Dijkstra Priority Queue Report

Emily Elzinga

September 2022

1 Checklist

20 Correctly implement Dijkstra's algorithm and the functionality discussed above. Include a copy of your (well-documented) code in your submission to the TA.

20 Correctly implement both versions of a priority queue, one using an array with worst case O(1), O(1) and O(|V|) operations and one using a heap with worst case O(log|V|) operations. For each operation (insert, delete-min, and decrease-key) convince us (refer to your included code) that the complexity is what is required here.
Check, mostly. There are some kinks to work out for larger graphs.

10 Explain the time and space complexity of both implementations of the algorithm by showing and summing up the complexity of each subsection of your code.
Check. The overall complexity for the binary heap implementation is $O((|V|+3)log_n)$, and the overall complexity for the array implementation is $O(V^2)$ because the $deleteMin$ method gets called |V| times.

20 For Random seed 42 - Size 20, Random Seed 123 - Size 200 and Random Seed 312 - Size 500, submit a screenshot showing the shortest path (if one exists) for each of the three source-destination pairs, as shown in the images below.
For Random seed 42 - Size 20, use node 7 (the left-most node) as the source and node 1 (on the bottom toward the right) as the destination. Below in Figure 1 are my results.

Figure 1 – Dijkstra with the given parameters

For Random seed 123 - Size 200, use node 94 (near the upper left) as the source and node 3 (near the lower right) as the destination. This didn't go as well as the last one, as can be seen from Figure 2.

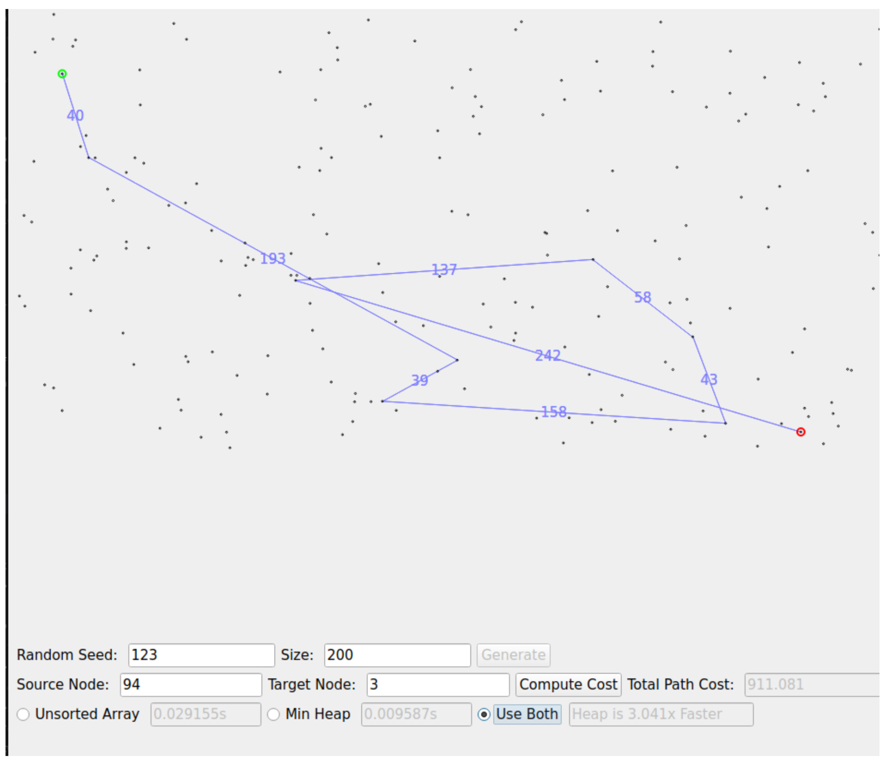

Figure 2 – Dijkstra with the given parameters

For Random seed 312 - Size 500, use node 2 (near the lower left) as the source and node 8 (near the upper right) as the destination, as in the third image below. This went about as well as the last one.

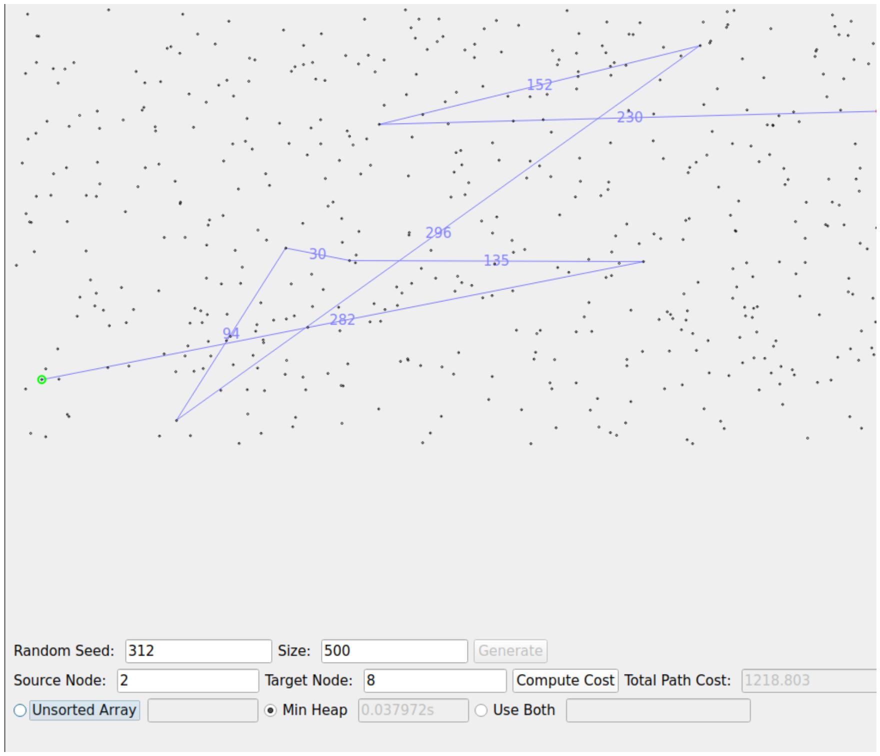

Figure 3 – Dijkstra with the given parameters

20 For different numbers of nodes (100, 1000, 10000, 100000, 1000000), compare the empirical time complexity for Array vs. Heap, and give your best estimate of the difference (for 1000000 nodes, run only the heap version and then estimate how long you might expect your array version to run based on your other results). For each number of nodes do at least 5 tests with different random seeds, and average the results. Redo any case where the destination is unreachable. Each time, start with nodes approximately in opposite corners of the network. Graph your results and also give a table of your raw data (data for each of the runs); in both graph and table, include your one estimated runtime (array implementation for 1000000 points). Discuss the results and give your best explanations of why they turned out as they did.

The array implementation looks exponential compared to the binary heap implementation, especially for large graphs.

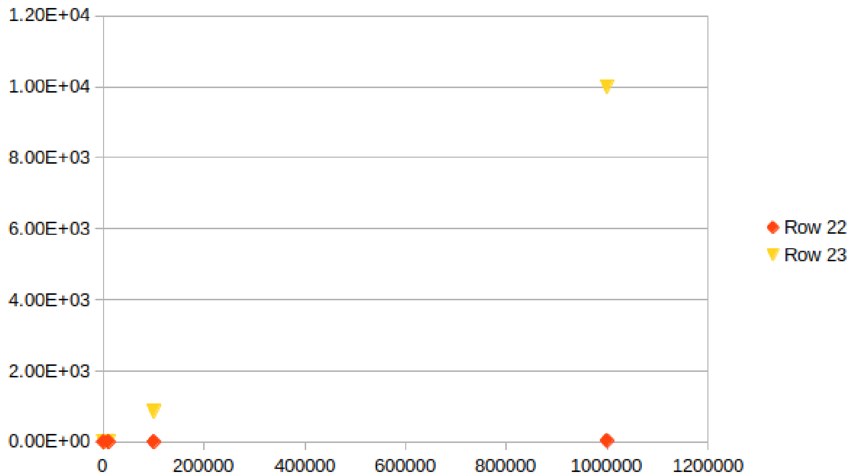

Figure 4 – Runtime comparison for the two algorithms

	A	B	C	D	E	F
1	Number Nodes	100	1000	10000	100000	1000000
2	Seed	47	25	89	103	169
3	Start Node	92	731	7439	11923	469637
4	End Node	38	241	6995	75489	379145
5	Binary Heap Ti	6.91E-04	1.49E-02	0.2079	2.411	36.43
6	Array Time	1.06E-03	0.8525	8.611	878	10000
7						
8	Number Nodes	100	1000	10000	100000	1000000
9	Seed	103	169	47	47	47
10	Start Node	52	362	8379	79750	446042
11	End Node	8	198	9529	37574	519442
12	Binary Heap Ti	5.88E-04	1.04E-02	0.1505	2.617	34.9
13	Array Time	1.22E-03	0.0878	7.034	878	10000
14						
15	Number Nodes	100	1000	10000	100000	1000000
16	Seed	25	89	169	169	169
17	Start Node	76	171	6320	58310	469637
18	End Node	85	380	822	8188	626181
19	Binary Heap Ti	6.56E-04	1.04E-01	0.2154	3.05	34.61
20	Array Time	1.14E-03	0.0751	7.228	800	10000
21						
22	Avg BH time	6.45E-04	4.31E-02	1.91E-01	2.69E+00	3.53E+01
23	Avg array time	1.14E-03	3.38E-01	7.62E+00	8.52E+02	1.00E+04
24						

Figure 5 – Runtime comparison for the two algorithms, spreadsheet form

2 Pseudocode

Algorithm 1: Builds a heap implementation of a priority queue for the given vertices and distances

```
function makeQueue (vertices, priorities);
Input  : Vertices V with dist array as Priorities
Output: A Priority Queue of vertices in heap format
bHeap = [inf]*(vertices size);
pointers = [-1]*(vertices size);
priorities = priorities;
for each index b of bHeap do
    pointer[bHeap[b]] = b
end
```

```
function deleteMin (H);
Input  : The Priority Queue H
Output: The vertex with the lowest priority
min = bHeap[0];
bHeap[0] = bheap[size - 1];
pointer[0] = (pointer(bHeap[size-1]));
bHeap = delete(bHeap[size-1]);
pointer = delete(pointers[bHeap[size-1]];
```

```
function decreaseKey (node, newPriority);
Input  : A node v in the graph and its new priority
Output: the PQ with the given node distance decreased
j = pointers[node];
priorities[node-1] = newPriority;
parentj = floor((j − 1)/2);
while newPriority < priorities[parentj] do
    temp = bHeap[j];
    bHeap[j] = bHeap[parentj];
    bHeap[parentj] = temp;
    temp = pointers[bHeap[j]];
    pointers[bHeap[j]] = pointers[bHeap[j];
    pointers[bHeap[parentj]] = temp;
    j = parentj;
    parentj = floor((j)/2);
end
```