EE365 Homework 8

1. *Dijkstra's Algorithm.* In this problem, you will write an implementation of Dijkstra's algorithm, and use it to find the shortest path in a social network.

   Consider a weighted, directed graph with vertex set $V = \{1, \ldots, n\}$, and edge set E. We can describe such a graph using 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 $W_{ij} = 0$ otherwise (we assume that there are no zero-weight edges).

   (a) A key step in Dijkstra's algorithm is computing the neighbors of a vertex x:

   $$\mathcal{N}(x) = \{y \mid (x, y) \in E\}.$$

   Implement the following MATLAB function.

```
[Nx , WNx] = get_neighbors(x,W)
% inputs
%   x : a vertex
%   W : the matrix of edges weights (0 if there is no edge)
% outputs
%   Nx : cell array containing the neighbors of x
%   WNx : vector containing the weights
%         of edges between x and its neighbors
```

   Report the output of `get_neighbors(246,W)` for the matrix `W` defined in `facebook_data.m`. The function `get_neighbors` is called a "successor oracle."

   (b) Implement the version of Dijkstra's algorithm given on page 23 of the "Shortest Paths" lecture. Use the following function header.

```
[dist , path] = dijkstra(neighbors , s , t)
% inputs
%   neighbors : function handle [Nx,WNx] = neighbors(x)
%               where Nx is a cell array of the neighbors of x
%               and WNx is a vector of weights
%   s : the source vertex
%   t : the destination vertex
% outputs
%   dist : the distance from s to t
%   path : cell array containing a shortest path from s to t
%          path{i} is the i-th vertex of the path
```

   (c) The file `facebook_data.m` define a matrix `W` that describes part of the Facebook social network. Report the shortest path between nodes $s = 800$ and $t = 3000$. Choose five random source/destination pairs, and compute the distance from each source to the corresponding destination. Report the maximum distance among the five pairs.

Implementation hints.

- You need to figure out how to store the sets `F` and `E`, and the value function `v`. Since `MATLAB` does not offer data structures such as priority queues and hash-tables, you need to use arrays. This is not the most efficient approach, and is not something you would do in most programming languages. However, it is the best solution we have found in `MATLAB`.
- In order to recover the optimal path, you need to keep track of the index `P(i)` that was last used to update the distance `v(i)`. You can use `P(i)` (which is called the predecessor of i), to trace the optimal path backwards from the target to the source. However, note that your algorithm should output the vertices in a path from `s` to `t`.
- Note that the `get_neighbors` function you wrote in (a) is slightly different from the `neighbors` function that you must pass to `dijkstra`. You can address this discrepancy using an anonymous function. If you have a function $f(x, y)$, you can define a function $g(x) = f(x, y_0)$ for some fixed value of y_0 using the expression `g = @(x) f(x,y0)`. The right side of this expression is called an anonymous function because it is not defined in a named function file.
- When searching for the state in F with minimum distance, break ties by always selecting the state that was inserted last in the frontier. This will reduce the number of states extracted from F before a solution is found.

2. *Navigating a maze.* In this problem, you will reproduce and extend the maze example from the lecture slides. You are given a two-dimensional maze (Figure 1) as a matrix $A \in \{0, 1\}^{81\times 81}$, where $A_{ij} = 1$ means that the position (i, j) is free, and $A_{ij} = 0$ that it is blocked. You are allowed to move horizontally or vertically between adjacent free positions.

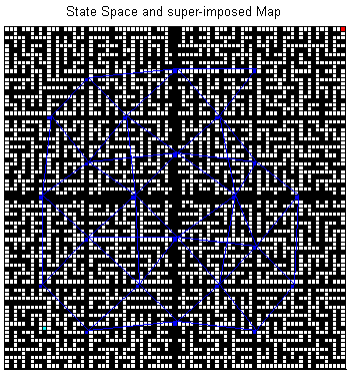

Figure 1: The maze along with the way-points graph.

The cost of moving horizontally is p, and the cost of moving vertically is q. Furthermore, this maze has a special structure in that it is partitioned into smaller mazes by vertical and horizontal lines in positions `X=[20 41 60]`, and `Y=[20 41 60]` respectively. One

can move between smaller mazes only through some way-points, that are marked in blue in Figure 1. The data are available in the `maze.dat` file. You will use Dijkstra's algorithm to investigate the impact of different heuristics for finding shortest paths in this maze.

(a) Implement the function `maze_succ` that plays the role of the successor oracle for the maze problem. Use the prototype `[Nbors,costs]=maze_succ(x,A,h,p,q)`, where

- `A` is an 81×81 matrix indicating whether locations are free or blocked
- `x` is a vector of length 2 given the current position in the maze,
- `p` and `q` are the costs of moving horizontally and vertically, respectively
- `h` is a function handle that defines a heuristic.

Recall that the heuristic function affects the costs through the formula $\hat{g}_{ij} = g_{ij} + h_j - h_i$.

(b) *Uninformed Search.* Find the shortest path between $s = (1, 19)$ and $t = (81, 61)$. Report the total cost of the path, the number of states extracted from the frontier F, and the 100-th state along the optimal path, i.e., `path{100}`, where `path{1}` is considered to be the source.

(c) *Manhattan Heuristic.* Define the "Manhattan heuristic" for this problem, and implement it in the function `manh(x,t,p,q)`. Show that it is a *consistent heuristic.* Report `manh(s,t,p,q)`.

(d) *A* with Manhattan heuristic.* Repeat (b) using the function `manh`.

(e) *Finding Close Way-points.* Implement the following helper function, which will be useful later in the problem.

```
wpind = waypts(x,X,Y,W)
% inputs
%   x : the current location
%   X : locations of the horizontal lines
%   Y : locations of the vertical lines
%   W : a matrix with two columns;
%       each row is the location of a waypoint
% outputs
%   wpind : the indices of waypoints that can be accessed from x
%           without going through any other waypoints
```

For states that are way-points. your function should simply return the index of the way-point. Report `waypts(s,X,Y,W)`.

(f) *Waypoint Distance heuristic.* Implement the following helper function.

```
hx = heurMap(x,t,X,Y,W,D)
% inputs
%   x : the current state
%   t : the target state
```

```
%   X : the locations of the horizontal lines
%   Y : the locations of the vertical lines
%   W : the locations of the waypoints
%   D : a matrix of estimates of the distances between way-points
% outputs
%   hx : an estimate of the distance from x to t (see below)
```

The function should output the following number:

$$h_D(x) = \min \left\{ \begin{matrix} \texttt{manh(x, W(i,:))} + D(i,j) + \texttt{manh(W(j,:), t)}, \\ \text{for } i \in \texttt{waypts(x)}, j \in \texttt{waypts(t)} \end{matrix} \right\}$$

when `waypts(x)`$\neq$`waypts(t)` and

$$h_D(x) = \texttt{manh(x,t)}$$

when the sets of way-points of the states `x,t` are identical (`x` is near the goal).

(g) *Way-point Manhattan Distance.* There is a graph associated with the way-points (also shown in blue in Figure 1). The data file defines the matrix `W`, which gives the locations of the way-points, and the adjacency matrix `G` of the graph associated with the way-points. Use this information to compute the weighted adjacency matrix `Gw` of the way-points graph, where we take the Manhattan distance between two way-points to be the weight of an edge. We have provided the following function for your convenience.

```
D = allPairsSP(A)
% inputs
%   A : weighted adjacency matrix
% outputs
%   D : matrix of distances between vertices of A
```

Use this function to compute `Dmanh = allPairsSP(Gw)`.

(h) *A* with way-point Manhattan heuristic.* Repeat (b) using the heuristic `heurMap` with the matrix `Dmanh` of estimated distances.

(i) *A* with actual way-point Distance.* Construct the matrix of actual distances between waypoints `Dtrue` by running `dijkstra` for each adjacent pair of way-points. Report `Dtrue` and repeat steps (g) and (h) using `Dtrue`.

Remark. The function `showPath.m` can be used to plot a path given in a cell array.

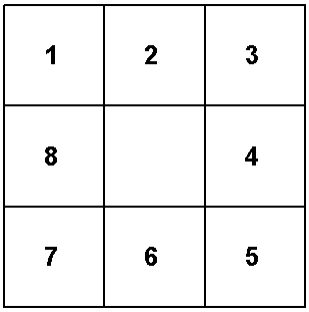

Figure 2: Goal state of the 8-Tile puzzle.

3. *Convoy on a graph.* You are a rich merchant, and you want to move some valuable cargo from city s to city t. Unfortunately, the road is full of bands of thieves aiming to steal your cargo. Your network of informants has given you an accurate map of the routes between different cities, and the number of bandits along each road. You can hire mercenaries to guard you on your journey You know that if the number of bandits along any road in your itinerary is at most half the number of mercenaries guarding your cargo, then you will not be attacked. Once you hire the mercenaries, you also need to plan your route. Each mercenary you hire costs 10 gold coins, and you want to find the minimum number of coins you need to spend in order to safely transport your cargo.

   (a) Explain how to modify Dijkstra's algorithm to solve this problem.

   (b) The file `convoy.mat` defines an instance of this problem. Report the minimum number of coins that you need to spend to hire mercenaries, and a corresponding path from $s = 1$ to $t = 23$.

4. *The 8-tile puzzle.* The 8-tile puzzle is a sliding puzzle in which 8 tiles, labeled $1, \ldots, 8$, are placed in a 3×3 box. Note that there is always one empty space in the puzzle. The rules are that you can change the the configuration of the puzzle by sliding any tile adjacent to the empty space into the empty space. The goal is to reach the configuration shown in Figure 2. In this problem, you will solve the 8-tile puzzle using Dijkstra algorithm and heuristics.

   (a) We can think of the 8-tile puzzle in terms of a graph. Let each configuration of the puzzle be a vertex, and let there be an edge between two vertices if the corresponding configurations can be reached from one another in one move. We assume that each edge has weight one. Given a starting configuration s, we can we can solve the 8-tile puzzle by finding a shortest path from s to the configuration t shown in Figure 2. Give a *consistent heuristic* `h` for this problem, and prove that your heuristic is consistent.

   (b) We can represent a configuration of the puzzle using a vector of length 9; for example, the goal configuration is $x = (1, 8, 7, 2, 0, 6, 3, 4, 5)$. Implement the following function.

```
[Nx , WNx] = tile8_neighbors(x,h)
% inputs
%   x : a configuration of the 8-tile puzzle
%   h : a heuristic
% outputs
%   Nx : the neighbors of x (that is, the configurations
%        that can be reached from x in one move)
%   WNx : the cost of moving to each neighbor using the heuristic
```

(c) Solve the 8-tile puzzle from the starting state $s = (7, 8, 1, 5, 4, 0, 3, 2, 6)$. Report your sequence of moves, as well as the number of states that you extracted from the frontier. Use can use the function `solution8tile(path)` to animate your solution of the puzzle. This function assumes that `path` is a cell array, where `path{i}` is the ith state in the path.