CSCI 2270 – Data Structures

Recitation 11, April 2020

Dijkstra's Algorithm

Objectives

1. BFS for Shortest Path
2. Weighted graphs and Dijkstra's algorithm

Please refer to the previous recitation document for DFS(Depth First Search).

The shortest path between two vertices in a graph is the one where the sum of weights of edges in the path is minimal.

BFS for Shortest Path:

BFS(Breadth-First Search) is used to compute the shortest path between any two vertices in an **unweighted** graph. In an unweighted graph, the **path length** is proportional to the number of vertices in the path.

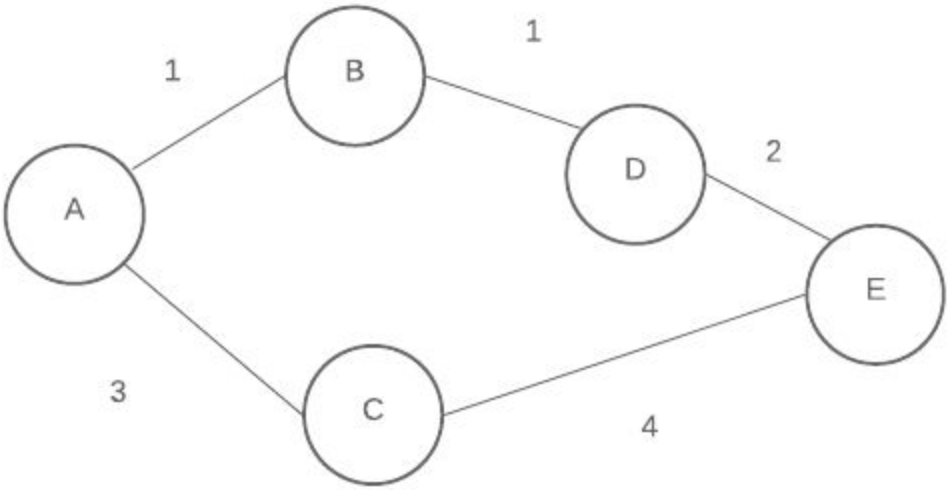
BFS exhausts all possible vertices at a level (say n) before moving on to vertices at a level - n+1. The claim for BFS is that the first time a node is discovered during the traversal, that distance from the source would give us the shortest path for that node.

The same claim cannot be applied for a weighted graph.
The following example will make it more clear.

CSCI 2270 – Data Structures

Recitation 11, April 2020

Dijkstra's Algorithm



For the above graph, if we apply BFS, it would give the shortest path as:

A ->C->E (**length** : 3+4 = 7)

But, the path **A->B->D->E** is shorter (**length**: 1+1+2 = 4)

Thus, in a weighted graph, a smaller number of vertices in the path need not imply a shorter path. (Since each edge could have any weight).
To solve this, we use **Dijkstra's** algorithm.

How does **Dijkstra's** solve it?

Dijkstra's algorithm works by marking one vertex at a time as it discovers the shortest path to that vertex. In this process, it helps to get the shortest distance from the source vertex to every other vertex in the graph. It keeps updating these distances for the vertices in increasing order of path length and re-uses them to compute the shortest distance of the destination. Let's look at the algorithm.

CSCI 2270 – Data Structures

Recitation 11, April 2020

Dijkstra's Algorithm

Dijkstra's algorithm:

Each vertex has a **distance** measure and a **solved** boolean flag.

- **distance -** stores the shortest path length from the source vertex.
- **solved -** tells if the shortest path for that vertex from the source has been found.
- **parent -** stores the parent of a vertex found during the traversal.

Initialization:

The **solved** fields for each vertex are initialized using below:

	solved
Source (A)	TRUE
Every other vertex	FALSE

Once, the shortest path for a vertex is found, we update the **solved** flag (to TRUE) and set the **distance** field for that vertex**.** We also have a **solvedList** which at any point in time stores the list of vertices for which the shortest distance has been computed. **Initially,** it has just the **Source(A)** in it.

The **algorithm** can be described as below:

Till **destination** is not solved, do:

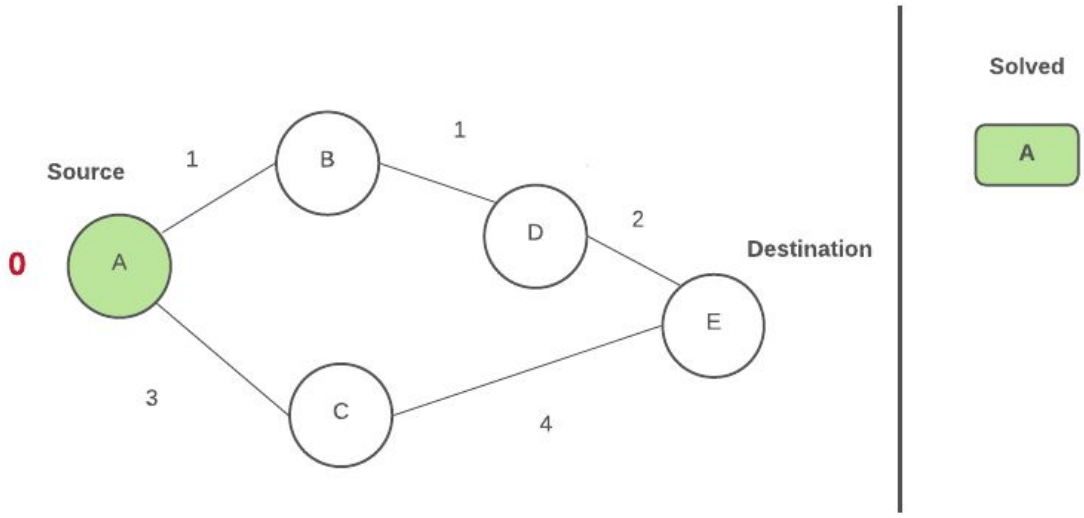
- For each element **s** in the **solvedList,** we look at the adjacent vertices that aren't solved yet.
- Among all these vertices, get the vertex whose distance (this would be parent's distance + weight of the edge) from the source is the shortest. Then mark it as **solved, set** its **distance field** and add it to the **solvedList.**

Look at the illustration below to understand it better!

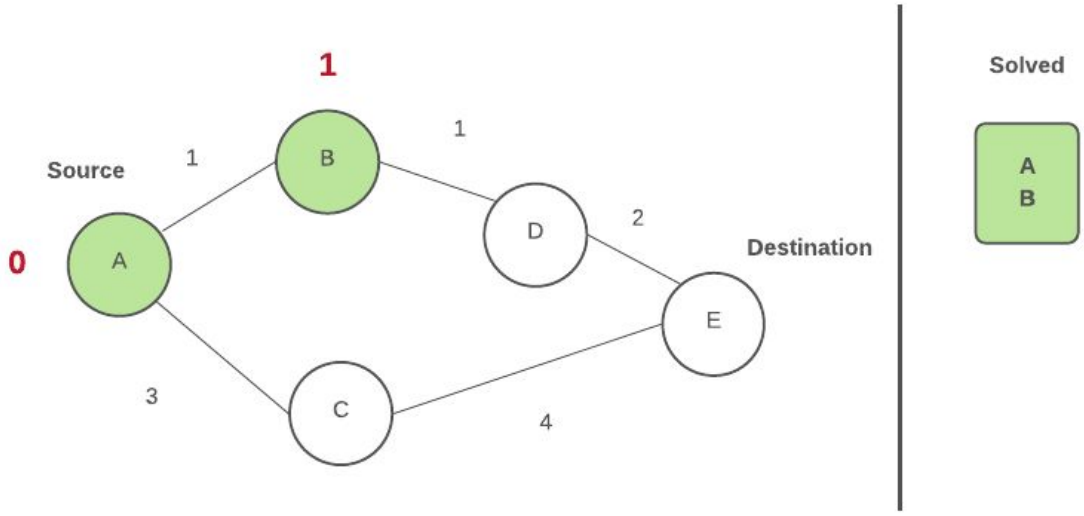
Recitation 11, April 2020

Dijkstra's Algorithm

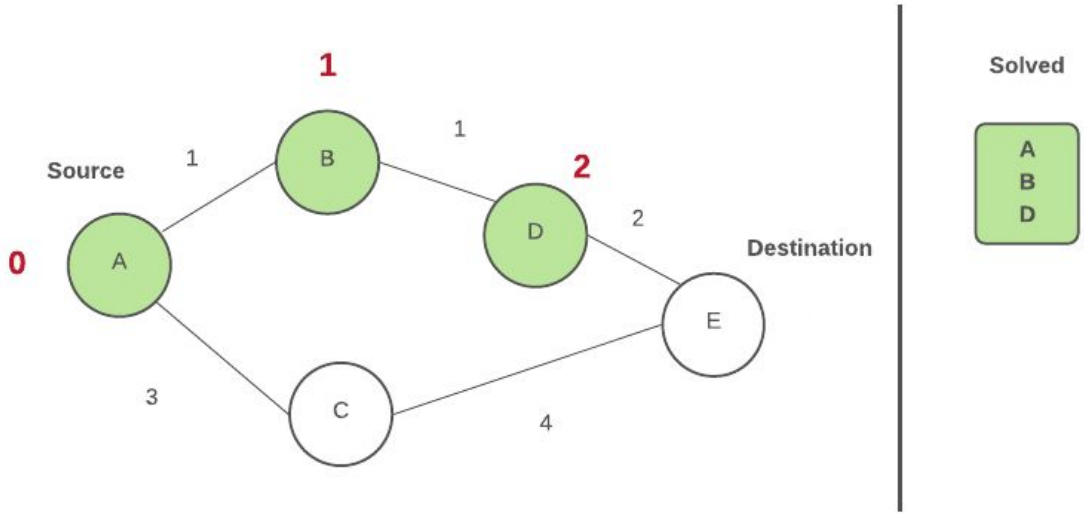
1- Initial State: (**Green** - **Solved** vertex)



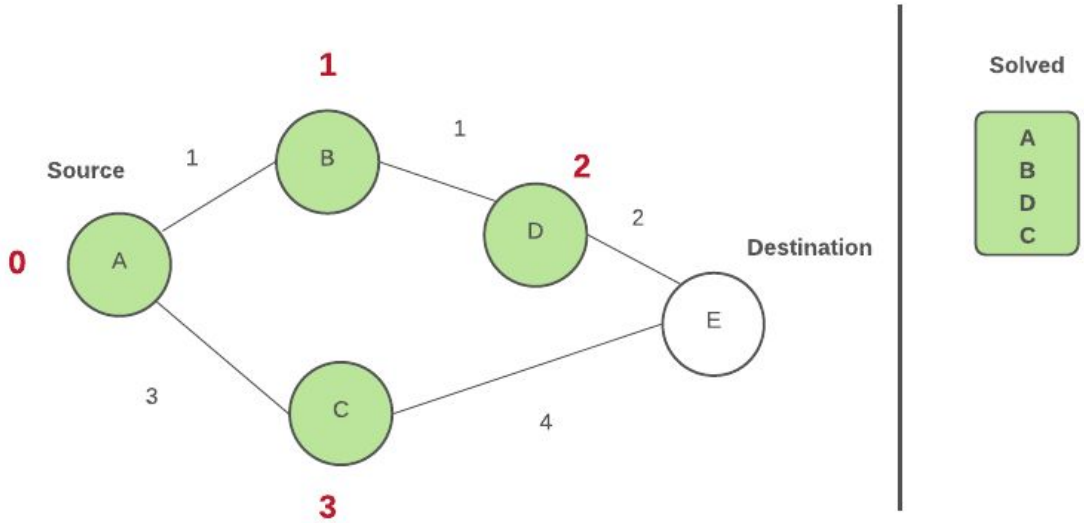
2 - Iterate over all adjacent vertices of A, set B's distance from Source(A):



3 - Iterate over all adjacent vertices of A and B, set D's distance from Source(A):

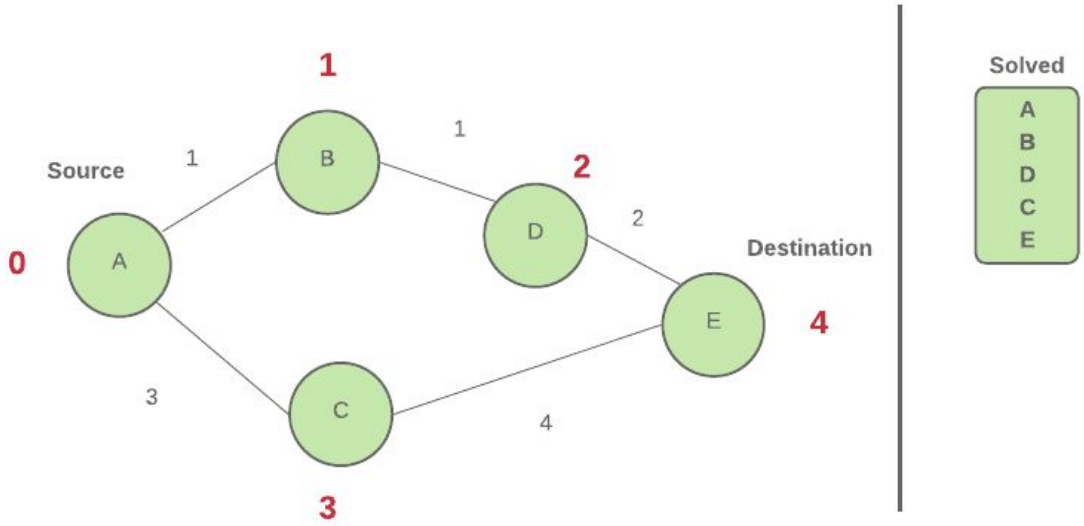


4 - Iterate over all adjacent vertices of A, B, and D and set C's distance from Source(A):



Dijkstra's Algorithm

5 - Iterate over all adjacent vertices of A, B, D, and C and set E's distance from Source(A):



In doing the above steps, we get the shortest path length from source A to all the vertices in the graph. Hence, Dijkstra is one of the ways to compute single-source shortest paths to every vertex. Note that, we have **solved** the vertices in increasing order of shortest path length from the source.

CSCI 2270 – Data Structures

Recitation 11, April 2020

Dijkstra's Algorithm

Exercise

A. Silver Badge Problem (Mandatory)

1. Download the Recitation11 **zipped** file from Moodle.
2. Follow the TODOs and in Graph.cpp, complete the **isBridge()** function.
3. This function finds if an edge is a bridge of the graph. (A bridge is an edge of a graph whose deletion increases its number of connected components.)