Taylor Cowley

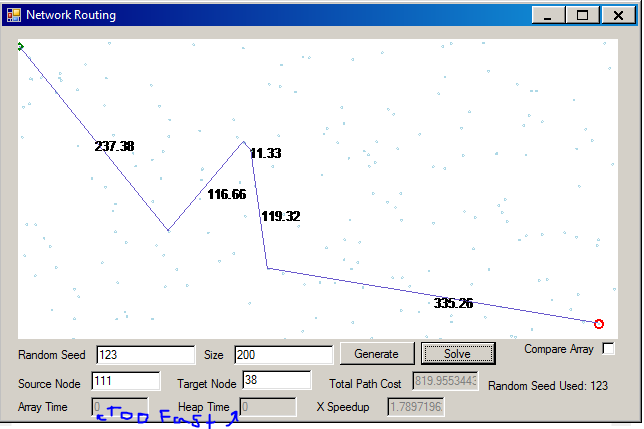
CS 312

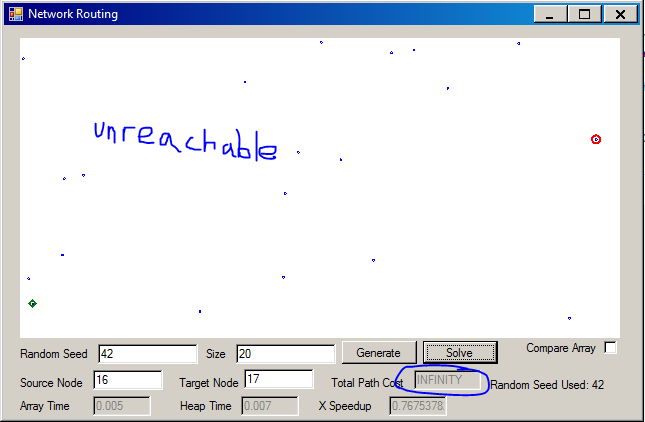
Project 3: Dijkstra’s Algorithm

1. See attached code.
2. Code complexity
   1. For the Array,
      1. insert is O(1) because we just put it at the end of the array
      2. delete-min is O(1) because we just throw away the first index of the array
      3. decrease-key is O(n) because we bubble-sort the single key to its proper location
   2. For the Heap,
      1. Insert is O(1) because when we insert, we just dump things at the end. (when you insert, the distance is infinity)
      2. Delete-min is O(logV) because we need to do the whole heap delete-min, where we take out the min, put the last element at the start, then bubble-down.
      3. Decrease-key is also O(logV) because after the key is decreased, it bubbles-up to its correct location
3. In each case, we might need to cycle through each vector. For each of these cycles, we
   1. Decrease-key the distances at 3 edges – O(1) for array and O(logV) for heap
   2. Delete-min to get the next node- O(V) for array and O(logV) for heap

Making the final complexity O(V^2) for the array and O(V(logV + logV)) ~ O(VlogV) for the heap

1. Screenshots! For 200 at seed 123, it calculated too quickly to really get times. For 20 at seed 42, it is not reachable, so the distance is INFINITY.





1. It looks like our complexity calculation is correct- the array time goes up squared relative to the number of nodes, and the heap time goes up by VlogV. It is too bad we ran out of memory; I was hoping for a nicer curve on the heap timing. There are some runs (such as seed 1476 with 100,000 points) that went a lot faster than the others. I expect this means there was an improbably short path between the nodes, and our algorithm found it really fast.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Data table** | | | | | | |
| **Seed** | **Size** | **Source** | **Target** | **Array Time** | **Heap Time** | **X Speedup** |
| 1456 | 100 | 39 | 83 | 0.004 | 0.084 | 0.04792 |
| 1457 | 100 | 59 | 14 | 0 | 0 | 1.06422 |
| 1458 | 100 | 95 | 71 | 0.005 | 0.008 | 0.634 |
| 1459 | 100 | 82 | 84 | 0 | 0 | 1.1571 |
| 1460 | 100 | 3 | 95 | 0 | 0 | 1.0483 |
| Average |  |  |  | 0 | 0 | 0.790308 |
| 1461 | 1000 | 224 | 259 | 0.005 | 0 | 7.08498 |
| 1462 | 1000 | 960 | 439 | 0.001 | 0 | 4.95111 |
| 1463 | 1000 | 721 | 972 | 0.005 | 0 | 6.20113 |
| 1464 | 1000 | 54 | 805 | 0.005 | 0.002 | 2.48097 |
| 1465 | 1000 | 905 | 499 | 0.006 | 0.001 | 5.53367 |
| Average |  |  |  | 0.0044 | 0.0006 | 5.250372 |
| 1466 | 10,000 | 9072 | 3662 | 0.39 | 0.011 | 34.5241 |
| 1477 | 10,000 | 7164 | 5481 | 0.398 | 0.009 | 40.9654 |
| 1468 | 10,000 | 1836 | 646 | 0.288 | 0.005 | 48.476641 |
| 1469 | 10,000 | 485 | 7146 | 0.365 | 0.008 | 43.4262 |
| 1470 | 10,000 | 4586 | 1417 | 0.456 | 0.013 | 33.3696 |
| Average |  |  |  | 0.3794 | 0.0092 | 40.152388 |
| 1472 | 100,000 | 12752 | 47962 | 32.058 | 0.11 | 289 |
| 1473 | 100,000 | 10236 | 84628 | 33.272 | 0.104 | 317 |
| 1474 | 100,000 | 62807 | 70160 | 35.691 | 0.177 | 201 |
| 1475 | 100,000 | 14029 | 13496 | 36.306 | 0.186 | 194 |
| 1476 | 100,000 | 9546 | 19592 | 7.614 | 0.034 | 220 |
| Average |  |  |  | 28.9882 | 0.1222 | 244.2 |
| 1477 | 1,000,000 | 504705 | 53686 | - | 2.571 | - |
| 1478 | 1,000,000 | 396618 | 860359 | - | 2.312 | - |
| 1479 | 1,000,000 | 428806 | 521130 | - | 2.16 | - |
| 1480 | 1,000,000 | 557796 | 587158 | - | 2.541 | - |
| 1481 | 1,000,000 | 887379 | 192927 | - | 1.697 | - |
| 1482 | 1,000,000 | 23816 | 668763 | - | 2.424 | - |
| Average |  |  |  | ?? 300 | 2.284166 | - |
| 1483 | 10,000,000 | “System.OutOfMemoryException” | | | | |