1 Lecture 7, December 2, 1999

We shall show how fast matrix multiplication can be used so as to speed up all pairs shortest paths computations in graphs.

1.1 Quick review of Dijkstra's algorithm

Let $G(V,E)$ be a directed graph with n vertices and m edges, and let w be a function that assigns nonnegative weights to edges. To find shortest paths from vertex s to all other vertices simultaneously we can use Dijkstra's algorithm that works as follows.

1. Initialize $d(s) = 0$ and $d(v) = \infty$ for all vertices $v \neq s$. Initialize S (the set of vertices to which shortest paths have already been computed) to $S = \phi$, and maintain also a set $Q = V \setminus S$.
2. *while* $Q \neq \phi$, let $v \in Q$ be the vertex with smallest $d(v)$. If $d(v) = \infty$, the algorithm ends (all vertices of Q are unreachable from s).
3. $S \longleftarrow S \bigcup \{v\}$.
4. For each vertex $u \in Q$ such that $(v,u) \in E$, $d(u) = \min[d(u), d(v) + w(v,u)]$. If $d(u)$ was decreased, note v as a tentative parent of u. (Without this parent notion, the algorithm computes shortest distances rather than shortest paths.)

The correctness of Dijkstra's algorithm follows from the fact that after iteration j, the set S contains exactly the j vertices closest to s. This can be proved by induction.

Dijkstra's algorithm explores each edge at most twice, and hence has a running time proportional to m. In addition, it needs to find the minimum in Q for n iterations, which takes $O(n^2)$ time in a naive implementation, $O((n+m)\log n)$ time if Q is maintained and updated as a heap, and $O(n \log n)$ time using more sophisticated data structures (Fibonacci heap) that have a more efficient update operation following the decrease of the value of $d(u)$.

There are also single source shortest path algorithms that handle edges of negative weight, but they are slower. Interestingly, finding the shortest path between two vertices does not appear to be significantly easier than doing single source shortest path.

1.2 All pairs shortest path

Let $X = \{x_{ij}\}$ and $Y = \{y_{ij}\}$ be two order n matrices. Define the *distance product* matrix $M = \{m_{ij}\}$ as

$$m_{ij} = \min_k [x_{ik} + y_{ki}]$$

Distance products are a variant of matrix multiplication over $(\min, +)$ instead of $(+, \cdot)$. As min does not have an inverse, fast matrix multiplication does not appear to apply, and hence we are left with an $O(n^3)$ algorithm for distance products.

Given graph $G(V,E)$ and weight function w on the edges (here we allow negative weights), let $A = \{a_{ij}\}$ be the matrix with $a_{ij} = w((v_i, v_j))$, with $a_{ij} = \infty$ if $(v_i, v_j) \notin E$. Then A is a matrix of all pairs shortest distance (apsd) for G, restricted to paths that contain at most one edge. It can be readily seen that using distance products, A^k is the apsd matrix for G, restricted to paths that contain at most k edges. If G has no negative cycles (directed cycles with negative total edge length), then A^n gives the apsd matrix for G. It can be computed in $O(n^3 \log n)$ time using repeated squaring.

If G does have negative cycles, this can be detected by searching the diagonal entries of A^n for negative entries.

1.3 Seidel's algorithm

In special cases, it is possible to use fast matrix multiplication so as to speed up computation of all pairs shortest paths (see for example the paper of Shoshan and Zwick and references therein). Perhaps the simplest case is that of an undirected graph with unweighted edges.

Let A be the adjacency matrix of such a graph G, with 1 along the diagonal. (As a convention for the following algorithm, each vertex is assumed to be a neighbor of itself.) It is a Boolean matrix that represents pairs of vertices reachabale within distance 1.

If A is the all 1 matrix J, then D, the all pairs shortest distance matrix is simply $A - I$ (where I is the identity matrix).

Consider now the matrix A^2, using Boolean matrix multiplication. It represents pairs of vertices reachable within distance 2. It is also the adjacency matrix of the graph G^2. Now assume that we find the all pairs shortest distance matrix C of G^2. Then the all pairs shortest distance matrix D of G is very similar to $2C$. The difference is that if d_{ij} is odd, then $d_{ij} = 2c_{ij} - 1$, rather than $2c_{ij}$. Hence it remains to check whether d_{ij} should be odd. Observe that the neighbors k of j must have $|d_{ik} - d_{ij}| \leq 1$ (because the graph is undirected and unweighted). Moreover, at least one neighbor k must have $d_{ik} = d_{ij} - 1$. If d_{ij} is odd, then this k has $c_{ik} = c_{ij} - 1$, and all other neighbors k' of j have $c_{ik'} \leq c_{ij}$. If d_{ij} is even, then every neighbor k of j has $c_{ik} \geq c_{ij} - 1$. This leads to Seidel's algorithm, which we call algorithm APSD. It receives as input a Boolean adjacency matrix A, and returns as output an integer all pairs shortest distance matrix D.

1. If $A = J$ then return $J - I$.

2. Else $C \longleftarrow APSD(A^2)$. ($A^2$ is computed using fast Boolean matrix multiplication.)

3. $X \longleftarrow CA$. (X is computed using fast integer matrix multiplication. x_{ij} is the sum of the distances (in G^2) of neighbors of j (in G) from i.)

4. $e_{ij} = 1$ if $x_{ij} < c_{ij} \cdot \text{degree}(j)$, and 0 otherwise. ($e_{ij}$ indicates if the distance from i to j is odd.)

5. $d_{ij} = 2c_{ij} - e_{ij}$. Return D.

As we are using repeated squaring, APSD is called $O(\log n)$ times, and the total running time is $O(n^\omega)$, up to logarithmic multiplicative factors.

1.4 Finding parents

We have seen how to find all pairs shortest distances in undirected unweighted graphs. To actually find paths, we wish to computed the *parent* matrix, that gives in entry ij the vertex immediately preceding j on the shortest path from i. We do this by inspecting the apsd matrix D.

A notion that is helpful is *witnesses* for Boolean matrix multiplication. Let X and Y be two Boolean matrices and M their Boolean product. Then if $m_{ij} = 1$ there is some k for which $x_{ik}y_{kj} = 1$. Such a k is called a witness for m_{ij}.

Suppose we had a method for efficiently computing witnesses for Boolean matrix multiplication. Then we could find parent vertices as follows. Let A be the adjacency matrix of an undirected unweighted graph G, and let D be the APSD matrix of G. Note that k is a parent of j for shortest paths starting at i iff $d_{ik} = d_{ij} - 1$ and k is a neighbor of j. The crucial observation that we shall use is that the above condition holds even if we take all distances modulo 3, because all neighbors k of j have $|d_{ik} - d_{ij}| \leq 1$. Let D_l for $l \in \{0, 1, 2\}$ be the Boolean matrix with entry ij equal to 1 iff $d_{ij} = l$ modulo 3. Then the witnesses for the Boolean product $D_l A$ are precisely the parent vertices for those entries in which $d_{ij} = l + 1$ modulo 3. Hence we need to find winesses for Boolean multiplication for three different products (for the three possible values of l) to find the parent matrix.

It remains to show how to efficiently compute witnesses for Boolean multiplication. This is shown in the next lecture.

References:

- R. Seidel. "On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.,* 51:400–403, 1995.

- A. Soshan, U. Zwick. "All pairs shortest paths in undirected graphs with integer weights". In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, 1999, 605–614.