CS 361 – Lab #12 – Weighted Graphs

The previous lab was about DAG's, and this week we'll look at adding weights to the edges.

Given a weighted graph, one of the most fundamental questions is finding the shortest distance from one vertex to another. In the basic case, we have Dijkstra's algorithm. However, this cannot handle negative weighted edges. We have seen 2 other algorithms which take care of negative edges, namely the Bellman-Ford algorithm which solves the general case but is a little tedious, and then the DAG shortest path algorithm. In today's lab we will look at the latter case. We are interested in weighted directed acyclic graphs.

A popular application of the weighted DAG is the PERT chart. PERT stands for "Program Evaluation and Review Technique." These are used in management to model a project that is broken down into tasks which take various amounts of time. In this weighted DAG, the vertices represent states or milestones within the overall project. The edges correspond to performing a job, and the weight is an amount of time, e.g. measured in days or weeks of work needed to complete that single job.

The DAG shortest path algorithm can be helpful to analyzing a PERT chart, because it can help determine the shortest amount of time needed to progress from one milestone to some future one. Or, in particular the time needed to do the entire project from beginning to end.

Of course, our algorithm can model more than just PERT charts because it can handle negative edges as well as positive.

The first step today is to modify or extend the Graph class you worked on last week so that it can handle edge weights. It was originally set up so that the adjacency matrix entries were 1 for the presence of an edge and 0 for the absence. Now, we need to change the representation so that 0 represents the edge going to itself (no edge necessary), and "infinity" for the absence of an edge. Your addEdge() method should also take a weight parameter.

Shortest Path

Next, you will implement the DAG shortest path algorithm we discussed in class. Given a weighted DAG and a "source" vertex, compute the shortest path to all the other (future) vertices. Here is the pseudocode:

```
Topologically sort the graph.  You did this in the previous lab.
Label the source vertex s as distance 0.
Label all the other vertices as infinity.
For u = each vertex in the graph, in topological order:
   For v = each neighbor of u:
      See if edge (u, v) can improve the label on v.
Output the distances from s to its reachable vertices.
```

Test your program on an example graph, representing a simple PERT chart.

Longest Path

Finally, management is also interested in the worst case scenario: what is the longest amount of time the project could take to complete? To answer this question, we need to determine the longest path from the beginning to end of the PERT chart. What is the algorithm to finding the longest path in the weighted DAG? Surprisingly, we can use the shortest path algorithm again! All we need to do is negate (multiply by –1) each edge in the graph, and compute the shortest path of the resulting graph.

Write a longest-path routine, which negates the edges and calls the shortest-path algorithm. Test it with the same graph you used in the previous step.