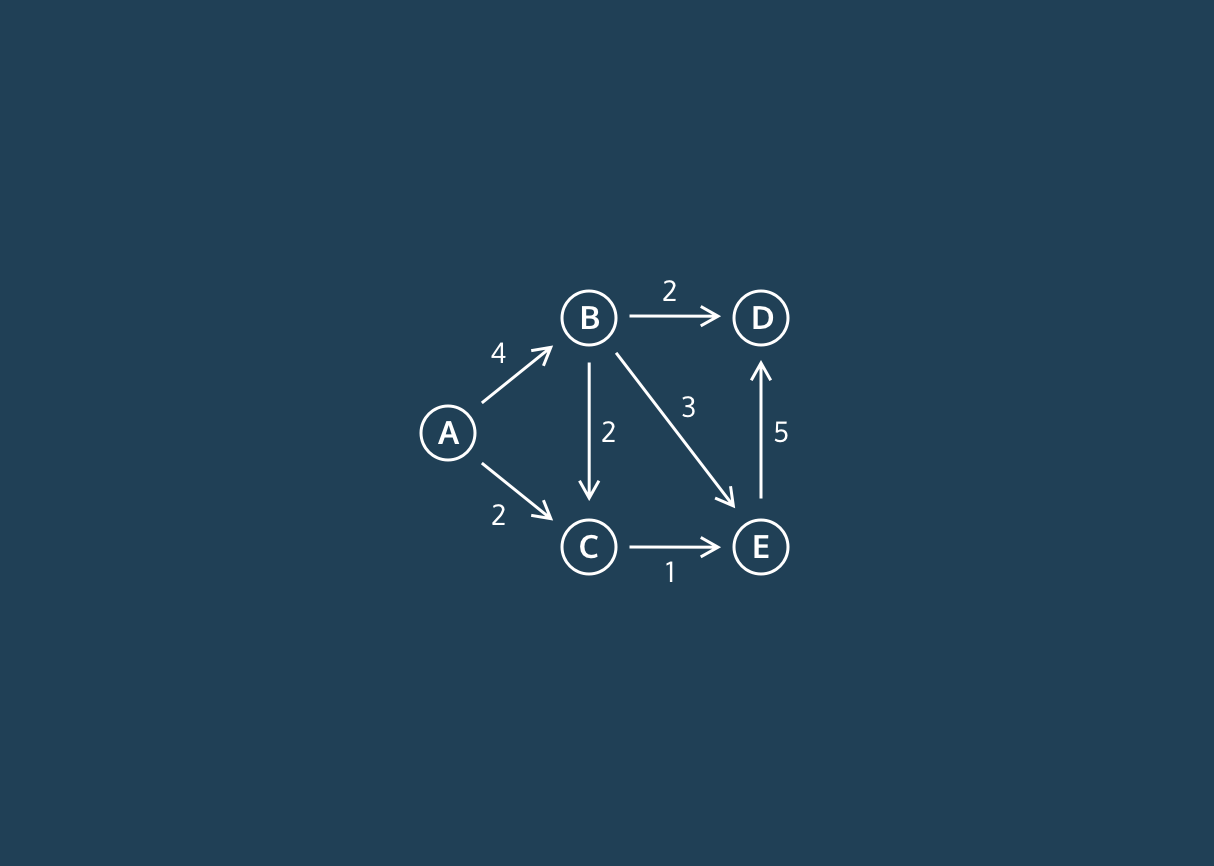
**Intro to Dijkstra's Algorithm: Conceptual**

Now that we know how to properly search a graph, how can we use these skills in real life?

One of the most common applications of graph searches is to find the shortest distance between vertices. Finding this distance has a variety of applications such as finding the optimal route to a destination or transferring data in a computer network.

Take a look at the graph below. Finding the shortest path between vertex A and vertex E may seem easy in your brain, but telling a computer how to find it is a bit more complicated.



Fortunately, there is an algorithm that computes the shortest distance from a given vertex to the rest of the vertices in a graph. This is called ***Dijkstra’s Algorithm***.

Dijkstra’s Algorithm works as following:

1. Instantiate a dictionary that will eventually map vertices to their distance from the start vertex
2. Assign the start vertex a distance of 0 in a min heap
3. Assign every other vertex a distance of infinity in a min heap
4. Remove the vertex with the smallest distance from the min heap and set that to the current vertex
5. For the current vertex, consider all of its adjacent vertices and calculate the distance to them as **(distance to the current vertex) + (edge weight of current vertex to adjacent vertex)**.
6. If this new distance is less than the current distance, replace the current distance.
7. Repeat 4 and 5 until the heap is empty
8. After the heap is empty, return the distances

That may seem confusing! Be sure to check out the video in order to better visualize this algorithm.

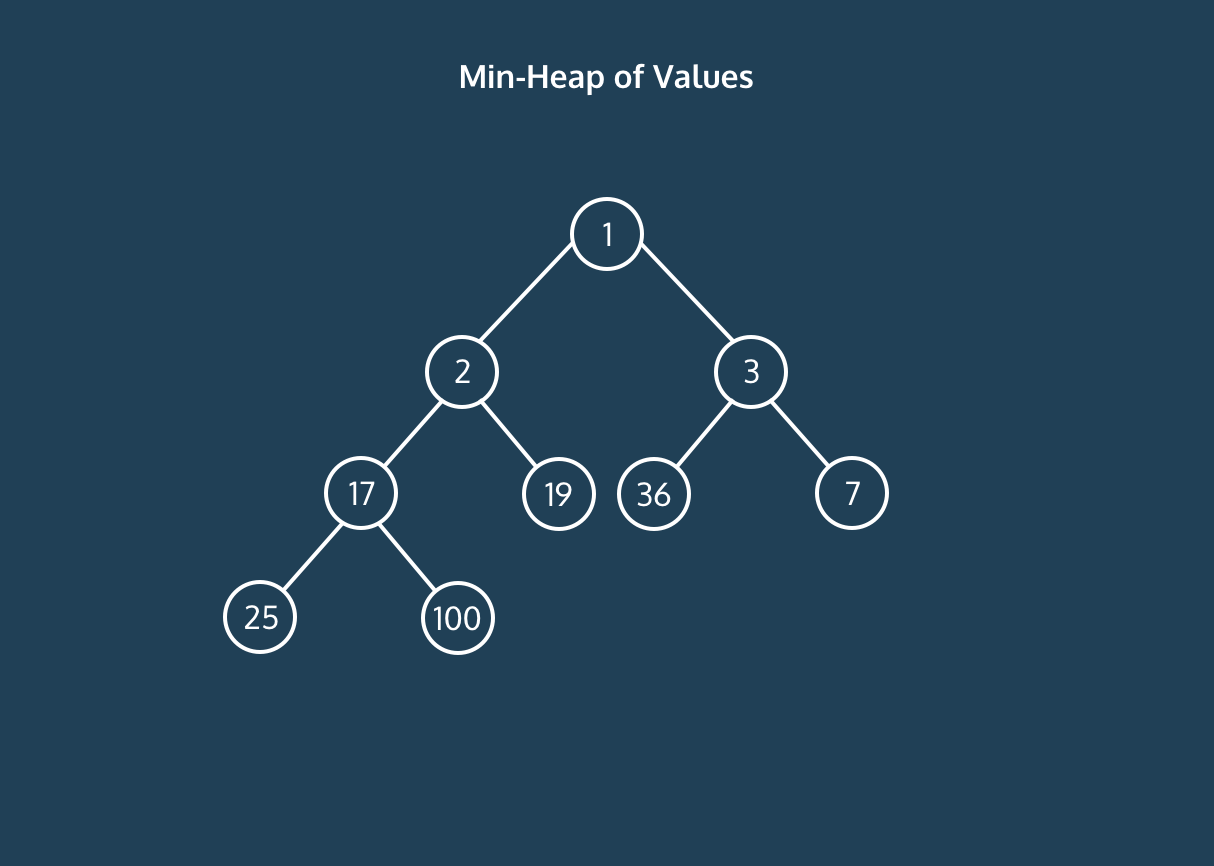
**Dijkstras Algorithm: Conceptual Runtime**

How efficient is Dijkstra’s algorithm? Let’s break it into different parts:

* Searching through the graph
* Keeping track of distances

Just like breadth-first search and depth-first search, to search through an entire graph, in the worst case, we would go through all of the edges and all of the vertices resulting in a runtime of O(E + V).

For Dijkstra’s, we use a min-heap to keep track of all the distances. Searching through and updating a min-heap with V nodes takes O(log V) because in each layer of the min-heap, we reduce the number of nodes we are looking at by a factor of 2.



In the worst case, we would update the min-heap every iteration. Since there are **at most E + V iterations** of Dijkstra’s and **it takes log V to update** a min-heap in the worst case, then the runtime of Dijkstra’s is **O((E+V)log V)**.

**Dijkstras Algorithm Review**

Congratulations on grasping a conceptual understanding of Dijkstra’s Algorithm!

Here’s a quick recap of what you’ve learned:

* Dijkstra’s algorithm is an algorithm to find all of the shortest distances between a start vertex and the rest of the vertices in a graph.
* The algorithm works by keeping track of all the distances and updating the distances as it conducts a breadth-first search.
* Dijkstra’s algorithm runs in **O((E+V)log V).**

<https://content.codecademy.com/programs/cs-path/dijkstra's%20conceptual/Dijkstra's%20Conceptual.mp4>

**Quiz: Dijkstra's Algorithm: Conceptual**

**Why must we initialize all the distances from the start vertex to infinity at the beginning of Dijkstra’s algorithm?**

* Since we haven’t searched the graph yet, we don’t know the distances, and thus, we must assume the maximum possible distance.

**True or False: Dijkstra’s algorithm follows a breadth first search rather than a depth-first-search?**

* True; we check all of the neighbors of our current vertex in each iteration, which is a key property of a breadth first search.

**Why does Dijkstra’s algorithm run in O((V+E)log V)?**

* In the worst case, we will visit **all V+E vertices and edges**. In each visit, we may have to update our min-heap which **takes log V time**. Thus, the runtime is **O((V+E) log V))**.

**Would Dijkstra’s Algorithm be able to work with negative edge weights?**

* No; We always consider the distance up to our current vertex to be the shortest possible distance. However, a negative edge would mean that edges after the current vertex could lead to a shorter distance. This breaks our algorithm.