CS 3510 - Design & Analysis of Algorithms

Co-Instructor : Prasad Tetali, office: Skiles 132, email: tetali@math.gatech.edu

BFS, Dijkstra's algorithm, Min-Heap and Bellman-Ford Algorithms

I. (*Overall summary of Chapter 4.*)

This chapter is all about shortest paths algorithms. We discussed BFS (briefly), Dijkstra's and Bellman-Ford's a bit more in depth. (I will comment on the Floyd-Warshall algorithm at the end and will discuss briefly on Monday, March 12th in class.) The following summarizes their features:

- **Breadth First Search** (BFS): **Task** - given an *unweighted* (directed or undirected graph with equal edge-lengths) $G = (V, E)$, and a start vertex $s \in V$, it finds the length of the shortest path from s to each vertex in the graph.
  Uses a standard queue (FIFO) to identify and process vertices in increasing distance from s – *for each $d = 0, 1, 2, \ldots$, all vertices of distance at most d have their distance set before those at distance larger than d.*
  Time-complexity: $O(|V| + |E|)$: Each vertex is placed on the queue only once; each edge is considered only once (in directed graphs) or twice (in the undirected case).

- **Dijkstra's** algorithm: **Task** - given a weighted (directed or undirected graph with *positive edge-lengths*) $G = (V, E)$, and a start vertex $s \in V$, it finds the length of the shortest path from s to each vertex in the graph.
  Based on the same logic as the BFS, but uses a priority queue to facilitate the operations of inserting and deleting *distance values as keys*. We discussed an array or a binary heap as potential data structures to realize the priority queue.
  Time-complexity: $O(|V|^2)$ **if an array is used, and** $O([|V| + |E|] \log(|V|))$ **if a binary min-heap is used.** *Makequeue* requires $O(|V|)$ *inserts*. *Deletemin* is used $|V|$ times, once for each vertex. and *Decreasekey* is used $|E|$ times.
  i) An *unordered* array of key values can achieve $O(1)$ for insert or decreasekey, but $O(|V|)$ for deletemin (since the array is unsorted).
  ii) A binary heap (with the property that *each node has the key value less than or equal to that of its children*) can achieve $O(\log |V|)$ for each of the operations: insert, decreasekey and deletemin.

- **Bellman-Ford** algorithm: **Task** - given a weighted (directed or undirected graph, with *positive or negative edge-lengths* (but no negative-cycles) $G = (V, E)$, and a start vertex $s \in V$, it finds the length of the shortest path from s to each vertex in the graph.
  Works since any shortest path from s to v (for any t) uses at most $|V| - 1$ edges!
  Time-complexity: $O(|V| \cdot |E|)$ – since the *update* step is done on each edge $|V| - 1$ times.

II. Discussed Dijkstra's algorithm in class using Figures 4.8, 4.9 and 4.11.

III. (*Discussed solutions to the following problems*).

Problem 4.3. (*Squares*). Determining if there is a *Square* is the same as determining if there is a pair of vertices u, v which have two common neighbors w and z. If so, a Square (4-cycle) is formed using u, w, v, z (and u). Let us assume that the graph is given using an adjacency matrix A.

i) $O(|V|^3)$ solution: For a fixed pair, u, v, we can determine in $O(|V|)$ time, if they have more than one common neighbor by comparing the two rows of A corresponding to u and v. To repeat this for all pairs of vertices requires that we perform it $O(|V|^2)$ times, giving a total time complexity of $O(|V|^3)$.

ii) $O(|V|^{2.71})$ or so solution (*Good Will Hunting* anecdote ...): Observe that the kth power of A can be used to find the number of paths of length k between pairs of vertices. In particular, the uv-entry of A^2 equals $\sum_w A(u,w)A(w,v)$, which is the the number of paths of length two between u and v, also equal to the number of common neighbors of u and v. Hence we can compute A^2 in matrix multiplication complexity of time and check if any of the entries of A^2 is bigger than 1 or not.

Problem 4.11. (*Shortest cycle*). Let us assume the graph has a cycle, since it can be detected easily using a (directed) DFS whether there is a cycle or not. For any pair of vertices u, v there is a cycle of length $D_{uv} + D_{vu}$, consisting of the two shortest paths between u and v, and crucially, this cycle is the shortest among cycles containing u and v. So we construct a matrix D, whose uv-entry is the length of the shortest path between u and v, and compute $\min_{u,v}[D(u,v) + D(v,u)]$ to determine the length of the shortest cycle in the graph.

$O(|V|^3)$ solution: in $O(|V|^2)$ time, using Dijkstra's with the unordered array implementation of a priority queue, we can find each row of the matrix D above. So in $O(|V|^3)$ time, we compute the matrix D. Then the min above can be performed in an additional $O(|V|^2)$ time. (If we use a binary heap, we get a worse performance, as discussed in class.)

Alternatively, we may use the Floyd-Warshall algorithm (see Pages 172-173 in the Dynamic Programming chapter of the textbook) that finds *all-pairs shortest paths* in $O(|V|^3)$ time, and compute the matrix D.

Problem 4.21. (*Currency Exchange Scheme/Scam*). The basic idea is to recognize that this is a *maximization* problem over paths in a directed graph, with the pathlength defined using the *product* over (the lengths of) the edges in the path. Once we recognize that we can solve it by i) converting the lengths using logarithms (so the product problem becomes additive) and ii) taking the negative of the logs, so the maximization becomes minimization, and iii) finally by using the Bellman-Ford algorithm to find shortest paths (in the presence of negative-length edges).

To elaborate: construct a *complete* directed graph with currencies being the vertices, and weight $w_{ij} := -\log(r_{ij})$ being the length on each edge i, j, where r_{ij} is the exchange rate from currency i to currency j.

a) Use Bellman-Ford to find the shortest path in the above graph from s to t.

b) Iterate the updating procedure one more time after $|V||E|$ rounds in the (Bellman-Ford) algorithm, and if any distance is updated, then it means there is a negative cycle. This implies that the corresponding sequence of exchange rates products to greater than 1.

Note: I will discuss Problems 4.12, 4.19 and a couple of others on Monday, March 12th.