# Written Assignment 1

Due: Friday 02/02/2024 @ 11:59pm EST

## Disclaimer

I encourage you to work together, I am a firm believer that we are at our best (and learn better) when we communicate with our peers. Perspective is incredibly important when it comes to solving problems, and sometimes it takes talking to other humans (or rubber ducks in the case of programmers) to gain a perspective we normally would not be able to achieve on our own. The only thing I ask is that you report who you work with: this is **not** to punish anyone, but instead will help me figure out what topics I need to spend extra time on/who to help. When you turn in your solution (please use some form of typesetting: do **NOT** turn in handwritten solutions), please note who you worked with.

Remember that if you have a partner, you and your partner should submit only **one** submission on gradescope.

### Question 1: Shortest Path Composition (25 points)

Consider a graph $G = (V, E, w : E \to \mathbb{R}^{\geq 0})$ where $V$ is a set of vertices, $E$ is a set of (directed) edges, and $w$ is a *weight function* that maps edges to weights where each weight is $\geq 0$. Let us define a path $p$ to be a sequence of edges where the destination vertex of one edge is the source vertex of the next edge (if the next edge exists). Let us define the *cost* of an path the traditional way, i.e. the cost of a path $p$ is the sum of the edge weights in $p$:

$$cost(p) = \sum_{e \in p} w(e)$$

Show that if we know a shortest path $p^* = a \overset{x}{\rightsquigarrow} b$ from vertex $a$ to vertex $b$ has cost $x$. If we know $p^*$ passes through intermediary vertex $c$, then let $p_1 = a \overset{y}{\rightsquigarrow} c$, and let $p_2 = c \overset{z}{\rightsquigarrow} b$. Show that if $p^* = p_1 \cup p_2$, then $p_1$ is a shortest path from $a$ to $c$, and that $p_2$ is a shortest path from $c$ to $b$.

---

If there was an edge that was in both $p_1$ and $p_2$, it would not be needed in $p_2$ because it would be going to a vertex that the path had already advanced to. (This wouldn't apply with negative weights, because cycling would give lesser costs, but we know the weights are nonnegative.) Therefore, $p_1 \cap p_2 = \emptyset$.

Now, since we also know $p^* = p_1 \cup p_2$ and they're disjoint, each path in $p^*$ is in either $p_1$ or $p_2$. Thus we may say $\sum_{e \in p^*} w(e) = \sum_{e_1 \in p_1} w(e_1) + \sum_{e_2 \in p_2} w(e_2)$. Looking back to our definitions this equation is the same as $x = y + z$.

For purposes of contradiction assume that $p_1$ is not the shortest path from $a$ to $c$. We can call the new cost of this path $y - k$, with $k > 0$. Then, revisiting our equation from before, if we use this new path, we have $x - k = y - k + z$ as the total cost. But then we've shown that it's possible to have a lower cost, so $p^*$ is not the shortest path.

In either case, we reach an impossibility, so $p_1$ must be the shortest path from $a$ to $c$. We can use the exact same argument with $p_2$.

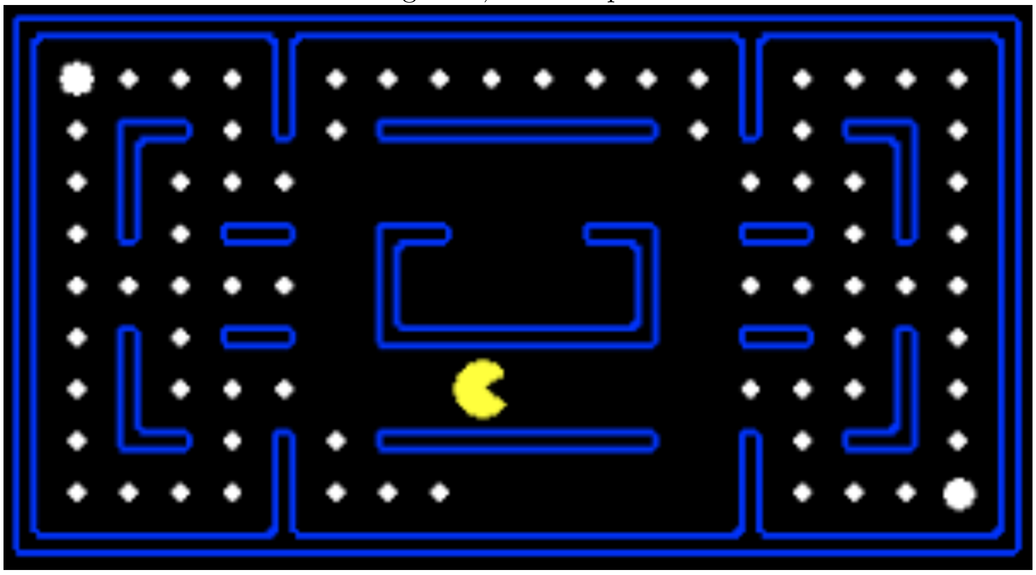
## Question 2: Best and Worst Cases for DFS (25 points)

Consider a graph $G = (V, E, w : E \to \mathbb{R}^{\geq 0})$ where $V$ is a set of vertices, $E$ is a set of (directed) edges, and $w$ is a *weight function* that maps edges to weights where each weight is $\geq 0$. Let us start a DFS search from vertex $a$, the goal of which is to find vertex $b$ (where $b \neq a$). In the worst case, where is the goal vertex $b$ in relation to the source vertex $a$ in the DFS expansion. How many vertices and edges must the expansion contain before when we reach $b$? What about the best case?

1. In the worst case the goal vertex would be at the very end of the DFS expansion. This could happen if the graph was a long chain with $a$ on one side and $b$ on the other.
2. The expansion would have to have all $|V|$ vertices and all $|E|$ edges. The worst case would be a graph that looks like a long line, thus, there is a path from one end to another that has the cost of the sum of all edge-weights in the graph. If a certain edge $e$ were not included in the expansion, we would be subtracting the weight $w(e)$ from the total, which is guaranteed to be nonnegative, so this new path's cost would be less than or equal to our worst-case cost.
3. The best case would occur if $a$ and $b$ were adjacent vertices with a directed edge from $a$ to $b$, and $a$ had only one outgoing edge or its outgoing edge to $b$ was the first one that we chose when conducting the search. That is, we'd have 2 vertices and 1 edge in our expansion. Note this is the best case as defined by the number of vertices and edges, not necessarily the total cost.

**Extra Credit: Heuristics for Pacman-NoGhosts (50 points)**

Consider a Pacman world with no ghosts, an example of which is shown below:



Here is the description of this search problem:

- **Environment:** A 2-d map of finite size where each square can contain a wall, a food pellet, Pacman, or is unoccupied.
- **Sensors/State:** The state of the world is a structure containing the following fields (in Java-ish syntax):
  - `current_loc: Coordinate`: The (x,y) location of Pacman in the map.
  - `food_remaining_locs: Collection<Coordinate>`: The (x,y) locations of all remaining food pellets in the map.
- **Initial state:** The initial state is an instance of the State with the following field values:
  - `current_loc = (9, 3); // technically any unoccupied square will do`
  - `food_remaining_locs = locations of all food pellets in the map;`
- **Actuators/Actions:** Pacman can move to an adjacent square in a cardinal direction.
- **Transition Model:** Pacman will move to the adjacent square if it is not a wall and if the action will not take Pacman out of bounds of the map. Additionally, if Pacman enters a square that contains a food pellet, Pacman will automatically consume that pellet. When Pacman consumes a pellet, the location of that pellet is removed from the `food_remaining_locs` of that state.
- **Path cost:** Moving to an adjacent square has a cost of 1. Therefore, the cost of a path is the number of edges in the path.
- **Goal Test:** Any state where the number of remaining pellets is zero is a goal state regardless of the (x,y) position of Pacman.
- **Performance Measure:** The number of turns it takes to eat all of the food pellets.

In this problem, we want to minimize the performance metric (i.e. eat all of the pellets in the fewest number of turns). If we cast this problem into a search problem (where states are vertices and actions are edges), then we can use a search algorithm to find the shortest path from our initial state to a goal state. We will use the $A^*$ algorithm equipped with the goal-test function to do so.

In order to use $A^*$, we need to define a heuristic that estimates the cost of the current state to a goal state. Design a heuristic that is admissible and consistent that will solve this problem. Your heuristic must be non-negative and must return 0 when the current state is a goal state.

*Hint:* If we are at a square that contains a food pellet, then the smallest cost to a goal state is the number of movements to enter all of the squares that still contain food pellets. So, something we would like to know (and probably cache beforehand) are the distances between pairs of squares that contain food pellets.

---

One heuristic that we could use is the number of pellets remaing (from `food_remaining_locs.size()`) plus the Manhattan distance from Pacman to the closest food pellet minus 1. If there are no remaining food pellets the distance is considered 1 (The smallest it could be otherwise).

This heuristic is admissible and consistent because it will always overestimate, or be a perfect estimate of, the actual cost. Consider the case where Pacman is directly adjacent to all $n$ of the remaing pellets, which are contiguous. In that case the heuristic will estimate $n + (1 - 1) = n$, which is the actual remaing cost. If there are any gaps between pellets in the future (after Pacman has gobbled up the closest food pellet), the heuristic will pretend that the cost of doing so is 0, so we will always overestimate here.

However, it is still reasonable in that it *will* incentivize Pacman to go towards nearby pellets and eat food when possible. Pacman will continue through groupings of pellets when possible, similar to how a human might play.