Shortest Path Algorithms for Contests

Saketh Are

December 03, 2010

1 Introduction to Shortest Path Problems

A graph is defined as a collection of vertices connected by a set of edges. We often wish to find the length of the shortest path between two vertices, with each edge having some cost to traverse. In this lecture we will be taking a look at two algorithms that find shortest paths on graph with weighted edges.

2 Floyd-Warshall

2.1 Algorithm

The simplest shortest path algorithm we'll be looking at is the Floyd-Warshall Algorithm. This algorithm finds the shortest path between every pair of vertices in the graph and runs in $O(V^3)$ time, where V is the number of vertices. It works on graphs with positive and/or negative edge weights, assuming no negative cycles.

We begin by creating an matrix of size V by V such that the value at i, j contains the shortest known distance between the vertices i and j. The matrix is initialized using the graph's edges; if there is an edge of cost k between the vertices i and j, we know that there is a path of length k between those two vertices. All pairs i, j without an edge directly from i to j are initialized to some arbitrarily large infinity value.

Once the matrix is set up, we proceed with a dynamic programming approach. The vertices are processed one by one, and as we process them we update the shortest path values in our matrix. It is sufficient to consider at each step only those paths that pass through the vertex being added, since any other paths will have been considered in a previous step. Suppose we are adding in vertex k. We consider all pairs i, j and check whether the best known path going from i to k to j is shorter than the best known path from i to j. If it is, we update the value in our array for the shortest path from i to j.

Since each node is processed in $O(V^2)$ time, the overall runtime is $O(V^3)$.

2.2 Pseudocode

```
int path[][]; //Initialized using edge costs
FloydWarshall()
    for k := 1 to n
        for i := 1 to n
            for j := 1 to n
                path[i][j] = min(path[i][j],path[i][k]+path[k][j]);
```

2.3 Summary

- Finds the shortest path between all pairs of vertices
- By far the quickest and easiest to code shortest path algorithm
- $O(V^3)$ runtime, where V is the number of vertices

3 Dijkstra's Algorithm

3.1 Algorithm

Dijkstra's Algorithm finds the shortest path from a selected root node to all other vertices. It works on all graphs with nonnegative edge weights.

We begin by creating an array that we will use to store the shortest distance for each node. The root node obviously has a distance of 0, while all the other distances are set to some arbitrarily large infinity value. We also create an array that will indicate whether each node has been processed. Initially, mark all the nodes as unvisited.

Start by processing the first node. We will refer to the node currently being processed as the current node.

1. Consider all unvisited neighbors of the current node and compute for each the sum of the current node's source-distance and the cost of the edge from the current node to the neighbor.
2. This sum provides a possible source-distance for the neighbor. If it is better than the source-distance stored for that node, update it with the new value.
3. Once all unvisited neighbors are considered, mark the current node as processed.
4. Select the unprocessed node with the minimum stored source-distance. This will be the new current node.

Once all the nodes have been processed, the source-distance array will contain for each node the length of the shortest path to the chosen root node.

The runtime of the algorithm depends largely on how efficiently we select the next node to process. With a linear search, the runtime is $O(V^2 + E) = O(V^2)$. Using a binary heap, we can reduce this runtime to $O((E + V)logV)$. With a Fibonacci heap, the runtime can be further reduced to $O(E+VlogV)$. However, a binary heap is usually sufficient for USACO contest problems.

3.2 Pseudocode

```
Dijkstra(graph, source):
    for each vertex $v$ in graph:
        dist[v] = infinity;
    dist[source] = 0;
    V = the set of all nodes in graph;
    while V is not empty:
        c = vertex in V with smallest dist[];
        if dist[c]=infinity: break;
        remove c from V;
        for each neighbor b of c:
            tmp = dist[c]+edge(c,b);
            if tmp<dist[b]: dist[b] = tmp;
    return dist[];
```

3.3 Summary

- Finds the shortest path between a selected root node and all other vertices
- $O(V^2)$ runtime if implemented with a linear search
- $O((E + V)logV)$ runtime if implemented with a binary heap
- $O(E + VlogV)$ runtime if implemented with a fibonacci heap

4 Shortest Path Retrieval

As described, both of these algorithms give us the lengths of the shortest paths either between all pairs of vertices or from a chosen source node to all other vertices. Suppose we wish to find the actual sequence of vertices we travel through to acheive the shortest path length. A few simple modifications can be made to our shortest path algorithms to produce this sequence without increasing the runtime complexity of the algorithm.

For Dijkstra's Algorithm, we begin by creating an array that will tell us which neighboring vertex we need to travel from in order to acheive the shortest path to a given node. This information is known as a back-pointer. As we run the algorithm, we simply update the back-pointer every time we update the minimum distance for a node. When we finish we will have back-pointers stored for each vertex, and we can start at an arbitrary node and trace it back to the root by moving down the path one edge at a time. A similar modification can be made for Floyd-Warshall using a 2-dimensional array of back-pointers.

5 Problems

1. Farmer John owns a collection of pastures with weighted edges between some pairs of locations, and a weighted edge from one or more of the pastures to his barn. His cows are distributed throughout the pastures, with each pasture containing at most one cow. At dinner time all of the cows start walking to the barn at the same speed, each taking the shortest possible path. Which cow will reach the barn first?

2. Farmer John owns a collection of pastures with weighted edges between some pairs of locations. Each pasture is inhabited by a cow, and the cows wish to all congregate at one of the pastures. Find the pasture at which the cows should meet in order to minimize combined travel distance.

   Extra: Suppose you know that the given graph is a tree. Can you find the pasture in $O(n)$ time?

3. You are given a set of locations, lengths of roads connecting them, and an ordered list of package dropoff locations. Find the length of the shortest route that visits each of the package dropoff locations in order.

4. Given an undirected, unweighted, connected graph, find the two vertices which are the farthest apart.

5. You are given two locations on an N by N chessboard. Determine the shortest sequence of knight moves from one square to the other.

6. Farmer John owns a collection of pastures with weighted edges between some pairs of locations, and a weighted edge from one or more of the pastures to his barn. Your task is to evaluate a query consisting of two pasture locations and a maximum distance. One of Farmer John's bulls wishes to start at the first pasture, travel to the barn to pick up chocolate, and visit his sweetheart at the other location. However, the bull is only willing to travel a certain distance. Devise an algorithm to efficiently decide for a large number of queries whether the bull can make it.