

## A history and analysis of the “Dijkstra’s Shortest Path” algorithm

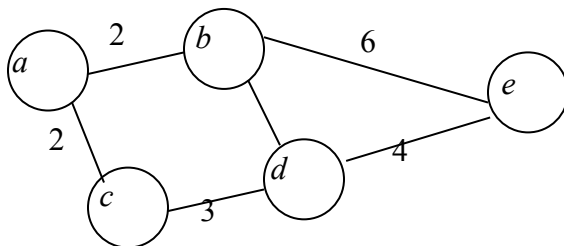
The mathematical field of graph theory was discovered by Leonard Euler on a windy Sunday afternoon in 1736 in Königsburg, Germany. While strolling the streets and smoking a pipe, Euler wondered if it was possible to traverse all seven bridges connecting three distinct land masses in Königsburg without crossing any bridge more than once. Euler would simplify the problem to a network of points, representing landmasses, and edges, representing bridges, and finally arrive at the principles of map traversal that are relevant to any system of interconnected nodes. Graph theory was born.

Two hundred twenty years later, a young, Dutch computer scientist named Edsger Dijkstra would come up with another revolutionary algorithm in the field that Euler created. While working at the Mathematical Centre in Amsterdam, Dijkstra was given the duty of showcasing the powers of the ARMAC computer, the Centre’s most powerful computer at the time. But with operating costs high, and copper wire unreliable and expensive, Dijkstra first had to search for a way to “convey electricity to all essential circuits, while using as little expensive copper as possible.”<sup>1</sup> In the end, Dijkstra’s solution to both problems was just one algorithm titled “the shortest subspanning tree algorithm.”<sup>2</sup> Although heralded by the mathematics community for its ramifications in graph theory, the algorithm was also utilized in electrical engineering (as seen with the ARMAC’s circuits), physics, and the field that would later become known as “computer science.”

This paper will examine how Dijkstra’s shortest path algorithm works, express it in pseudocode, and evaluate its complexity. Also, an in-depth look at the real world uses of the algorithm will showcase the breadth of problems benefiting from Dijkstra’s genius.

### The idea behind the algorithm

Suppose we are given a graph,  $G$ , that contains a set of vertices,  $\{a, b, c, d, e\}$ . Some of these vertices may be connected to one another by edges. And along these edges, there are numbers to indicate the distance between the vertices that the edge connects. This is called the weight.



<sup>1</sup> “Edsger Wybe Dijkstra.” 23 November 2007. “The History of Computing Project.”

<sup>2</sup> Ibid.

In this graph, there are various ways to get from  $a$  to  $e$ , such as  $a \rightarrow b \rightarrow e$ ,  $a \rightarrow c \rightarrow d \rightarrow e$ , and  $a \rightarrow c \rightarrow d \rightarrow b \rightarrow e$ . But clearly, it would make no sense to travel from  $a \rightarrow c \rightarrow d \rightarrow b$  if we can travel from  $a \rightarrow b$  directly. Another way of expressing this idea is that the shortest path from the source node to the destination node has to come from one of the shortest subpaths already discovered. Dijkstra's shortest path algorithm exploits this simple idea in what is referred to as the "explorer" model:

"A way to think about this "explorer" model is that starting from the source, we can send out explorers each traveling at a constant speed and crossing each edge in time proportional to the weight of each edge being traversed. Whenever an explorer reaches a vertex, it checks to see if it was the first visitor to that vertex: if so, it marks down the path it took to get to that vertex. [Because this explorer is the first to arrive at this vertex], the explorer must have taken the shortest path possible to reach the vertex. Then, this explorer deploys more explorers along each edge connecting the current vertex to its neighbors."<sup>3</sup>

Note that for an explorer to be able to mark down the path it took to arrive at the vertex it is currently standing on, the explorer's previous position must be saved in memory. A pointer called "previous" does just that: keep track of what vertex the explorer was at before the vertex it currently stands on. Without this pointer, it would be impossible to retrace the steps the explorer followed to arrive at its current vertex.

### From "explorer" to pseudocode

Before writing any code, we will define our input – namely, a graph,  $G$ , with edges  $E$  of the form  $\{v1, v2\}$  and vertices  $V$ , and a source vertex,  $s$ . This idea of explorers traversing the vertices can be broken down into two lists:  $U$ , which is a list of unexplored or unfinished vertices, and  $F$ , a list of explored or finished vertices.

Each vertex will have two attributes. The first is the distance to the source vertex. For the source vertex, this is defined to be zero, since the source vertex's distance to itself is zero. For all other vertices, this distance will initialize to infinity, and will change once a shorter path is found in the main body of the algorithm.

The second attribute is the previous vertex. Think of this as the vertex that the explorer was on before arriving at the current vertex. Since no vertex has been discovered when the algorithm starts, "previous" initializes to NULL for all vertices. Then, once the vertex is discovered, NULL will be overwritten. The code follows:

dist : array of distances from the source to each vertex  
prev : array of pointers to preceding vertices  
i : loop index  
F : list of finished vertices  
U : list of unfinished vertices

---

<sup>3</sup> "Dijkstra's Algorithm For Shortest Paths." 2005. CProgramming.com.

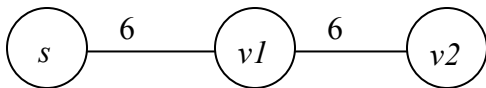
//The initialization of setting dist to infinity for all vertices except the source

```
for i = 0 to |V| - 1
    dist[i] = infinity
    prev[i] = null
dist[s]=0
```

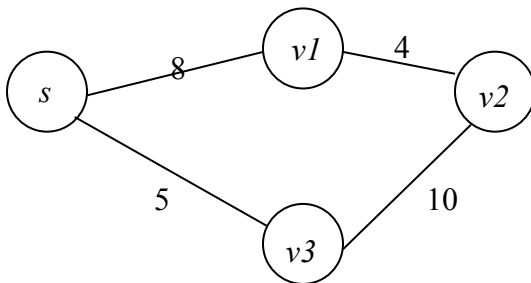
From here, explorers need to be sent out starting from the start node. So, fill  $U$  with all vertices except  $s$ . Then, pick the vertex,  $v$ , in  $U$  with the shortest distance to  $s$ , remove it from  $U$  and add it to  $F$ . Note that appending  $v$  to  $F$  indicates that there is no possible shorter path from  $s$  to  $v$ . This must be true because if a different shorter path existed, it would have to travel from  $s$  to a vertex other than  $v$ . We know that all vertices sharing edges with  $s$  have a length of at least that of the edge connecting  $s$  to  $v$ . So, such a path will never be shorter than that of  $s \rightarrow v$ .

The next step is where most of the work in the algorithm happens, and, not surprisingly, the most confusing. Let's approach it step by step.

Once we find out which vertex,  $v$ , has the shortest distance to  $s$ , we want to calculate the distance from all the other vertices that  $v$  connects with back to  $s$ .



In this example,  $v2$ 's distance to  $s$  is initialized to infinity, but now note  $v2$  is only 12 units away from  $s$  by the path  $s \rightarrow v1 \rightarrow v2$ . So, since this new distance to  $s$  (or just "distance") is less than  $v2$ 's current distance,  $v2$ 's distance must be changed to reflect this new value.



In the graph above however, suppose the distances of  $v1$ 's neighbors have already been updated. So,  $\text{distance}[v2]=12$ . Since the shortest path to  $v3$  from  $s$  is directly to  $v3$ , the algorithm will try to update the distances of all of  $v3$ 's neighbors. Here, however, we see that  $5+10$  is not less than 12, which is  $v2$ 's current distance. So,  $\text{distance}[v2]$  will not have to be updated because a shorter path already exists to arrive at  $v3$ .

This technique of conditionally tightening the upper bound on a vertex's distance back to the source vertex is referred to as "relaxation," although this is a bit of a misnomer since the value of the vertex's distance to  $s$  is shrinking, rather than growing or relaxing.

Relaxing proves to be a very powerful technique because it keeps track of the distances for all the vertices still in  $U$ . Choosing the vertex,  $v_u$ , in  $U$  with the smallest distance must necessarily be the shortest path from  $s$  to  $v_u$  because this path is comprised only of a shortest path from  $s$  to  $v_x$ , an intermediate vertex, and then an edge from  $v_x$  to  $v_u$ . So, for each iteration of the algorithm, the shortest path to a new vertex is found, the size of  $U$  decreases by one, and eventually  $U$  will be empty. That means that the algorithm will have found the shortest path from  $s$  to every vertex in  $G$ .

As stated above, the algorithm will run until all vertices have been processed, which is the same as saying until  $U$  is not empty. The code follows:

```
while (U is not empty)
    pick the vertex,  $v_1$ , in  $U$  with the shortest path to  $s$ 
    add  $v_1$  to  $F$ 
    for each edge of  $v_1$ ,  $(v_1, v_2)$  //the notation for an edge
        //Do the "relaxation" move
        if ( $\text{dist}[v_1] + \text{length}(v_1, v_2) < \text{dist}[v_2]$ )
             $\text{dist}[v_2] = \text{dist}[v_1] + \text{length}(v_1, v_2)$ 
             $\text{prev}[v_2] = v_1$ 
        remove  $v_1$  from  $U$ 
        end if
    end for
end while
```

Combing the above code with our initialization steps from earlier yields the final code.<sup>4</sup>

## Complexity

There are two factors that affect the efficiency of Dijkstra's Algorithm: the number of vertices,  $|V|$ , and the number of edges,  $|E|$ . If  $U$  is a linear array, then extracting the vertex from  $U$  with the minimum distance takes time  $O(V)$ , and since this operation is performed  $|V|$  times, the total time for extracting vertices from  $U$  is  $O(V^2)$ . Then, once inside the **for** loop, each edge of the vertex is examined exactly once. This gives a total of  $|E|$  iterations of this **for** loop, with each iteration taking time  $O(1)$ . So, the final runtime of the algorithm is  $O(V^2 + E) = O(V^2)$ .

## Real World Uses

The real-world implementation of Dijkstra's Algorithm has exploded in the last decade due mostly to the rise of geographic information systems (GIS). As F. Benjamin Zhan notes in his analysis of shortest path algorithms on real road networks, "A key problem in network and transportation analyses is the computation of shortest paths

---

<sup>4</sup> "Dijkstra's Algorithm For Shortest Paths." 2005. CProgramming.com.

between different locations on a network...[all computed] in real time.”<sup>5</sup> Zhan’s essay provides the example of a 911 call requesting that a patient be rushed to a hospital. In a city or any moderately congested area, it is impossible to know in advance current traffic situations. So, an ambulance’s route to the patient must be computed in real time using real traffic data. A delay of seconds could literally mean the difference between life and death. But the number of variables needed to determine the time required to travel between two points in a city can be high, and very computationally intensive (weather, other accidents, construction, train crossings, etc.). For this reason, computer scientists have focused hard on tweaking Dijkstra’s Algorithm by experimenting with more efficient data structures, like heaps and buckets, rather than lists, to keep track of which nodes should be pulled next from the list of unfinished vertices.

Parcel carriers like UPS also rely on GIS and shortest-path algorithms to keep their businesses profitable and environmentally-friendly. In early 2007, UPS realized that eliminating left-hand turns from its drivers’ routes reduces idling, which in turn lowers fuel consumption: "It seems small, but when you multiply it across 88,000 vehicles making nearly 15 million deliveries every day during the course of a year, it adds up," said Steve Holmes, a spokesmen for the company.<sup>6</sup> And because 98% of all UPS packages are processed electronically, the routes the drivers take are predetermined by a computer. So, before these left-hand-only routes were rolled out, the graphs (and specifically the vertices) that represent the drivers’ delivery areas had to be updated to include edges that connect vertices only via left-hand turns; edges representing vertices connected by right-turns were eliminated. So long as the vertices were stored in a dictionary, this would have been an easy modification since dictionary attributes can be changed while preserving the integrity of the rest of the entries.

As consumers have more access to real-time information on GIS networks, I can only see the usage and importance of Dijkstra’s Algorithm continuing to grow. GPS systems are already in cars, but are also finding their way into cell phones and even your Domino’s pizza.<sup>7</sup> As the computational complexity of these systems increases (think “if I get extra cheese will my delivery guy hit traffic?”), the need for faster hardware will become apparent. Also, the question as to whether a computer can run two (or more) instances of Dijkstra’s Algorithm on a single graph concurrently will be investigated. Theoretically, concurrent processing on such a problem is possible so long as the data structures remain easily maneuverable in memory so they can be rejoined later, but such a concurrent algorithm will surely be complex, and bear little resemblance to powerful, yet concise, tree spanning algorithm that came to Dijkstra in 1956.

---

<sup>5</sup> F. Benjamin Zhan. “Three Fastest Shortest Path Algorithms on Real Road Networks: Data Structures and Procedures.” *Journal of Geographic Information and Decision Analysis*. Vol. 1. No. 1. pp. 69-82. 1997.

<sup>6</sup> Lovell, Joe. “Left-Hand-Turn Elimination.” 9 December 2007. *The New York Times*.

<sup>7</sup> Horovitz, Bruce. “Where’s your Domino’s pizza? Track in online.” 28 January 2008. *USA Today*

## Works Cited

- Bradley, Robert E. and Charles Edward Sandifer. Leonhard Euler: Life, Work and Legacy. 30 January 2007. Elsevier Science. Pp. 84-86.
- Cormen, Thomas H, et al. Introduction to Algorithms. 1990. The Massachusetts Institute of Technology. New York.
- “Dijkstra’s Algorithm For Shortest Paths.” 2005. CProgramming.com.  
< <http://cprogramming.com/tutorial/computersciencetheory/dijkstra.html>>
- “Edsger Wybe Dijkstra.” 23 November 2007. 2008. “The History of Computing Project.” < [http://www.thocp.net/biographies/dijkstra\\_edsger.htm](http://www.thocp.net/biographies/dijkstra_edsger.htm)>
- Eppstein, David. “Dijkstra’s algorithm for shortest paths.” 4 April 2004. Active State Programming Network. 2006.  
<<http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/119466>>
- F. Benjamin Zhan. “Three Fastest Shortest Path Algorithms on Real Road Networks: Data Structures and Procedures.” *Journal of Geographic Information and Decision Analysis*. Vol. 1. No. 1. pp. 69-82. 1997.
- Horovitz, Bruce. “Where’s your Domino’s pizza? Track in online.” 28 January 2008. 2008. USA Today, a division of [Gannett Co. Inc.](#)  
<[http://www.usatoday.com/money/industries/food/2008-01-29-pizza-tracker\\_N.htm](http://www.usatoday.com/money/industries/food/2008-01-29-pizza-tracker_N.htm)>
- Laffra, Carla. “Dijkstra’s Shortest Path Algorithm – Applet.” March 1996.  
<<http://www.dgp.toronto.edu/people/JamesStewart/270/9798s/Laffra/DijkstraApplet.html>>
- Lovell, Joe. “Left-Hand-Turn Elimination.” 9 December 2007. The New York Times.  
< <http://www.nytimes.com/2007/12/09/magazine/09left-handturn.html?ex=1354856400&en=c9e577b4f8c25645&ei=5124&partner=permalink&expod=permalink>>