EE365 Homework 4

1. *The Bellman-Ford algorithm.* Consider a directed, weighted graph with vertex set $\{1, \ldots, n\}$. We can represent such a graph by a matrix $W \in \mathbb{R}^{n \times n}$, where W_{ij} is the weight of the edge (i, j) if the graph contains the edge (i, j), and $W_{ij} = \infty$ if the graph does not contain the edge (i, j). Suppose we want to compute a minimum-weight path from vertex s to vertex t.

   (a) Write a `MATLAB` function that implements the Bellman-Ford algorithm from the lecture slides to solve this shortest path problem. Use the following function header.

   ```
   function [p,wp] = bellman_ford(W,s,t)
   ```

   Here, `W` is the matrix of weights that describes the directed graph, `s` is the given start vertex, `t` is the given destination vertex, `p` is a sequence of vertices that form a minimum-weight path from vertex s to vertex t, and `wp` is the minimum weight of such a path. You may assume that there is a finite-weight path from vertex s to vertex t, and that every cycle in the graph has strictly positive weight.

   (b) The file `shortest_path_data.m` contains a cell array `W` whose entries are matrices describing directed graphs, and vectors `s` and `t` whose entries are vertex labels. Use your `bellman_ford` function to solve each of these shortest-path problems; attach a printout of the minimum weights returned by your function.

2. *The forward Bellman-Ford algorithm.* The version of the Bellman-Ford algorithm covered in the lecture slides solves the minimum-weight-path problem by finding a minimum-weight path *to* the destination vertex t *from* each of the other vertices in the graph. It is also possible to solve this problem by computing a minimum-weight path *from* the source vertex s *to* each of the other vertices in the graph; this is the forward Bellman-Ford algorithm. (The lecture slides describe the reverse Bellman-Ford algorithm.)

   (a) Give a precise description of the forward Bellman-Ford algorithm. For reference, we consider the top part of slide 6 of the Bellman-Ford lecture slides to be a precise description of the reverse Bellman-Ford algorithm. You may assume that there is a finite-weight path from s to t, and that every cycle in the graph has strictly positive weight.

   (b) Write a `MATLAB` function that implements the forward Bellman-Ford algorithm. Use the following function header.

   ```
   function [p,wp] = forward_bellman_ford(W,s,t)
   ```

   (c) Apply your function `forward_bellman_ford` to each of the example graphs defined in `shortest_path_data.m`. Attach a printout of the minimum weights returned by your function.

3. *The weighted scheduling problem.* You are given a set of n jobs, labeled $1, \ldots, n$. Job i has an associated start time $a_i \in \mathbb{R}$, end time $b_i > a_i$, and weight w_i. The jobs you are given may overlap, but you only have the resources to execute one job at a time. Your task is to choose a subset S of the jobs in order to maximize the total weight

$$W = \sum_{i \in S} w_i,$$

subject to the constraint that no two jobs in S overlap: *i.e.*, $(a_i, b_i) \cap (a_j, b_j) = \emptyset$ for $i, j \in S$, $i \neq j$. (In particular, note that if job j starts at exactly the same time that job i ends, then we are allowed to schedule both i and j.)

   (a) Formulate the weighted scheduling problem as a minimum-weight-path problem on a directed graph.

   Hint. In one possible formulation, the vertex set of the graph is $0, \ldots, n+1$, where 0 is a special "start" vertex, $n+1$ is a special "end" vertex, and vertices $1, \ldots, n$ represent jobs. You must define the edges and weights of the graph.

   (b) The file `job_scheduling_data.m` defines an instance of the weighted scheduling problem. Use your function `bellman_ford` to solve the problem. Report the a maximum-weight job schedule, and the corresponding maximum weight.

4. *The knapsack problem.* Suppose you have n items, labeled $1, \ldots, n$. Item i has an associated value v_i, and weight w_i. You have a knapsack with weight capacity W. You want to choose a subset of the items to load into the knapsack whose total weight is at most W, and whose total value is as large as possible. You may assume that the values v_i, the weights w_i and the weight capacity W are all positive integers.

   (a) Formulate the knapsack problem as a minimum-weight-path problem on a directed graph.

   Hint. In one possible formulation, the vertex set of the graph is

$$(\{1, \ldots, n+1\} \times \{0, \ldots, W\}) \cup \{z\},$$

   where

   - z is a special "end" vertex,
   - each $(n+1, w)$ is a special pre-"end" vertex,
   - and, for $i = 1, \ldots, n$, the vertex (i, w) represents an item and a weight.

   We have intentionally been somewhat vague about the meanings of these vertices. You need to figure out exactly what each of these vertices represents, and define the edges and weights of the graph.

   (b) The file `knapsack_data.m` defines an instance of the knapsack problem. Use your function `bellman_ford` to solve the problem. Report the an optimal subset of the items, as well as the weight and value of your optimal subset.

   Hint. The functions `sub2ind` and `ind2sub` are often useful for going between index pairs (i, j) and single indices k.