Convoluted Kernel Maze

By Kerry Driscoll

Table of Contents

The Situation

- Build a corn maze filled with goodies for the community to explore
    - Goodies placed using Breadth First Search (BFS)

- Reflect and discuss on different strategies (ahem, or algorithms) the locals can use to explore the maze and pick up goodies

Maze Design

- Current Design
  - The Maze was created by beginning at a random coordinate in our field (or grid)
  - Then we mowed 2 paces in a random cardinal direction (Up, Down, Left or Right) until we reached a point where there were no viable directions to mow in (when 2 spaces away in every direction is either already mowed or outside of the field)
  - Placed goodies at random locations within the maze (approximately 10% of the maze's area)

- Alternative Design
  - If we mowed more than 2 cells at a time: there would be fewer twists and turns in our maze, meaning that explorers would further walk in one direction at a time, and will have fewer alternative routes to explore
  - If we mowed at random paces: our maze would be less predictable, and perhaps more fun for the explorers. The farmers would have to be careful when constructing the maze that they are mowing in a viable direction

Finding a Path

- Possible Algorithms to Solve the Maze
  - Breadth First Search
    - No initial knowledge required
    - Strategy: Search every vertex from your starting point and repeat at each subsequent point until you find the “end”
    - Ideal if your maze has many “forks” and if your objective is pick up as many goodies as possible
    - Drawback: will not be efficient at finding the end of the maze because you are exploring every possible fork
  - A* Algorithm
    - Requires initial knowledge of you starting and end coordinates.
    - Strategy: Select path to explore based on an estimate of distance between that path and the end point.
    - Ideal if your objective is to find the end of the maze in the most efficient route
    - Drawback: in a very winding maze, this algorithm may initially select an incorrect path simply because it is in the direction of our end point

- Does knowing the algorithm used to generate the maze influence the best to solve it?
  - The algorithm used to create the maze may not be necessarily be the best to solve it. Your ideal algorithm may vary depending on your the graph’s overall shape (the average number of vertices at each node or the depth of the graph) and your objective (to pick up as much swag as possible or to reach the end of the maze first).

Collecting Swag

- Potential Sorting Algorithms
    - Bubble sort
        - Compare two adjacent items in my collection and re-order in alphabetic order based on the first letter of the item and continue until the entire collection is in order
        - As the size of collection grows, I can adjust this sort to be based on the amount of each items I have from smallest to largest, like Candy Corn: 2, Werewolf: 3, Pumpkin: 5
    - Quicksort
        - Select a random item from my collection as a “pivot” and place all the other items in my list in “less than” or “greater than” partitions relative to this pivot item. Repeat this process until I have only have partitions of made up of one item type.
        - Best to save quicksort for when I am comparing my collection based on the quantity of each items, because it can be tricky to have items with the same value as my “pivot” key

Conclusion

- Algorithms
  - Graph Search
    - Can be used to navigate the maze
    - Different strategies have different payoffs
      - BFS optimal for picking up the most swag possible
      - A* optimal for finding the end of the maze efficiently
  - Sorting
    - Can be used to organize the swag we pick up in the maze
      - Bubble sort could be used for ordering the items alphabetically, then counting
      - Quicksort could be used to order the items based on their quantity