1. **(40 marks)** Implement (in C, C++, or Java) a procedure `minNumMistakes(w1, w2)` that computes the minimum number of simple spelling mistakes required to transform word `w1` into word `w2`. There are four kinds of simple spelling mistakes:

(a) Change a symbol to another symbol, e.g., `bat` → `cat`.
(b) Insert a symbol, e.g., `at` → `bat`.
(c) Delete a symbol, e.g., `bat` → `at`.
(d) Swap two adjacent symbols, e.g., `bat` → `abt`.

Write a main program that uses this procedure to implement a basic spell-checker – that is, given a word `w` and a list `L` of words, the program prints out all words in `L` that are within the minimum number of mistakes of `w`. Run your main program on the test cases described in file `test_a4_1.txt` available off the Assignments home page. Hand in your code and copies of your output on these test cases.

2. **(25 marks)** Consider an edge-labelled directed graph `G = (V,E,σ)` in which each edge `(u,v) ∈ E` is labeled with a symbol `σ(u,v)` drawn from a finite alphabet `Σ` and there may be multiple edges labelled with any `σ ∈ Σ` adjacent to a vertex `u` in `G`. If `Σ` is interpreted as a set of sounds, this labelled graph is a formal model of a person speaking a restricted language. In this graph, each path starting from a distinguished vertex `v0 ∈ V` corresponds to a possible sequence of sounds produced by the model. Define the label of such a directed path as the concatenation of the labels on the edges of that path.

   a) **(15 marks)** Describe an efficient algorithm that, given an edge-labeled graph `G` with distinguished vertex `v0` and a sequence `s = ⟨σ1,σ2,…,σk⟩` of characters from `Σ`, returns a path in `G` that begins at `v0` and has `s` as its label, if any such path exists. Otherwise, the algorithm should return `NO-SUCH-PATH`. Analyze the running time of your algorithm.

   Now, suppose that every edge `(u,v) ∈ E` has also been given an associated nonnegative probability `p(u,v)` of traversing the edge `(u,v)` from vertex `u` and producing the corresponding sound. The sum of the probabilities of the edges leaving any vertex is 1. The probability of a path is defined to be the product of the probabilities on its edges. We can view the probability of a path beginning at `v0` as the probability that a “random walk” beginning at `v0` will follow the specified path, where the choice of which edge to take at a vertex `u` is made probabilistically according to the probabilities of the available edges leaving `u`.

   b) **(10 marks)** Extend your answer to part (a) so that if a path is returned, it is a *most probable path* starting at `v0` and having label `u`. Analyze the running time of your algorithm.
3. **(15 marks)** Consider the following directed graph:

![Graph Image]

Assume that the algorithms below consider vertices in alphabetical order and that each adjacency list is ordered alphabetically.

(a) **(5 marks)** Show how breadth-first search works on this graph. Give the graph at the end of the execution of the BFS algorithm on page 532 of the textbook when the search is started at vertex Q, with the $d$- and $\pi$-values for all vertices as well as all BFS-search tree edges clearly marked.

(b) **(5 marks)** Redo part (a) above with the BFS algorithm starting at vertex T.

(c) **(5 marks)** Show how depth-first search works on this graph. Give the graph at the end of the execution of the DFS algorithm on page 541 of the textbook, with the $d$-, $f$-, and $\pi$-values of all vertices as well as the types of all edges clearly marked.

4. **(10 marks)** Give an algorithm that determines whether or not a given undirected graph $G = (V, E)$ contains a cycle. Your algorithm should run in $O(|V|)$ time, independent of $|E|$.

5. **(10 marks)**

(a) **(4 marks)** We know that the presence of negative-weight cycles invalidates the use of shortest-path algorithms. Does the presence of zero-weight cycles, i.e., cycles whose edge-weights sum to zero, cause similar problems? If so, why? If not, why not?

(b) **(6 marks)** Given a weighted directed graph $G = (V, E)$ with no negative-weight cycles, let $m$ be the maximum over all pairs of vertices $u, v \in V$ of the minimum number of edges in a shortest path from $u$ to $v$ in $G$. (Here, the shortest path is by weight, not the number of edges.) Suggest a simple change to the Bellman-Ford algorithm that allows it to terminate in $m + 1$ passes, assuming we don’t know what the the value of $m$ is for the given input graph.