P1 Introduction

Pathfinding in a maze using Dijkstra’s algorithm

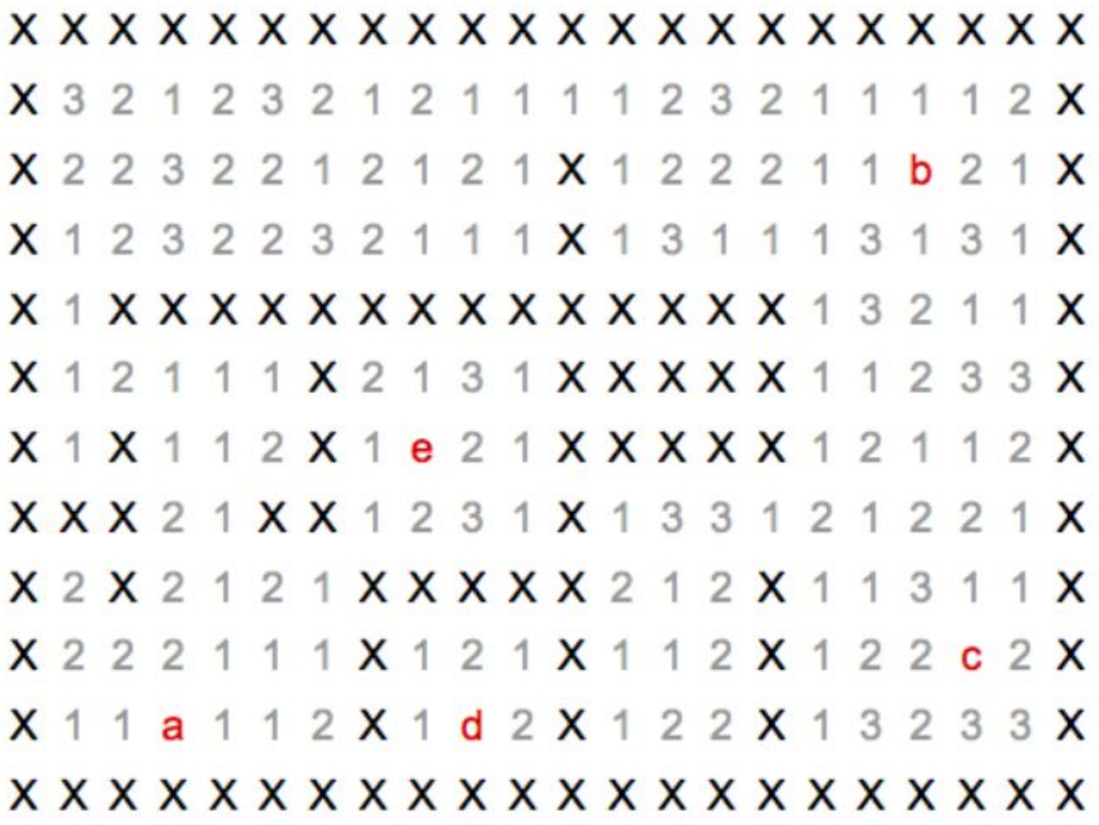
Programming Assignment 1

Provided in Files > P1:

- Code:
  - p1_support.py
  - p1.py (skeleton with specs)
- Some input/output files
- Detailed assignment description in .docx
- These slides

Your job: fill in the blanks in p1.py, test, and submit

Example level text file



```
X X X X X X X X X X X X X X X X X X X X X X
X 3 2 1 2 3 2 1 2 1 1 1 1 2 3 2 1 1 1 1 2 X
X 2 2 3 2 2 1 2 1 2 1 X 1 2 2 2 1 1 b 2 1 X
X 1 2 3 2 2 3 2 1 1 1 X 1 3 1 1 1 3 1 3 1 X
X 1 X X X X X X X X X X X X X X 1 3 2 1 1 X
X 1 2 1 1 1 X 2 1 3 1 X X X X X 1 1 2 3 3 X
X 1 X 1 1 2 X 1 e 2 1 X X X X X 1 2 1 1 2 X
X X X 2 1 X X 1 2 3 1 X 1 3 3 1 2 1 2 2 1 X
X 2 X 2 1 2 1 X X X X X 2 1 2 X 1 1 3 1 1 X
X 2 2 2 1 1 1 X 1 2 1 X 1 1 2 X 1 2 2 c 2 X
X 1 1 a 1 1 2 X 1 d 2 X 1 2 2 X 1 3 2 3 3 X
X X X X X X X X X X X X X X X X X X X X X X
```

Data types

A level is a dict():

```
level = {'walls': walls,
         'spaces': spaces,
         'waypoints': waypoints}
```

Whose values are also collections:

- Walls: A **set** of (x,y) integer tuples
- Spaces: A **dict** from (x,y) tuple keys to numeric cost values (cost to traverse)
- Waypoints: A **dict** from letter waypoint names to (x,y) positions

p1_support.py

load_level(filename)

show_level(level, path=[])

- Recall what the level data structure looks like from slide 4

save_level_costs(level, costs, filename='distance_map.csv')

p1.py

dijkstras_shortest_path(initial_position, destination, graph, adj)

dijkstras_shortest_path_to_all(initial_position, graph, adj)

navigation_edges(level, cell)

test_route(filename, src_waypoint, dst_waypoint)

cost_to_all_cells(filename, src_waypoint, output_filename)

Getting started

Start by experimenting with p1_support:

- Load a level
- Display it to the console
- Display a level and a “dummy” solution to the console (make sure you understand the data type of a path)
- See if you can construct a “dummy” costs dict and try out save_level_costs

Getting started

Look at the skeleton you have to fill in and tackle the pieces one at a time.

- You have a full spec so you can work bottom-up
- Start by figuring out navigation_edges. You can test it from the interpreter (*importlib.reload(module)* could help), use a Jupyter notebook, or run a test program repeatedly.
- Then write out a Dijkstra's implementation!

Last notes

- Look at the documentation for the heapq module! That uses a list as a priority queue.
    - Its documentation will tell you how the list should be structured
- Note that the main part of the program calls your dijkstra functions, passing the navigation_edges *function* as an *argument*
- Keep in mind that a “position” is an (x,y) tuple.
- Refer to the assignment .docx file for the list of files to submit!

Any questions?