DISCRETE MATHEMATICS

Writing Assignment 2

Authors
Paul Beggs, Thomas Sebring

2nd Semester, 2024

1 Overview

Every day, we rely on applications such as Google Maps to guide us on the fastest route between point A and point B. How does Google's routing system consistently determine the quickest path? This is recognized as the Shortest Path Problem, a fundamental challenge encountered in numerous real-world scenarios and applications. It is characterized by weighted graphs, where each edge is assigned a weight – these weights can be considered as costs or "travel times" as exemplified by Google Maps.

The Shortest Path Problem is not only a crucial aspect of navigation applications but is also a commonly encountered issue in various fields such as logistics, network design, and resource optimization. This paper will explore the principles of weighted graphs, delve into an algorithm that is used to solve the Shortest Path Problem. This exploration will provide us with valuable insights into the computational foundations behind optimizing routes and paths in diverse scenarios. (For demonstration purposes, we will only be using positive integers.)

2 The Shortest Path

For any two points, there exists the shortest distance path between them. However, the shortest distance might not be a linear path, or line, between the two points. If two points are separated by more than one non-linear path, it might not be so easy to decide which path between the points is the shortest distance. Hence, as the name suggests, the Shortest Path Problem is finding what path between two points is the shortest possible distance.

Weighted Graphs

To determine the shortest total distance, we must identify all potential paths that can be traversed. Weighted graphs prove invaluable in this regard, as they illustrate each point and the respective paths connecting them. Each path is assigned a weight or distance value, signifying the distance between two points. By leveraging these distances, algorithms can be employed to calculate the potential distances between any two points and identify the shortest possible distance.

3 Dijkstra's Algorithm

3.1 Using Djikstra's Algorithm

Dijkstra's Algorithm is one of many algorithms that calculates the possible distances between any two points to find the shortest possible distance. It requires maintaining a separate list of points that have been visited during the traversal of the algorithm. In summary, the algorithm involves four major steps:

1. Assign a distance value of 0 to the starting point and set all other points in the graph to a value of ∞.
2. Move to the starting point. Add the starting point to the list of visited points. Assign each of the points adjacent to the starting point a value equal to the distance value from the starting point. The starting point cannot be revisited.
3. Move to the adjacent point with the least value, making it the current point. Examine each point adjacent to the current point that is not in the list of visited points. Calculate the distance value from the adjacent point to the current point, adding it to the value of the current point. If the calculated value is less than the current value of the adjacent point, update the adjacent point with the calculated value; otherwise, leave it unchanged. The current point cannot be revisited.
4. Repeat the previous step if there are adjacent points not in the list of visited points. Once all reachable points have been processed, the algorithm concludes, and each point has been assigned the shortest possible distance to reach it.

To find the shortest possible distance between any two points, we use the separate list from our graph. We can extract the list to get the shortest possible distance up to our ending point.

3.2 Examples

In the context of our graph analysis, the designated starting point will consistently be labeled as point a, and our goal is to determine the shortest distance between point a and point z. We will employ a straightforward table representation to track the current values of each point in our weighted graph and the list of points moved to, denoted as S, using Dijkstra's Algorithm. We will halt the algorithm when we reach z. Note that circled values in the graph represent the point with the shortest distance, and boxed values represent points that have been traversed.

Example 1

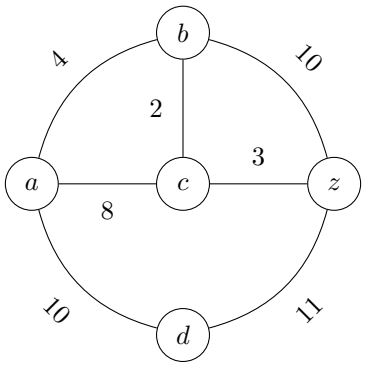

Figure 1: Graph for Example 1

Step	a	b	c	d	z	S
0	⓪	∞	∞	∞	∞	$\emptyset$
1	0	④	8	10	∞	$\{a\}$
2	0	4	⑥	10	14	$\{a, b\}$
3	0	4	6	10	⑨	$\{a, b, c\}$
4	0	4	6	⑩	9	$\{a, b, c, z\}$

Walkthrough

1. We will start Step 0 by assigning our starting point a value of 0 and all other points a value of ∞. 0 is our smallest point value so it is circled.

2. For Step 1, we move to our starting point and add it to our list S. A box is placed around point a's value. It is adjacent to points b, c, and d so we assign each adjacent point a value equal to the distance from our starting point. Point b has the shortest distance, so it is circled.

3. For Step 2, we move to point b because it was the adjacent point with the shortest distance, and add it to our list S. A box is placed around point b's value. We then look at the adjacent points, not in S, points c and z. Calculate the distance value from each adjacent point to the current point plus the value of the current point, if it is lower than the adjacent point's current value, assign it the calculated value. Point c has a calculated value of 6, which is less than its current value of 8, so it is assigned. Point z has a calculated value of 14, which is less than its current value of ∞, so it is assigned. Point c has the shortest distance so it is circled.

4. For Step 3, we move to point c because it was the adjacent point with the shortest distance, and add it to our list S. A box is placed around point c's value. We look at the adjacent points not in S, point z. Calculate the distance value from the adjacent point to the current point plus the value of the current point, if it is lower than the adjacent point's current value, assign it the calculated value. Point z has

a calculated value of 9, which is less than its current value of 14, so it is assigned. Point z has the shortest distance so it is circled.

5. For step 4, we move to point z because it was the adjacent point with the shortest distance, and add it to our list S. A box is placed around point z's value. We look at the adjacent points not in S, point d. Calculate the distance value from the adjacent point to the current point plus the value of the current point, if it is lower than the adjacent point's current value, assign it the calculated value. Point d has a calculated value of 20, which is not less than its current value of 10, so it is not assigned. Point d has the shortest distance so it is circled. Since we've reached point z, we can halt our algorithm. Finally, we can extract S as our shortest distance to point z.

Example 2

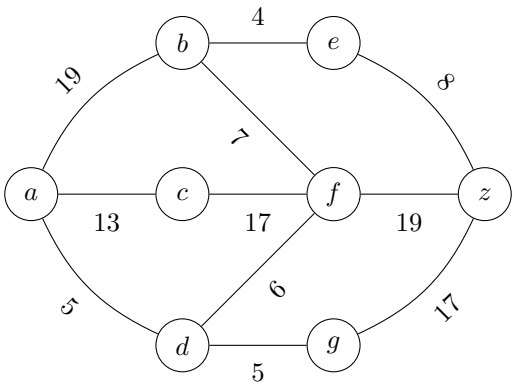

Figure 2: Graph for Example 2

Step	a	b	c	d	e	f	g	z	S
0	⓪	∞	∞	∞	∞	∞	∞	∞	$\emptyset$
1	0	19	13	⑤	∞	∞	∞	∞	$\{a\}$
2	0	19	13	5	∞	11	⑩	∞	$\{a, d\}$
3	0	19	13	5	∞	11	10	㉗	$\{a, d, g\}$
4	0	19	13	5	35	⑪	10	27	$\{a, d, g, z\}$

Walkthrough

1. We will start Step 0 by assigning our starting point a value of 0 and all other points a value of ∞. 0 is our smallest point value so it is circled.

2. For Step 1, we move to our starting point and add it to our list S. A box is placed around point a's value. It is adjacent to points b, c, and d so we assign each adjacent point a value equal to the distance from our starting point. Point d has the shortest distance, so it is circled.

3. For Step 2, we move to point d because it was the adjacent point with the shortest distance, and add it to our list S. A box is placed around point d. We then look at the adjacent points, not in S, points f and g. Calculate the distance value from each adjacent point to the current point plus the value of the current point, if it is lower than the adjacent point's current value, assign it the calculated value. Point f has a calculated value of 11, which is less than its current value of ∞, so it is assigned. Point g has a calculated value of 10, which is less than its current value of ∞, so it is assigned. Point g has the shortest distance so it is circled.

4. For Step 3, we move to point g because it was the adjacent point with the shortest distance, and add it to our list S. A box is placed around point g. We look at the adjacent points not in S, point z. Calculate the distance value from the adjacent point to the current point plus the value of the current point, if it is lower than the adjacent point's current value, assign it the calculated value. Point z has a calculated value of 28, which is less than its current value of ∞, so it is assigned. Point z has the shortest distance so it is circled.

5. For step 4, we move to point z, because it was the adjacent point with the shortest distance, and add it to our list S. A box is placed around point z. We look at the adjacent points not in S, points f and c. Calculate the distance value from the adjacent point to the current point plus the value of the current point, if it is lower than the adjacent point's current value, assign it the calculated value. Point f has a calculated value of 36, which is not less than its current value of 11, so it is not assigned. Point c has a calculated value of 35, which is less than its current value of ∞, so it is not assigned. Point f has the shortest distance so it is circled. We will halt the algorithm because point z has been reached. Finally, we can extract S as our shortest distance to point z.

4 Conclusion

The world continually encounters the Shortest Path Problem, as discussed in our overview, exemplified by mapping applications like Google Maps. By illustrating Dijkstra's algorithm, we aim to present one of several solutions to this issue. Furthermore, we have successfully illustrated one of the pivotal aspects of Dijkstra's algorithm through our examples: time complexity. In Example 2, despite nearly doubling the potential points to visit, a solution was attained in the same number of steps. This crucial characteristic forms the basis for practical real-world applications, not solely theoretical ones. Additionally, we have devised an accessible method for coding a version of Dijkstra's algorithm by delineating the algorithm's steps in simple language and demonstrating how data might be stored by a computer using our tables.