Kyle Duncan

Dr. Joseph Eyles

CS 3304

20 November 2017

Dijkstra’s Algorithm and Its Implementations

Dijkstra’s shortest path algorithm was created by Edsger Wybe Dijkstra. He was born May 11, 1930 in Rotterdam, Netherlands and passed away on August 6, 2002. He began studying computer science before computer science was considered an academic field of study. He struggled with the decision to pursue programming and computer science and was torn between those topics and theoretical physics. While he was in school, he felt lost and asked to speak to his adviser, boss and mentor, Prof. Adriaan van Wijngaarden. In an interview later, Dr. Dijkstra had this to say about that experience:

After having programmed for some three years, I had a discussion with [A. van Wijngaarden](https://en.wikipedia.org/wiki/Adriaan_van_Wijngaarden), who was then my boss at the [Mathematical Center in Amsterdam](https://en.wikipedia.org/wiki/Centrum_Wiskunde_%26_Informatica), a discussion for which I shall remain grateful to him as long as I live. The point was that I was supposed to study theoretical physics at the University of Leiden simultaneously, and as I found the two activities harder and harder to combine, I had to make up my mind, either to stop programming and become a real, respectable [theoretical physicist](https://en.wikipedia.org/wiki/Theoretical_physicist), or to carry my study of physics to a formal completion only, with a minimum of effort, and to become....., yes what? A [programmer](https://en.wikipedia.org/wiki/Programmer)? But was that a respectable profession? For after all, what was [programming](https://en.wikipedia.org/wiki/Computer_programming)? Where was the sound body of knowledge that could support it as an intellectually respectable discipline? I remember quite vividly how I envied my hardware colleagues, who, when asked about their professional competence, could at least point out that they knew everything about vacuum tubes, amplifiers and the rest, whereas I felt that, when faced with that question, I would stand empty-handed. Full of misgivings I knocked on van Wijngaarden's office door, asking him whether I could "speak to him for a moment"; when I left his office a number of hours later, I was another person. For after having listened to my problems patiently, he agreed that up till that moment there was not much of a programming discipline, but then he went on to explain quietly that automatic computers were here to stay, that we were just at the beginning and could not I be one of the persons called to make programming a respectable discipline in the years to come? This was a turning point in my life and I completed my study of physics formally as quickly as I could.

— *Edsger Dijkstra, The Humble Programmer (EWD340),*[*Communications of the ACM*](https://en.wikipedia.org/wiki/Communications_of_the_ACM)

After this discussion, the young Edsger Dijkstra went on to become one of the most important pioneers of modern computing. In 1959, at 29 years old, he published his doctoral thesis called ‘Communication with an Automatic Computer.’ That same year, he wrote a paper called ‘A Note on Two Problems in Connexion with Graphs’ as a proof for his shortest path algorithm. He came up with this algorithm fully in his head, implemented it in a program he wrote to help non-computer people understand the usefulness of the machines and only later recorded the algorithm to submit for peer review.

In the time since he published his algorithm, it has been used in the development of routing protocols and has been adapted for use in search algorithms. It is the basis for the way Google Maps and even MapQuest before it works.

Dijkstra’s algorithm is meant to be used on a graph, or connected network of vertices or nodes. The graph can be a directed or undirected graph, all connections in the graph must have non-negative weights and the graph must be fully connected. The weights can represent whatever aspect of graph is important to the users.

Since the people around him weren’t programmers and didn’t totally understand the implications of what he had created, Dr. Dijkstra created a data structure to hold a map of the Kingdom of the Netherlands and implemented his algorithm to demonstrate the earliest form of computerized point-to-point navigation. His map consisted of some 68 cities in the Netherlands and he used the algorithm to show shortest possible travel distance to each city from the starting point.

For my example, I’m going to use a small map of the Atlanta area to accomplish the same thing. I’ve included Alpharetta, Roswell, Sandy Springs, Vinings, Marietta, Kennesaw, Acworth, Woodstock, Smyrna, Norcross and Atlanta in the map. The weights of the edges connecting each city represent the distance in miles to travel between the cities while taking the largest road possible. In nearly every case, this means we’re using I75, 400 or 285 to get around.

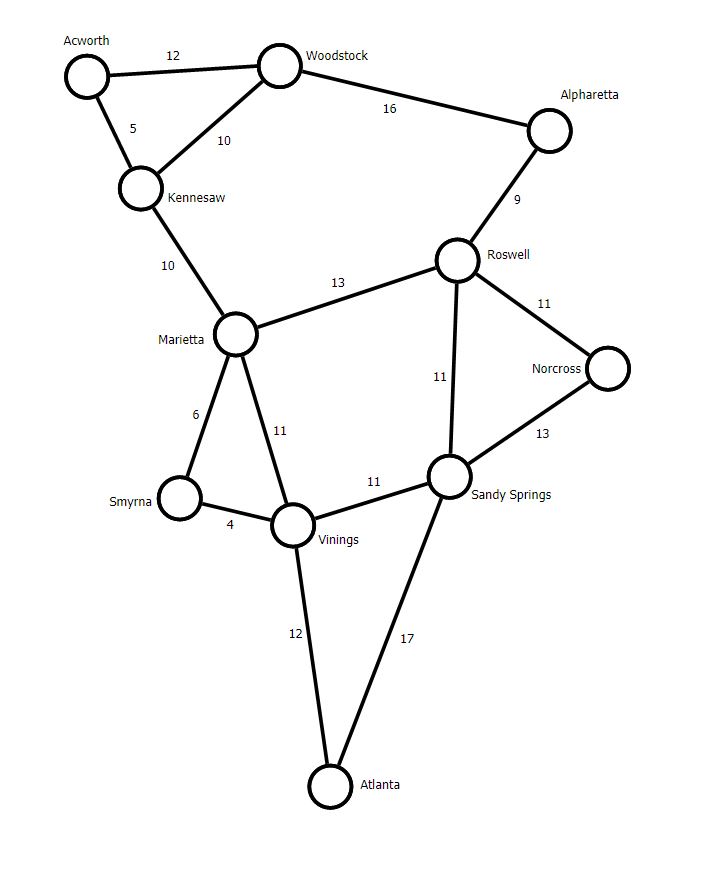


Figure 1 - The map to be used in the creation of the data structure

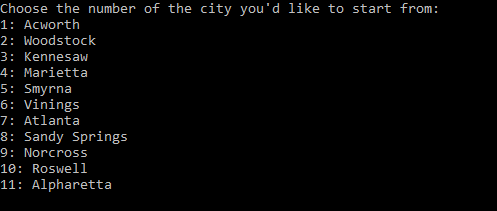


Figure 2 - Console promt for user input

In order to use Dijkstra’s algorithm, we need to pick a starting node – a city on the map above. In my application, if anything other than a listed number is supplied, you will be asked to resubmit a choice until a valid choice is made or the application is terminated. This is done by validating user input using a while statement that tests whether or not the value can be parsed as an integer or, if it *is* an integer, if it falls in the required range. Since the array starts at zero rather than one, one is subtracted from the given value for use in the rest of the application. For output back to the screen, I used a switch function to output each city’s string Name. This was in its own function so that it was easy to use where ever I needed to.

On to the important part of the application: the implantation of Dijkstra’s algorithm. Dijkstra’s algorithm is what’s called a greedy algorithm, meaning it makes the most efficient decisions in each moment rather than considering future possibilities. This is possible because of the way we construct possible paths. When using this algorithm by hand, the easiest way to proceed is to look at the cost of travel to each vertex. Initially, we want to set the cost at each vertex to infinite to indicate that it has not been reached yet. We then start at the home or starting vertex (or city in my application) at take the cheapest connection available and write the weight of that connection at the vertex we’ve just checked. The algorithm is completed by repeating this with every vertex. However, on subsequent vertices, it’s necessary when recording the cost of travel to take the sums of all weights on the path to that vertex. We simply take the path with the lowest sum and keep that path while discarding the more expensive paths. The goal is to end with a spanning tree with the connections that are most efficient **for the starting point**. Note that a spanning tree built this way is not necessarily the most efficient spanning tree available. Primm’s and Kruskal’s algorithms are designed to provide those trees.

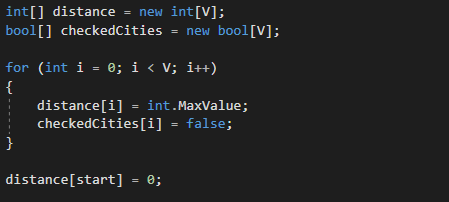
 The way this algorithm is implemented in my code is by first creating an adjacency matrix in a 2D array. The matrix contains the weights of adjacent connections while filling the nonadjacent vertices’ connections with zeros. Additionally, I needed an array to show whether vertices had been checked and needed to be included in the shortest path calculations as well as a third array to store the minimum distance found in those calculations.

Figure 3 – The declaration, creation and filling of the distance and checkedCities arrays.

Note that distance[start] is being initialized as 0 since the distance from start to itself does *not* need to be checked as a possible shortest path later.

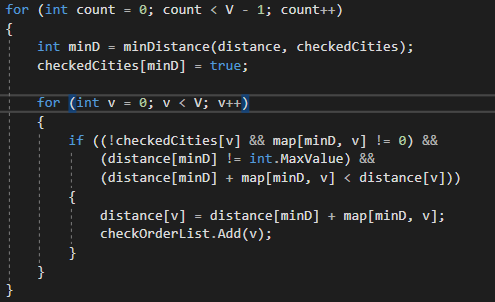


Figure 4 – This for loop iterates through every

The minD variable stores the index of the vertices closest to the one being tested using the minDistance function below. Afterward, the tested vertex is marked as checked.

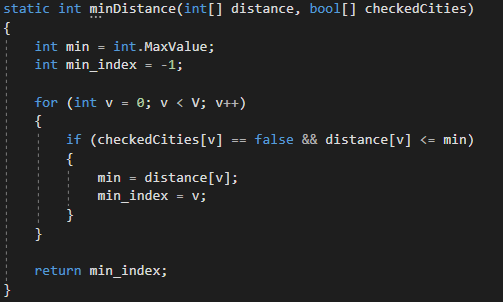


Figure 5 - Ther minDistance function find the closest vertex and returns it's index.

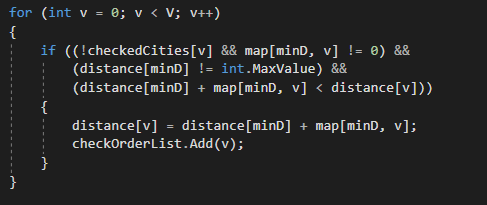
 The minDistance function checks only the vertices that have yet to be checked for a distance value smaller than what’s already stored. Since the every distance was initialized using int.MaxValue, the values are replaced with the smallest checked value as the function iterates through the available vertices using the for loop. Then then index of the smallest value is returned to minD. Note that the distance array is used only to store the smallest distances checked is not changed in this function. Once the checked vertex is marked as done, we need to go back to that long and ugly if statement.

Figure 6 - This slightly unintuitive if-statement is one of the most important parts of this application

The if-statement above is used to update the distance values of unchecked vertices. The first part of the statements requires that a vertex has *not* been checked and that it’s map value is *not* equal to zero. This prevents both the starting city from being included since its value was set to zero as well as any cities that aren’t adjacent to the city being checked. The second part prevents a int.MaxValue distance from being stored in the distance array or causing a rollover error in the final calculation. The third condition simply checks to make sure we don’t generate negative distances. Anything that results in a negative distance would be going in reverse through the map and shouldn’t be checked, but we need to make sure.

The point of this entire if-statement is to make adjustments to the distance values of the remaining vertices. If it weren’t for this statement, we would have to calculate the distance from the starting vertex by summing the weights of every edge along the path – every single time. This step saves time in both hand solving a problem like this and calculating it with a computer.

After every vertex is checked and the distances are recorded, the results are outputted in the console. The results for each city are different and differ from the results we would see if we used Primm’s or Kruskall’s algorithms.

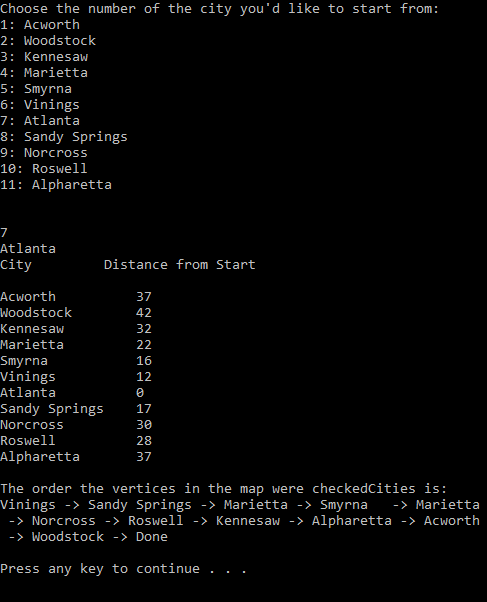
 Above is the output when the application is run using Atlanta as the starting point. The distances listed, as is the point, are all the shortest possible paths with the given connections shown on the map. Also shown is the order of traversal for this instance. Notice after Marietta was checked, Smyrna was checked next because it was the shortest available connection. However, after Smyrna was checked, the next shortest connection was out of Marietta again, rather than Smyrna. If we look at the map, this makes sense.

Figure 7 - Console output after choosing Atlanta as the starting point.

If we wanted to, we could also record the paths taken to each destination. In fact, as I mentioned earlier, every time we run Dijkstra’s algorithm on a graph, the result is a spanning tree that is most efficient for the starting point.

One of the Dr. Dijkstra’s legacies is the simplicity of his work. He always strove to accomplish his proofs and programming goals in the fewest steps in the most elegant was possible. This algorithm is a perfect example of that. His proof is just over two pages long. He came up with it while he was walking between appointments and decided later that it should be written down. Overall, his contributions to modern computer science cannot be overstated. This algorithm alone helps everything from network planning, including things like sewage systems and ISP networks, to GPS navigation with services like Google maps and… well no one uses MapQuest anymore, but it was based on the same concept.

I don’t have much else to say about this algorithm, though I could go on about Dr. Dijkstra for days. After my sources I’ve attached a copy of his proof for this algorithm as well as the obituary published by the University of Texas, where he contributed his knowledge and expertise for 40 years before retiring. He was an incredible man and accomplished some extraordinarily important things. In my opinion, both documents are worth a read.

# References

Anderson, Rodney. *everythingcomputerscience.com (Dijkstra's Algorithm)*. 2013-2017. Web. Oct 2017.

Andrew V. Goldberg, Haim Kaplan, Renato F. Werneck. *Reach for A\*: Efficient Point-to-Point Shortest Path Algorithms*. Technical Report. Redmond, WA: Microsoft Research, 2005. Document.

Byrne, Michael. *motherboard.vice.com (The Simple, Elegant Algorithm That Makes Google Maps Possible)*. 22 Mar 2015. Web. Oct 2017.

Dijkstra, E.W. "A note on Two Problems in Connexion with Graphs." *Numberishe Mathematik* 1959: 269-271. Document.

Dijkstra, Prof. Dr. Edsger W. Interview. Rogier F. van Vlissingen. 3 April 1985. Document.

Gary Chartrand, Ping Zhang. *A First Course In Graph Theory*. Boston: Dover, 2012. Text.

Dr. Dijkstra’s Obituary from University of Texas

Professor Edsger Wybe Dijkstra, a noted pioneer of the science and industry of computing, died after a long struggle with cancer on 6 August 2002 at his home in Nuenen, the Netherlands.

Dijkstra was born in 1930 in Rotterdam, The Netherlands, the son of a chemist father and a mathematician mother. He graduated from the Gymnasium Erasmianum in Rotterdam and obtained degrees in mathematics and theoretical physics from the University of Leyden and a Ph.D. in computing science from the University of Amsterdam. He worked as a programmer at the Mathematisch Centrum, Amsterdam, 1952-62; was professor of mathematics, Eindhoven University of Technology, 1962-1984; and was the Burroughs Corporation research fellow, 1973-1984. He held the Schlumberger Centennial Chair in Computing Sciences at the University of Texas at Austin, 1984-1999, and retired as Professor Emeritus in 1999.

Dijkstra is survived by his wife of over forty years, Maria (Ria) C. Dijkstra Debets, by three children, Marcus J., Femke E., and computer scientist Rutger M. Dijkstra, and by two grandchildren.

Dijkstra was the 1972 recipient of the ACM Turing Award, often viewed as the Nobel Prize for computing. He was a member of the Netherlands Royal Academy of Arts and Sciences, a member of the American Academy of Arts and Sciences, and a Distinguished Fellow of the British Computer Society. He received the 1974 AFIPS Harry Goode Award, the 1982 IEEE Computer Pioneer Award, and the 1989 ACM SIGCSE Award for Outstanding Contributions to Computer Science Education. Athens University of Economics awarded him an honorary doctorate in 2001. In 2002, the C&C Foundation of Japan recognized Dijkstra "for his pioneering contributions to the establishment of the scientific basis for computer software through creative research in basic software theory, algorithm theory, structured programming, and semaphores".

Dijkstra is renowned for the insight that mathematical logic is and must be the basis for sensible computer program construction and for his contributions to mathematical methodology. He is responsible for the idea of building operating systems as explicitly synchronized sequential processes, for the formal development of computer programs, and for the intellectual foundations for the disciplined control of nondeterminacy. He is well known for his amazingly efficient shortest path algorithm and for having designed and coded the first Algol 60 compiler. He was famously the leader in the abolition of the GOTO statement from programming.

Dijkstra was a prodigious writer. His entire collection of over thirteen hundred written works was digitally scanned and is accessible at http://www.cs.utexas.edu/users/EWD. He also corresponded regularly with hundreds of friends and colleagues over the years --not by email but by conventional post. He strenuously preferred the fountain pen to the computer in producing his scholarly output and letters.

Dijkstra was notorious for his wit, eloquence, and way with words, such as in his remark "The question of whether computers can think is like the question of whether submarines can swim"; his advice to a promising researcher, who asked how to select a topic for research: "Do only what only you can do"; and his remark in his Turing Award lecture "In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind."

Dijkstra enriched the language of computing with many concepts and phrases, such as structured programming, separation of concerns, synchronization, deadly embrace, dining philosophers, weakest precondition, guarded command, the excluded miracle, and the famous "semaphores" for controlling computer processes. The Oxford English Dictionary cites his use of the words "vector" and "stack" in a computing context.

Dijkstra enjoyed playing Mozart for his friends on his Boesendorfer piano. He and his wife had a fondness for exploring state and national parks in their Volkswagen bus, dubbed the Touring Machine, in which he wrote many technical papers.

Throughout his scientific career, Dijkstra formulated and pursued the highest academic ideals of scientific rigour untainted by commercial, managerial, or political considerations. Simplicity, beauty, and eloquence were his hallmarks, and his uncompromising insistence on elegance in programming and mathematics was an inspiration to thousands. He judged his own work by the highest standards and set a continuing challenge to his many friends to do the same. For the rest, he willingly undertook the role of Socrates, that of a gadfly to society, repeatedly goading his native and his adoptive country by remarking on the mistakes inherent in fashionable ideas and the dangers of time-serving compromises. Like Socrates, his most significant legacy is to those who engaged with him in small group discussions or scientific correspondence about half-formulated ideas and emerging discoveries. Particularly privileged are those who attended his reading groups in Eindhoven and Austin, known as the "Tuesday Afternoon Clubs".

At Dijkstra's passage, let us recall Phaedo's parting remark about Socrates: "we may truly say that of all the men of his time whom we have known, he was the wisest and justest and best."

