Hoang Thinh Pham

R11771069

CS 3364 – Algorithms

Project 3 Report

**Problem interpretation**

The project's primary objective revolves around determining the shortest paths from the Computer Science (CS) building to various other buildings within a campus environment. To achieve this, two fundamental algorithmic strategies, Greedy (Dijkstra's Algorithm) and Dynamic Programming (Bellman-Ford Algorithm), are implemented and evaluated. The problem centers on utilizing graph structures to model the campus layout and employing these algorithms to compute the shortest routes efficiently.

**Methodology**

The programming language of choice for this project is Java, and since Java was used, the code base of this project is based on object-oriented paradigm.

Graph Representation and Data Structures:

* Graph Representation: The campus layout is represented as a graph where buildings are vertices, and pathways between them are edges.
* Vertex and Edge Representation: The Vertex class represents a building, containing pertinent information such as building name and unique identifier. Edges between vertices are defined through the Edge class, considering weights representing distances between buildings.
* Algorithm Implementation: Dijkstra's algorithm (Dijkstra) and Bellman-Ford algorithm (BellmanFord) are implemented to find the shortest paths. The algorithms in the class lectures are modified to implement using Java language, for example, the dist[] and prev[] arrays has been changed to HashMap, the use of HashMap helps keeping track and identify buildings, makes coding and debugging easier.

Algorithm for the method findPath() of Dijkstra class:

public *void* findPath() {

dist.put(s, 0);

H.add(s);

while (!H.isEmpty()) {

Vertex u = H.poll();

for (Edge e : u.getEdges()) {

Vertex v = e.getDestination();

*int* l = e.getWeight();

*int* alt = dist.get(u) + l;

if (alt < dist.get(v)) {

dist.put(v, alt);

prev.put(v, u);

// Remove and add v to update its position in the priority queue

H.remove(v);

H.add(v);

}

}

}

}

The method uses Java PriorityQueue to keep track and update the shortest paths, then the folowing printShortestPath method can be used to extract the previous nodes stored in prev to print out the building along the path:

public *void* printShortestPath(Vertex *source*, Vertex *destination*) {

*List*<Vertex> path = **new** ArrayList<>();

for (Vertex v = *destination*; v != null; v = prev.get(v)) {

path.add(v);

}

Collections.reverse(path);

for (*int* i = 0; i < path.size(); i++) {

System.out.print(path.get(i).getId());

if (i < path.size() - 1) {

System.out.print(" -> ");

}

}

System.out.println();

}

Similarly, Bellman-Ford algorithms can be implemented by modifying the current Dijkstra code:

public *void* findPath() {

dist.put(s, 0);

for (*int* i = 0; i < g.getVertices().size() - 1; i++) {

for (Vertex u : g.getVertices()) {

for (Edge e : u.getEdges()) {

Vertex v = e.getDestination();

*int* l = e.getWeight();

*int* alt = dist.get(u) + l;

if (dist.get(v) > alt) {

dist.put(v, alt);

prev.put(v, u);

}

}

}

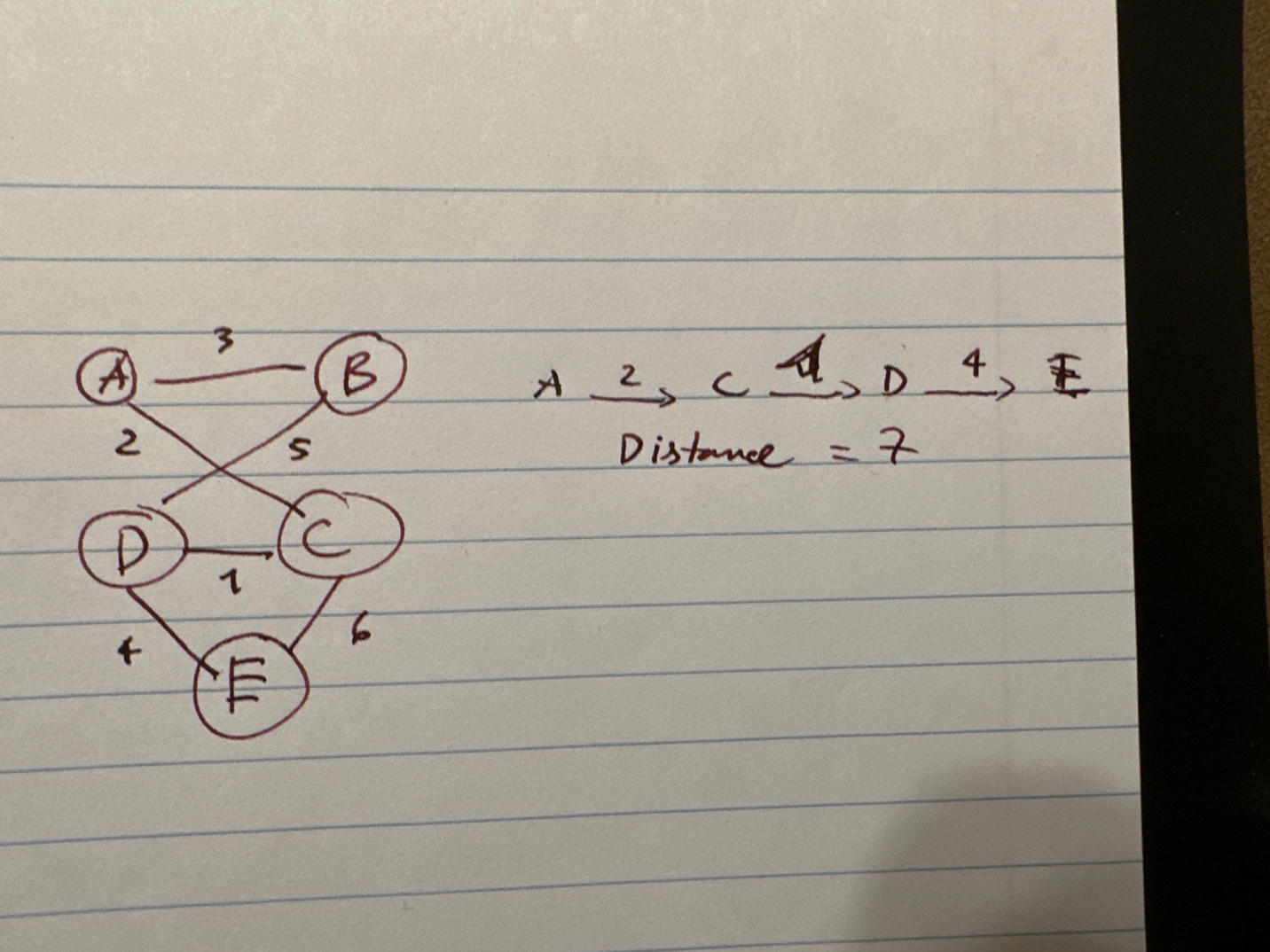
}

}

Experimental Results

The expected result is the marching numbers and path from both algorithms since there are only one shortest path from the CS building to specific destination.

Simple mock data is created to test the algorithms with manual verification and yielded correct results:



A computer screen shot of a program

Description automatically generated

A screenshot of a computer

Description automatically generated

Conclusion

The program yields similar results and are correct results based on manual verification. The choice of Java is suitable for this kind of project since I work alone and Java language help minimizing several intermediate steps since it provide some data structures that languages like C doesn’t. Java also help further development with extra feature such as visual representation of the graph and algorithms via GUI and animations.