CSC 363, Winter 2010 — Short Assignment #3 Solution

1) We can reduce NOADJ-SQUARE to VERTEX-COVER as follows: On input $\langle S, k\rangle$ output a graph, G, with a node corresponding to every square, and an edge between every pairs of nodes that correspond to adjacent white squares. Then output the same k that is in the input.

   This is a valid reduction because, first, if there is a way of changing k white squares in S to grey that eliminates any pair of adjacent white squares, then the set of the k corresponding nodes in G will be a vertex cover. Conversely, if there is a k node vertex cover for G, then changing the corresponding squares in S to grey will eliminate any adjacent white squares.

   To show that this is a polynomial time reduction, we need to specify how the reduction is done in more detail. Let's use a multitape Turing Machine (using a one-tape Turing Machine would change the running time, but it would still be polynomial). We can do the reduction as follows. I'll omit error checks (input not of the form $\langle S, k\rangle$), that are needed to make the reduction precise (they would just check for errors, and if one was found, output something not of the form $\langle G, k\rangle$).

   (a) Scan the "+" and "@" symbols in the input, copying them to another tape. Keep a count of the number, m, of such symbols on another tape (in binary notation).

   (b) Copy the number k from the input to another tape.

   (c) Clear the input tape to blanks, then write the number m at the beginning. This will be the number of nodes in G.

   (d) Find the square root of m, and keep it on a tape. This is the length, ℓ, of a side of the square.

   (e) Scan the copy of the "+" and "@" symbols made in stage (a), incrementing a counter (stored on some tape) so we know the index of the symbol we're looking at. For each "+" symbol, look at the symbol before, the symbol after, the symbol ℓ before, and the symbol ℓ after. (Looking ℓ before and ℓ after will require using a counter on another tape to get there and back.) After each of these looks at adjacent symbols, write an edge to the output tape if the symbols are both "+".

   (f) Output the value of k saved in stage (b).

   Stage (a) takes $O(n \log n)$ time, where n is the length of the input, since it looks at $O(n)$ symbols, for each of which it increments a counter, which involves operations on $O(\log n)$ bits. Stage (b) takes $O(\log n)$ time, stage (c) takes $O(n)$ time, stage (d) takes $O((\log n)^3)$ time (a generous allowance for finding the square root by binary search), and stage (f) takes $O(\log n)$ time. Stage (e) is the most complicated. It takes $O(\ell \log n)$ time for each of $O(n)$ symbols, which gives a total of $O(n^{3/2} \log n)$ time. This dominates the other times, so the time complexity of the reduction is $O(n^{3/2} \log n)$.

2) To decide NOADJ-SQUARE, we feed its input into the reduction procedure above, then take the output of that reduction and feed it the the $O(n^9)$ procedure for deciding VERTEX-COVER that we assume exists. The total time is the sum of the time for the reduction and the time for deciding the VERTEX-COVER problem. Note, however, that n is the size of the input string, which is **not** the same for VERTEX-COVER as for NOADJ-SQUARE. The reduction procedure will output a graph with $O(n)$ nodes and $O(n)$ edges, where n is the size of the input to NOADJ-SQUARE, but the string representing this graph will use node numbers of length $O(\log n)$, so the best bound we have on the size of the output is $O(n \log n)$. The total time to decide NOADJ-SQUARE using this reduction is therefore $O(n^{3/2} \log n) + O((n \log n)^9) = O(n^9 (\log n)^9)$.

3) No. As we've seen, a polynomial time algorithm for VERTEX-COVER would imply a polynomial time algorithm for NOADJ-SQUARE because of the reduction in part (1), but this reduction doesn't imply the converse of that.

4) A procedure that produces the same reduction as in part (1) but with this sparse encoding may take time that grows exponentially with the size of the input. In particular, if the square is all white, the input will just be the binary encoding of the size of the square, but the output will be a list of edges that is proportional to the square of the size of the square, which is exponential in the number of binary digits needed to encode the size. Since the output is exponential in n, the time must be at least exponential in n.