CSCI 2270 – Data Structures

Recitation 10

Dijsktra Algorithms

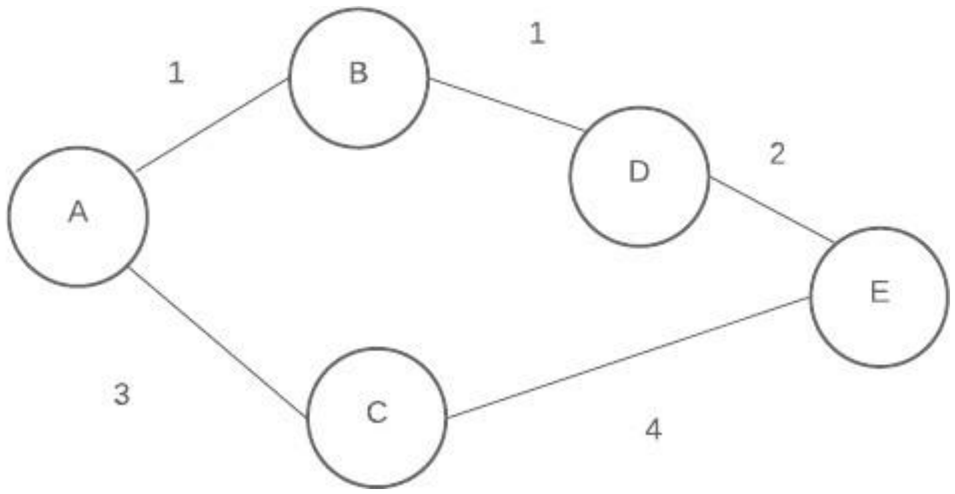
BFS for Shortest Path:

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

The way BFS works is, it exhausts all possible vertices at a level (say n) before moving on to vertices at a level - n+1.
In an unweighted graph, the **path length** is proportional to the number of vertices in the path.

Hence, moving level(n) after level(n+1) in increasing order helps us find the shortest path.
as we are considering paths of increasing length.

The problem with weighted graphs:



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 also exhausts all possible paths of length n before considering paths of length greater than n. But unlike BFS, it doesn't use the level of the vertex for this computation. It uses the weights of the edges.
It uses a list to store the shortest distance from the source vertex to every other vertex in the graph. It keeps updating this list in increasing order of path length and re-uses them to compute the shortest distance of the destination. Let's look at the algorithm.

The Dijkstra's algorithm:

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

- **distance -** measures the distance from the source vertex.
- **solved -** tells if the shortest path for that vertex from the source has been found.

Initialization:

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

	distance	**solved**
Source (A)	0	TRUE
Every other vertex	∞(infinity)	FALSE

Initially , we assume that every vertex apart from the source is not reachable (hence the distance is set to infinity). Also, shortest path from source to source is of length 0.
Once, the shortest path for a vertex is found, we update the **solved** flag to TRUE and the **distance** field for that vertex**.** This step is called the **relax** operation.

CSCI 2270 – Data Structures

Recitation 10

Dijsktra Algorithms

We also have a **solvedList** which at any point of time stores the list of vertices for which the shortest distance has been computed. **Initially** it has just the **Source(A)** in it.

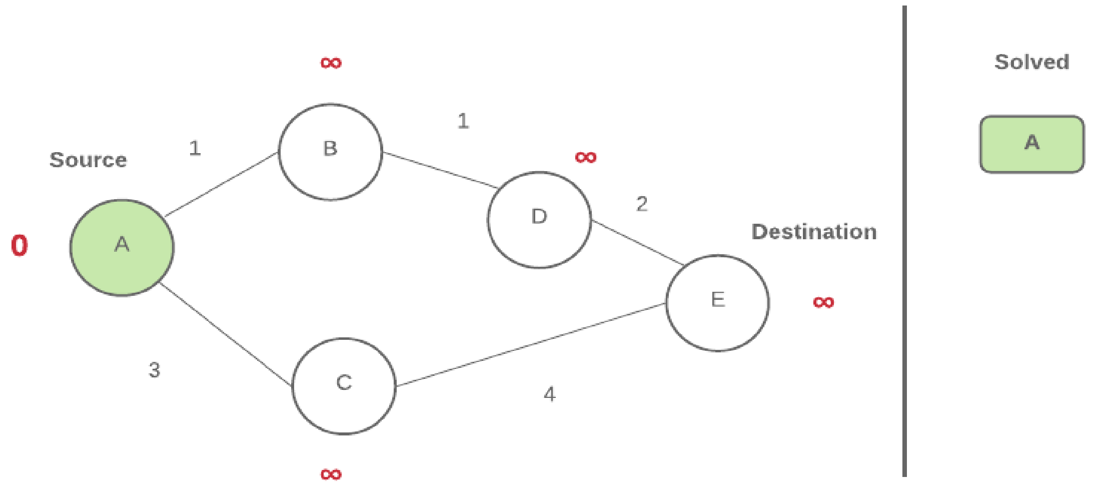
The **algorithm** can be described as below:

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

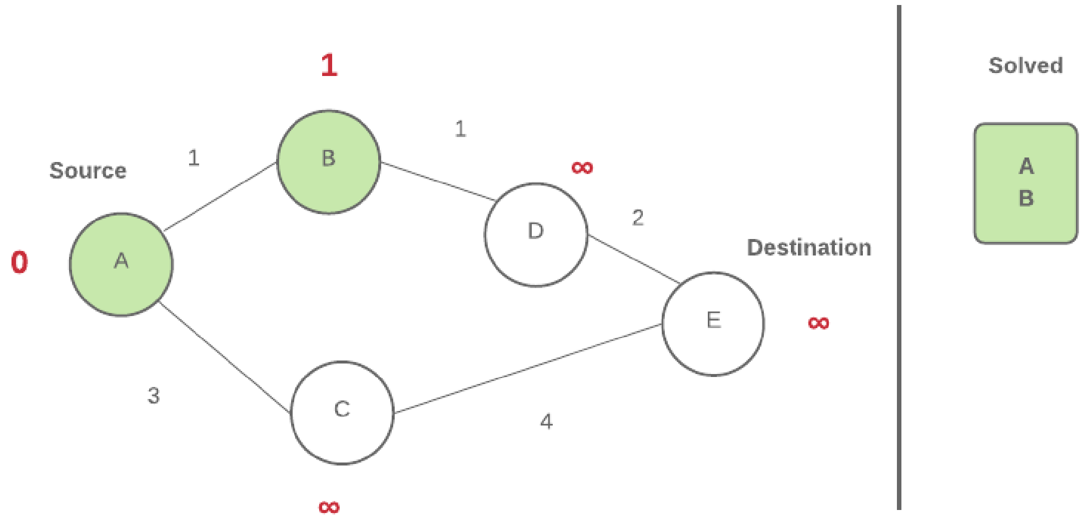
- For each element,**s** in the **solvedList,** we traverse over the adjacent vertices which aren't solved yet.
- Among all the above traversed vertices, perform the **relax** operation only on the one whose distance from the source, is the shortest.
  (Relax operation on a vertex is marking it as **solved, updating** the **distance variable** and adding it to the **solvedList)**

Look at the illustration below to understand it better.

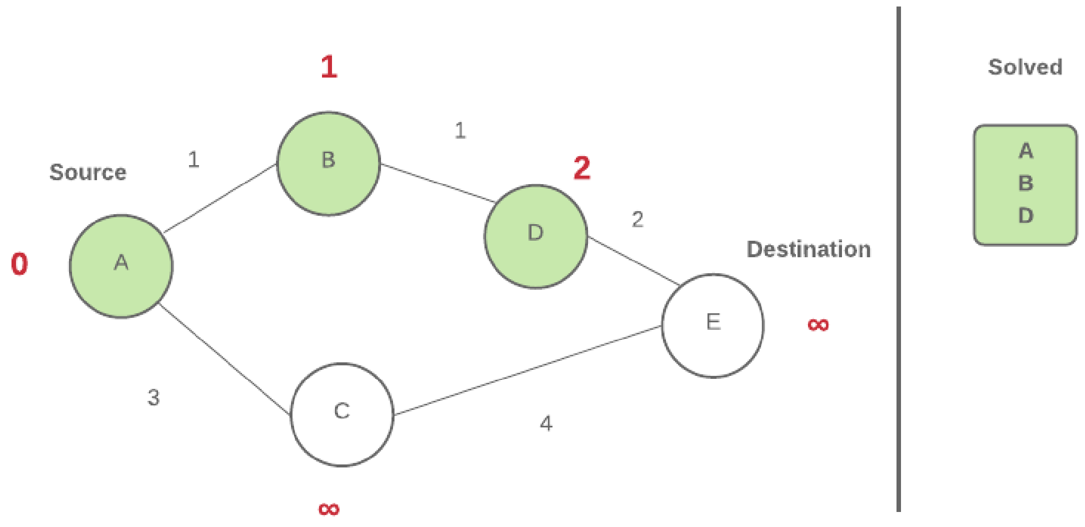
1- Initial State: (**Green - Solved** vertex)



2 - Iterate over all children of A, and relax B’s distance from Source(A):



3 - Iterate over all children of A and B, and relax D’s distance from Source(A):

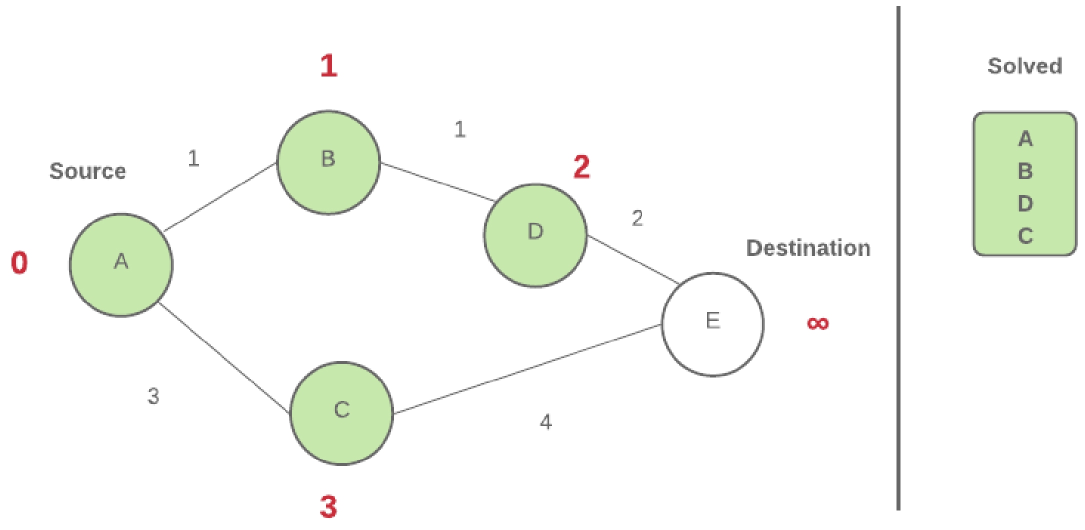


4 - Iterate over all children of A,B and D and set C’s distance from Source(A):

CSCI 2270 – Data Structures

Recitation 10

Dijsktra Algorithms

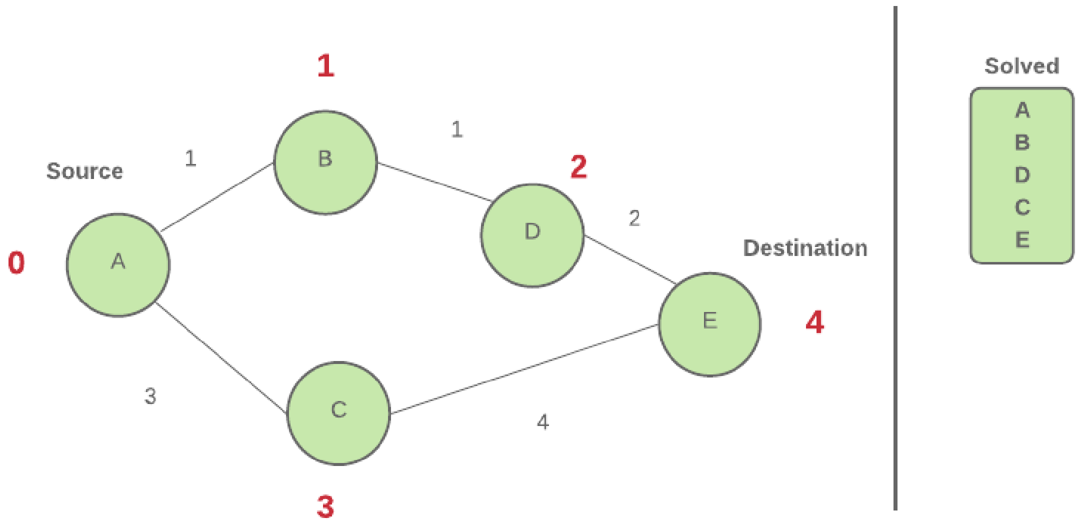


5 - Iterate over all children of A,B,D and C and set E’s distance from Source(A):

CSCI 2270 – Data Structures

Recitation 10

Dijsktra Algorithms



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.

Exercise

A. Silver Badge Problem (Mandatory)

1. Accept the assignment on Canvas, clone the repo.
2. Follow the TODOs and in Graph.cpp, complete the **isBridge()** function.

   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. You will be using DFS (Depth First Search) to solve this problem. Please refer to the previous recitation document for DFS).

CSCI 2270 – Data Structures

Recitation 10

Dijsktra Algorithms