Kennesaw State University

College of Computing and Software Engineering

Department of Computer Science

CS 5070, Mathematical Structures for Computer Science, Section W01

Algorithm Project (See ya later, Algo-gator)

Amrit Singh, asingh59@students.kennesaw.edu

07/22/2025

Problem Statement

An algorithm is a step-by-step way to solve a problem. More formally, an algorithm is a set of rules or a computational procedure that is typically used to solve a specific problem [1, p. 5]. Graphs are an abstract way of representing connectivity using nodes, vertices, and edges [2, slide 3]. There are many algorithms that solve specific graph problems. We'll be studying four of them – a search algorithm that deals with graphs, Bellman-Ford algorithm, Floyd-Warshall algorithm, and Prim's algorithm.

Summary / Purpose

Purpose of this document is to provide a clear and detailed explanation of the step-by-step workings of each algorithm. We'll cover practical use cases and scenarios where each algorithm is used, along with visual aids.

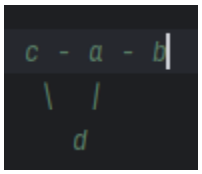
Solutions

Graphs

Formally, a graph is an ordered pair $G = (V, E)$ consisting of a nonempty set V (called the vertices) and a set E (called the edge) of two-element subsets of V [3, p. 234]. For example, a graph G could be represented the following:

$$G = (V, E) = (\{a, b, c, d\}, \{\{a, b\}, \{a, c\}, \{a\ d\}, \{c, d\}\})$$

This can be drawn like so:



Graphs are connected if we can get from any vertex (or node) to any other vertex by following some path of edges [4]. An edge is a line between two vertices. An edge can have an associated weight, represented by some function w known as a weight function [1, p. 591]. Weights can indicate the strength, or another attribute such as cost, distance, etc, of each connection between the nodes [6]. Graphs can be either directed, undirected, or both. For directed graphs, edges have a specific direction in which we can traverse, also known as unidirectional, or it can be bidirectional. A unidirectional edge signifies a one-way relationship between the two vertices. A bidirectional edge represents a two-way relationship [6].

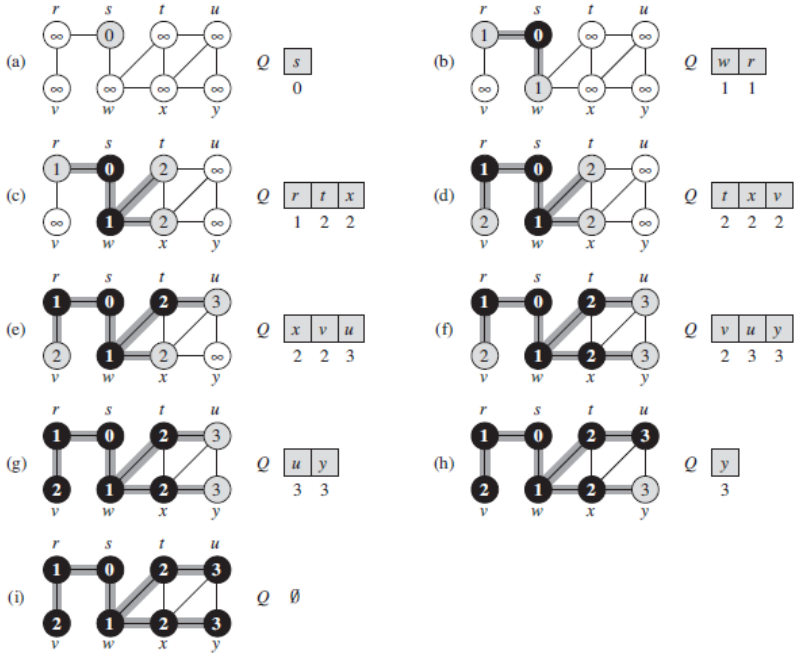
A Search Algorithm* (BFS)

There are many search algorithms for graphs. A fundamental example of which is the bread-first search (BFS) algorithm. It sets the basis for more advanced graph algorithms, like Prim's algorithm and Dijkstra's shortest path algorithm [1, p. 594]. Given a graph G, and a distinguished

source vertex *s*, breadth-first search explores the edges of the source vertex *s* to find any vertex *v* that can be reached from the source *s*. It works by visiting adjacent vertices, and then visits those vertices' adjacent vertices until all vertices are traversed. Doing so will build a tree where the root is *s* [8]. Visited vertices are often placed into a data structure such as a queue. The algorithm will keep adding a vertex's neighbors into the queue as it's dequeued, while processing the vertex that was dequeued if need be until the queue is exhausted. This algorithm is represented by the following [8]:

1. **Initialization:** Enqueue the given source vertex into a queue and mark it as visited.
2. **Exploration:** While the queue is not empty:
   - Dequeue a node from the queue and visit it (e.g., print its value).
   - For each unvisited neighbor of the dequeued node:
   - Enqueue the neighbor into the queue.
   - Mark the neighbor as visited.
3. **Termination:** Repeat step 2 until the queue is empty.

A step-by-step operation of BFS on an undirected graph is below. The Q queue is filled as we visit neighbors from the source *s* node [1, p. 596].



Bellman-Ford Algorithm

The shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized [5]. Edge weights may or may not be negative.

A technique, known as relaxation, is used to test if we can improve the shortest path to a node so far by traversing to that node from a shorter path. For an edge $(u \rightarrow v)$ with weight w, we can denote a shorter path to v from a source node s. Relaxation technique checks whether or not it is best to go through to v from s or to go to v via an edge $u \rightarrow v$, updating the distance to get to v [1, p. 648).

To do this, we'll initialize the distance to go to from s to v infinity. The distance to go from s to s is 0. See the following pseudocode:

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

Relaxation may decrease the value of the shortest path to v and update v's preceding value. The process of relaxing is done repeatedly by many algorithms, such as the Bellman-Ford Algorithm. See the following pseudocode for relaxation method [1, p. 649]:

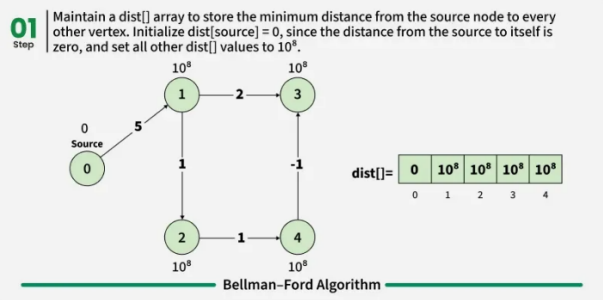
```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

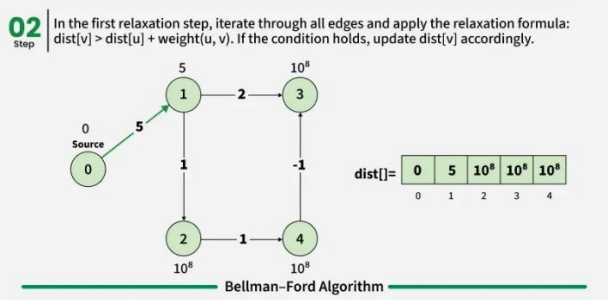
The Bellman-Ford algorithm solves the shortest paths problem in the general case in which edge weights may be negative from a source to all other vertices. [1, p. 651]. The algorithm will produce the shortest paths and their weights, unless there is a negative-weight cycle that is reachable from the source. If such a cycle exists, then we'll have no solution.

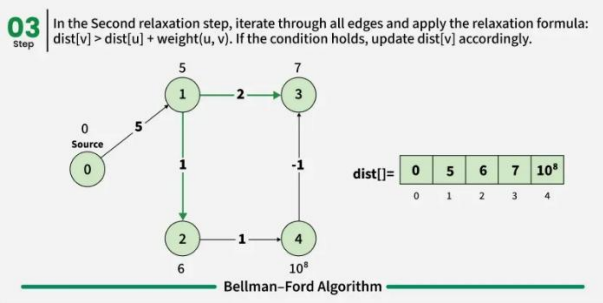
This algorithm will repeatedly relax the distance to v (i.e weight) of a shortest apth from the source s to each vertex in the graph until we get our shortest-path distance (i.e weight). If we detect a negative weight cycle, we will return false else we will return true:

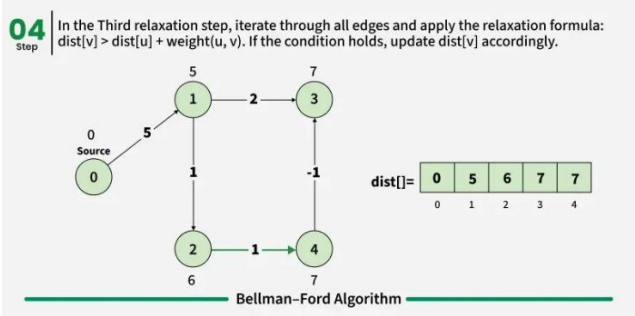
```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

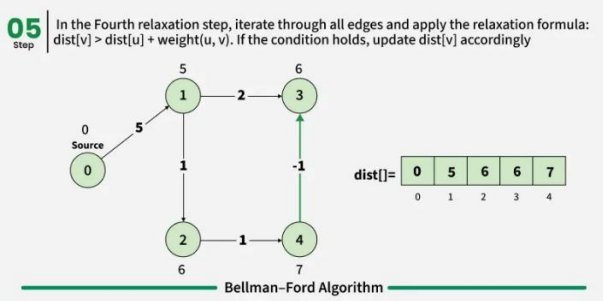
The following example illustrates the algorithm at work, note that 10^8 represents infinity, and an array is to keep track of the minimum distance from the source node to every vertex [7].

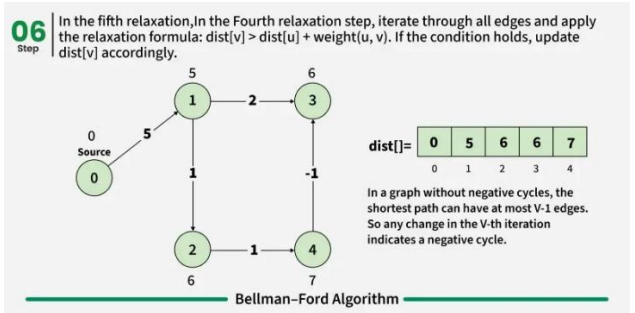




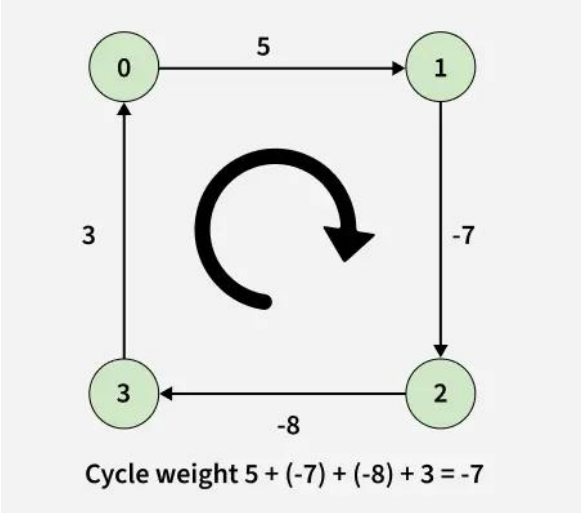








If a negative weight cycle is detected, then the shortest path doesn't exist every cycle the shortest path will continue to decrease [7]. See the following example:



This algorithm is best used when there are negative and non-negative weights in a graph system. Examples in real life scenarios may include network routing, financial applications, social networks, logistics, and emergency services, making it a very practical algorithm to apply to many systems.

Floyd-Warshall Algorithm

Dynamic programming combines solutions to overlapping subproblems in order to find a solution [1, p. 359]. We typically try to find an optimal solution by finding the structure of it, recurse to define a value of the optimal solution, compute the value from the bottom up, and finally construct the solution from our information [1, p. 359]. This method of solving problems can typically be approached from the top down via memoization, i.e store existing solutions to subproblems to use again, or from a bottom up approach, where we solve smaller sub problems to combine later [1, p. 365].

The Floyd-Warshall algorithm is a dynamic programming algorithm to find the shortest paths between all pairs of vertices in a weighted and directed graph. The weights can be positive or negative, with no negative weight cycles. This algorithm can be used to find a table of distances between all pairs of cities in an atlas (i.e Geographical Information Systems (GIS) applications), finding the shortest path between all pairs of nodes in a network (i.e network routing), or flight connectivity between airports [1, p. 684][9].

Suppose we have a matrix dist[][] of size n x n where dist[i][j] represents the weight from node i to node j. If a direct edge does not exist, dist[i][j] is set to a large value (representing infinity). Since the distance from a node to itself is 0, diagonal entries are set to 0.

We maintain a two-dimensional array that is the distance between nodes that are initially filled using only the direct edge between nodes. This array will gradually update the distances by checking to see if shorter paths exist through intermediate nodes [9]. See the following pseudocode [1, p. 695]:

```
FLOYD-WARSHALL(W)
1  n = W.rows
2  D^(0) = W
3  for k = 1 to n
4      let D^(k) = (d_ij^(k)) be a new n × n matrix
5      for i = 1 to n
6          for j = 1 to n
7              d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
8  return D^(n)
```

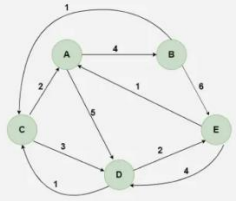
A step by step implementation is addressed in the following [9]:

- **Start by updating the distance matrix** by treating each vertex as a possible intermediate node between all pairs of vertices.
- **Iterate through each vertex**, one at a time. For each selected vertex k, attempt to improve the shortest paths that pass through it.
- When we pick vertex number **k** as an intermediate vertex, we already have considered vertices **{0, 1, 2, .. k-1}** as intermediate vertices.
- For every pair **(i, j)** of the source and destination vertices respectively, there are two possible cases.
  - **k** is not an intermediate vertex in shortest path from **i** to **j**. We keep the value of **dist[i][j]** as it is.
  - **k** is an intermediate vertex in shortest path from **i** to **j**. We update the value of **dist[i][j]** as **dist[i][k] + dist[k][j]**, if **dist[i][j] > dist[i][k] + dist[k][j]**
- Repeat this process for each vertex k until all intermediate possibilities have been considered.

The following is an illustration of the algorithm at work [9]:

01 Step **Understanding the Problem**: Given a matrix dist[n][n] where dist[i][j] represents the weight from node i to j (e.g., A → B → C). If no direct edge exists, dist[i][j] = INF(e.g. 10^8)and for all nodes, dist[i][i] = 0.

	A	B	C	D	E
A	0	4	10^8	5	10^8
B	10^8	0	1	10^8	6
C	2	10^8	0	3	10^8
D	10^8	10^8	1	0	2
E	1	10^8	10^8	4	0



Floyd Warshall Algorithm

02 Step Treat first node (e.g. A) as an intermediate node and calculate dist[][] for every (i, j) node using the formula dist[i][j] = min(dist[i][j], dist[i][A] + dis[A][j]).

	A	B	C	D	E
A	0	4	10^8	5	10^8
B	10^8	0	1	10^8	6
C	2	10^8	0	3	10^8
D	10^8	10^8	1	0	2
E	1	10^8	10^8	4	0

	A	B	C	D	E
A	0	4	10^8	5	10^8
B	10^8	0	1	10^8	6
C	2	6	0	3	10^8
D	10^8	10^8	1	0	2
E	1	5	10^8	4	0

Floyd Warshall Algorithm

03 Step Treat second node (e.g. B) as an intermediate node and calculate dist[][] for every (i, j) node using the formula dist[i][j] = min(dist[i][j], dist[i][B] + dis[B][j]).

	A	B	C	D	E
A	0	4	10^8	5	10^8
B	10^8	0	1	10^8	6
C	2	6	0	3	10^8
D	10^8	10^8	1	0	2
E	1	5	10^8	4	0

	A	B	C	D	E
A	0	4	5	5	10
B	10^8	0	1	10^8	6
C	2	6	0	3	12
D	10^8	10^8	1	0	2
E	1	5	6	4	0

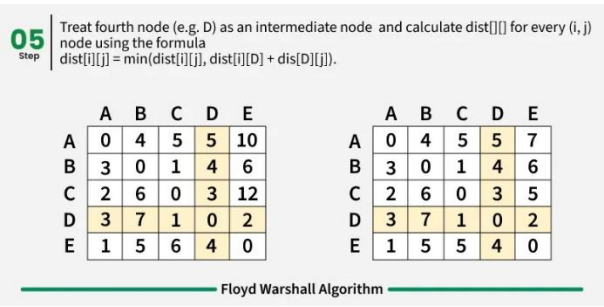
Floyd Warshall Algorithm

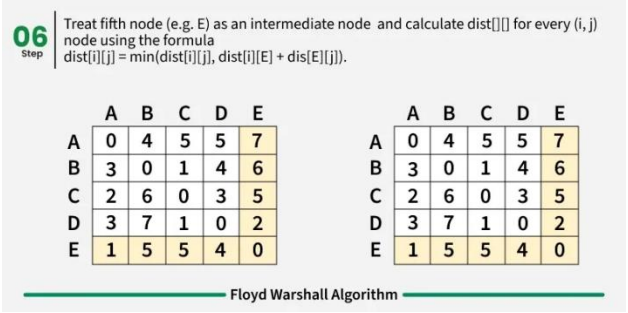
04 Step Treat third node (e.g. C) as an intermediate node and calculate dist[][] for every (i, j) node using the formula dist[i][j] = min(dist[i][j], dist[i][C] + dis[C][j]).

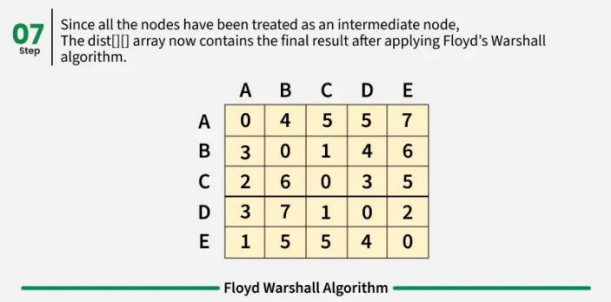
	A	B	C	D	E
A	0	4	5	5	10
B	10^8	0	1	10^8	6
C	2	6	0	3	12
D	10^8	10^8	1	0	2
E	1	5	6	4	0

	A	B	C	D	E
A	0	4	5	5	10
B	3	0	1	4	6
C	2	6	0	3	12
D	3	7	1	0	2
E	1	5	6	4	0

Floyd Warshall Algorithm







Prim's Algorithm

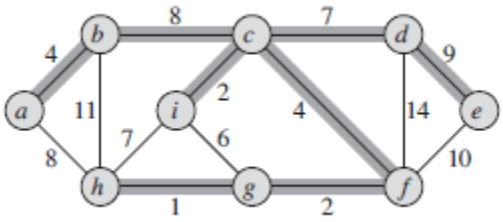
A spanning tree is an acyclic tree subgraph of a connected, undirected graph that includes all vertices of the graph [10]. The minimum spanning tree (MST) is also a tree subgraph that is very similar to a spanning tree, but it also has the smallest possible weights possible in the tree. There can be several solutions to a MST type problem [10]. Examples of use cases may include telecommunications or a network.

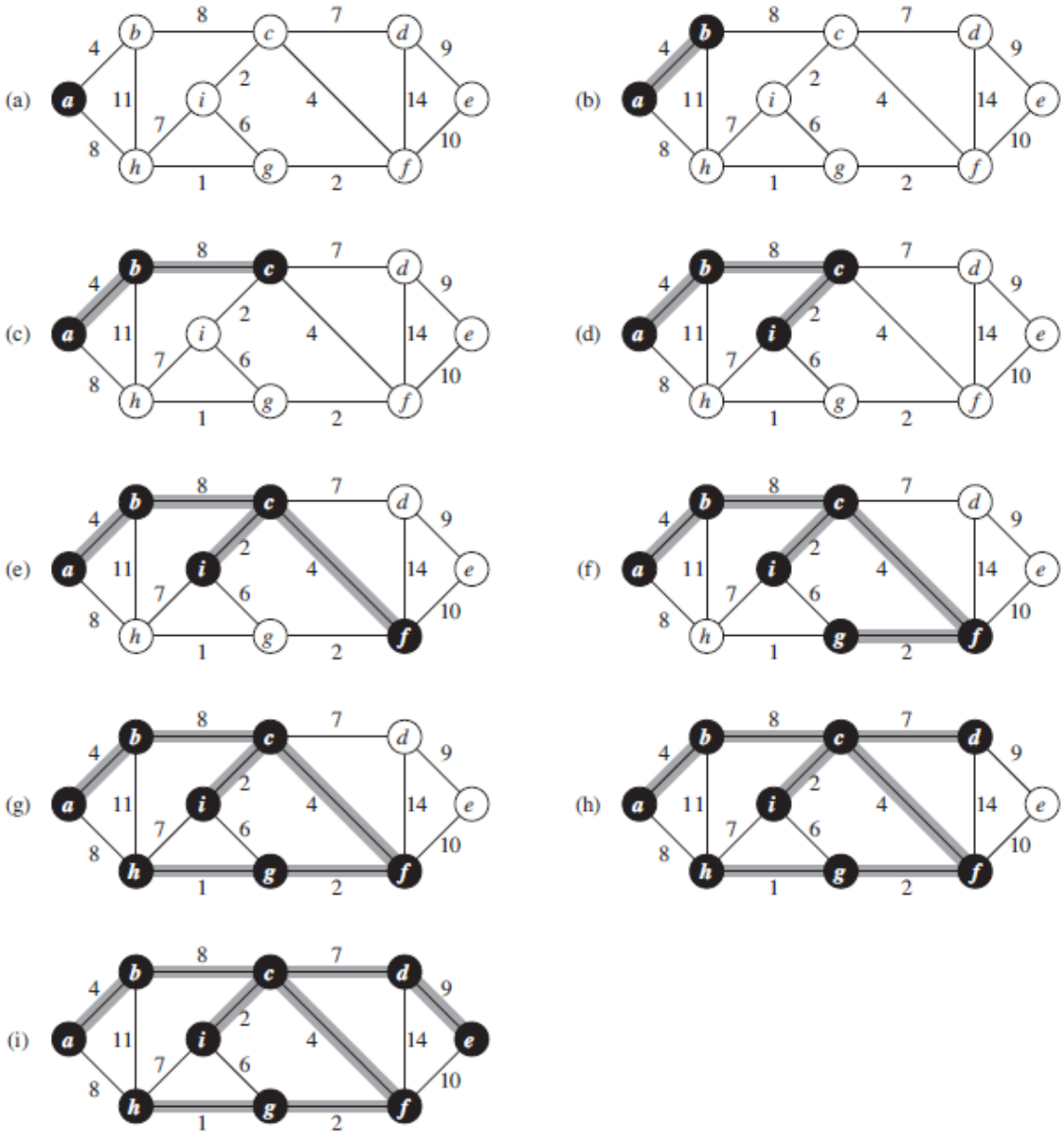
One such algorithm to arrive to a solution is called Prim's algorithm. It is a greedy algorithm that is a special case of the generic minimum tree spanning algorithm [1, p. 634]. The algorithm starts from a single node *r* and grows until the tree encompasses all connected nodes in the graph *G*. Two sets are maintained, one that contains nodes that are already in the MST and the other is one with nodes that are not yet in the MST. For each step, the algorithm considers all edges that connect the two sets and selects the one with the smallest weight. The algorithm then adds the edge and its opposite endpoint to the set containing the MST [1, p. 634][11].

While the algorithm runs, all vertices not included in the tree are stored in a minimum priority queue *Q*, ordered by a key attribute. For each node *v*, the value *v.key* represents the minimum weight of any a edge connecting *v* to a node already in the tree. By convention, if no such edge exists *v*.key is infinite. The attribute *v.pi* indicates the parent of *v* in the tree. The following is the pseudocode for this algorithm [1, p. 634].

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

The following is the execution of Prim's algorithm on the graph depicted [1, p. 635, 635]. The root of the tree is vertex a. Shaded edges represent those already included in the growing tree, and black-colored vertices are those that are part of the tree. At each step, the vertices currently in the tree define a cut in the graph, and the algorithm adds a light edge (i.e., the minimum-weight edge) that crosses this cut. For instance, in the second step, the algorithm can choose to add either edge (b, c) or edge (a, h), as both are light edges crossing the current cut.





Comments / Conclusions

There are many algorithms that solve complex problems that apply to real life scenarios. Identifying and choosing the correct approach to a problem can help systemically solve the problem or even develop a program to solve those problems. We learned of four major graph algorithms in this paper, the inner works of each, and where we can apply them. Learning these are vital for understanding graph theory.

References

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms, third edition*. MIT Press.

[2] https://web.stanford.edu/class/cs97si/06-basic-graph-algorithms.pdf

[3] Levin, O. (2016). *Discrete mathematics: An Open Introduction*.

[4] Garrido, J. (2021, August 14). *CS5070 Mathematical Structures for Computer Science - Notes 6* [Slide show; Powerpoint]. D2L.

[5] *The Shortest-Path Problem. Synthesis Lectures on Theoretical Computer Science. 2015. doi:10.1007/978-3-031-02574-7. ISBN 978-3-031-01446-8.*

[6] https://www.mathworks.com/help/matlab/math/directed-and-undirected-graphs.html

[7] https://www.geeksforgeeks.org/dsa/bellman-ford-algorithm-dp-23/#

[8] https://www.geeksforgeeks.org/dsa/breadth-first-search-or-bfs-for-a-graph/

[9] https://www.geeksforgeeks.org/dsa/floyd-warshall-algorithm-dp-16/#

[10] https://www.geeksforgeeks.org/dsa/what-is-minimum-spanning-tree-mst/#

[11] https://www.geeksforgeeks.org/dsa/prims-minimum-spanning-tree-mst-greedy-algo-5/