Automata Theory (CSE4058-01)
Homework # 03 – Solution

1.9 Let G be your graph. Construct a new graph G' by doubling each edge in G. All vertices in G' have even degree. According to Euler, G' has an Eulerian cycle which crosses every edge in G' exactly once. In G, this cycle crosses each edge exactly twice. Now consider the Chinese postman problem for a tree graph T. Since the postman travels a cycle, each vertex has at least one arrival and one departure edge. Let e be a departure edge for vertex v. If we follow the postman tour along e, we must eventually come back to v, but the corresponding arrival edge also has to be e since otherwise we would have discovered a loop in T. Hence e is crossed twice, once for arrival and once for departure, and this holds for any edge in T.

The "Eulerification" of a graph by doubling edges is also the key to an algorithm for the Chinese postman problem. Instead of doubling every edge, we focus on those vertices that have odd degree in G. Make a list of all pairs of vertices of odd degree. For each pair determine the shortest path in G that connects them. This can be done in polynomial time, see Section 3.4. Now construct a complete graph F whose vertices are the odd vertices of G. The edges in F are weighted with the length of the shortest paths that separates the vertices in G. Then find the minimum weight matching in F. This can be done in polynomial time, see Note 5.6. Finally add edges in G along each path given by the matching. The resulting graph is Eulerian, and the Euler cycle is the solution of the Chinese postman problem.

1.10 There are many ways to solve this problem. Here's one. Suppose the original graph G has n vertices, and that the oracle says "yes" to the question of whether G is Hamiltonian. Go through the edges in any order. For each edge e, ask whether G would still be Hamiltonian if we removed e. If she says "yes,"

remove e from the graph. If she says "no", mark that edge as essential, and never ask about it again. Continue in this way until there are only n edges left—all of which are essential—which form a Hamiltonian cycle for G.

Since each edge is either removed are marked as essential when we ask about it, we ask about each edge at most once. Thus the total number of questions we ask (or drachmas we spend) is at most the number of edges, which in a simple graph is $O(n^2)$.

2.1 If she can upgrade once every two years, Brilliant Pebble should wait for the next upgrade if T is greater than four years, since this will let her graduate in $2 + T/2 < T$ years instead.

On the other hand, if we assume that processing speed is constantly improving and she can upgrade at any time, then waiting t years lets her graduate after a total time of $t + 2^{-t/2}T$. If $T > 2/\ln 2 \approx 2.89$, she can minimize the total time by starting her program at $t = 2(\log_2(T \ln 2) - 1)$.

2.18 If there are n vertices, the adjacency matrix takes n^2 bits. If there are m edges, each one takes $O(\log n)$ bits to describe its endpoints, for a total of $O(m \log n)$ bits. In a sparse graph where $m = O(n)$, the list of edges is $O(n \log n)$ bits long, much shorter than the adjacency matrix. In a dense graph with $m = \Theta(n^2)$, the list of edges is $\Theta(n^2 \log n)$ bits long, which is longer than the adjacency matrix by the factor $\log n$.

In a multigraph, the adjacency matrix can no longer consist just of 0s and 1s. Instead, A_{ij} should be the number of edges connecting i to j. If there are m edges total then A_{ij} is $O(\log m)$ bits long, so A's total length is $O(n^2 \log m)$. The list of edges is again $O(m \log n)$ if we naively list each copy of an edge separately.

2.20 If $f(n)$ is quasipolynomial with $k > 1$ and $g(n)$ is polynomial, there are many ways to show that $f(n) = \omega(g(n))$. One way is to consider the ratio of their logarithms:

$$\log \frac{f(n)}{g(n)} = \Theta(\log^k n) - O(\log n).$$

Since this tends to ∞ as n does, $f(n)/g(n)$ tends to ∞ as well, and $f(n) = \omega(g(n))$.

Similarly, if $h(n) = 2^{n^c}$ for $c > 0$, then

$$\log \frac{f(n)}{h(n)} = \Theta(\log^k n) - \Theta(n^c)$$

tends to $-\infty$, so $f(n)/h(n)$ tends to 0 and $f(n) = o(h(n))$.

Finally, we show that the composition of two quasipolynomials is a quasipolynomial. If $f(n) = 2^{\Theta(\log^k n)}$ and $g(n) = 2^{\Theta(\log^\ell n)}$,

$$f(g(n)) = 2^{\Theta(\log^k g(n))} = 2^{\Theta(\log^{k\ell} n)}.$$