

We can assume three cases that the algorithm can terminate. If we find a path with the anchor search, since we know w's should be positive we can conclude the following:

$$g_0(s_{goal}) \leq w_1 * g'(s_{goal}) \leq w_1 * w_2 * g'(s_{goal})$$

If we find a path with another search, it will also result with a solution that is within the cost range of $w_1 * w_2$.

Lastly, if we cannot find a solution and the algorithm terminates, although $g(s_{goal}) \geq \infty$, the cost will still be bounded by $w_1 * w_2$.