All-Pairs Shortest Path

11-17-2003

Opening Discussion

- Do you have any questions about the quiz?
- Have you thought more about the single-source shortest path problem and how you might improve on the Bellman-Ford algorithm? How did that algorithm work?
- Do you have any questions about the assignment?

More Single-Source Shortest Path

- In the case of a DAG we can create our shortest path tree even more efficiently, O(V+E). We do this by first doing the topological sort that was discussed last time, then visiting nodes and relaxing them in the order of the topological sort. This way we never visit a given node until we have visited all of it's predecessors in the DAG.

Dijkstra's Algorithm

- Only works when edge weights are non-negative.
- This algorithm picks the next node to do the relaxation on in a more intelligent way. It keeps a min-priority queue of all the vertices that aren't finalized yet. The smallest one can be finalized and all the nodes it has edges to are relaxed. This is repeated until every node is finalized.
- Is easily done in $O(V^2+E)$ time. With a normal heap we get $O((V+E) \log V)$ time. With a Fibonacci heap we get $O(V \log V + E)$ time.

All-Pairs Shortest Path

- What if you want to calculate the shortest path from any source to any destination? We could just run the single-source algorithms V times to get that information, but that would have an order of V times whatever the order of the algorithm that we used.
- As you might guess though, there are more efficient algorithms for solving this.

Data Representation

- In general we will use a matrix representation here, where the matrix element $w_{ij}=0$ if i=j, the weight if an edge connects i to j, and ∞ if there isn't an edge from i to j.
- We also keep a matrix D where each element d_{ij} is the minimum distance path from i to j. Lastly we keep a predecessor matrix Π where π_{ij} is the predecessor of j on the shortest path from i or nil.

Using "Matrix Multiplication"

- Define $l_{ij}^{(m)}$ as the minimum path from i to j with at most m edges. By definition, $l_{ij}^{(0)}$ is 0 if i=j and ∞ if i≠j. We can recursively define it for higher m values as follows.

$$l_{ij}^{(m)} = \min\left(l_{ij}^{(m-1)}, \min_{1\le k\le n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\}\right) = \min_{1\le k\le n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\}$$

- When m=V-1 we know we have the true shortest paths.
- If we start with a matrix W we can evolve $L^{(m)}$ to get our solution.

Improving Performance

- A basic application of this would produce $O(V^4)$ performance. However, the algorithm that extends the shortest path is associative, just like matrix multiplication, so we can compute the answer through repeated squaring.
- Also, because $L^{(m)}=L^{(n-1)}$ for all $m\ge n-1$, we can overshoot in the squaring and it doesn't cause a problem.
- This gives $O(V^3 \log V)$ performance.

Minute Essay

- How are the problems we are talking about different from the TSP? These algorithms are all polynomial, yet you are told the TSP isn't. Why is that?