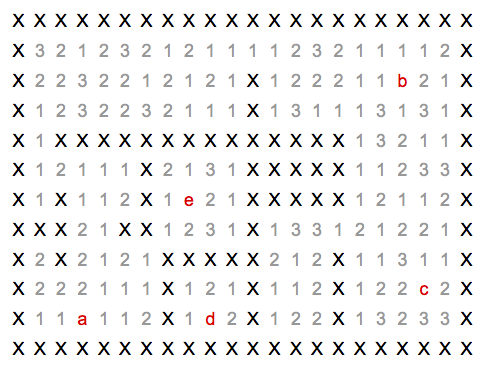
Dijkstra’s in a Dungeon

**Description**

* Implements a Dijkstra’s forward search pathfinding algorithm which returns the path with minimal cost between two waypoints in a maze. Returns None if no path can be found.
* Computes the adjacent cells to a given cell on the level map. Allows movement in 8 directions on the grid, including the diagonals.
* Computes the cost function for horizontal and vertical moves differently, per the calculation covered at the end of this document.
* Movement is only allowed between “spaces” in the level file (not “walls”).

**Example Level File (example.txt)**



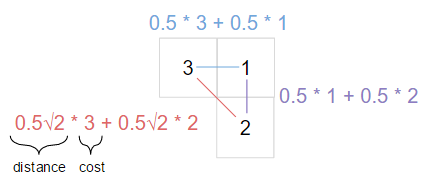
Key:

* X – an impassable wall
* a,b,c,d,e – waypoints
* 1,2,3 – the costs of moving through the open spaces of the maze (Note: all waypoints are assumed to have a cost of 1.)

**Path Cost Computation**

Each open space is associated with a cost. The weight of an edge linking two cells is the sum of the distances traveled in each cell, as weighted by the cells’ respective costs, as shown in the image below. Here we have computed the costs of transitioning between the three cells shown. Euclidean distance is used for the distance between cells, resulting in a longer distance for diagonal movement.

Note: all waypoints are assumed to have a cost of 1



**Support Functions**

The “load\_level” function returns a dictionary:

from maze\_environment import load\_level

level = load\_level('example.txt')

list(level.keys()) # --> ['walls', 'spaces', 'waypoints']

level['walls'] # --> {(0, 0), (0, 1), (0, 2), (0, 3), … }, a set

level['spaces'] # --> {(7, 3): 2, (6, 9): 2, (12, 1): 1, … }, a dictionary

level['waypoints'] # --> {'a': (3, 2), 'b': (18, 2), 'c': (19, 9), …}, a dictionary

**References**

* <https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm>
  + Suggested differences from the pseudo-code from Wikipedia:
* Using heapq for priority queue operations:

from heapq import heappush, heappop

queue = [] # Just a plain list

heappush(queue, (2, 'a')) # enqueuing some pairs

heappush(queue, (42,'b'))

heappush(queue, (1, 'c'))

p1, x1 = heappop(queue) # dequeuing some pairs

p2, x2 = heappop(queue)

p3, x3 = heappop(queue)

assert [x1, x2, x3] == ['c','a','b']

assert [p1, p2, p3] == [1, 2, 42]

assert queue == []