Finding the Most Beautiful Shortest Path

By: Maha Alkhairy

Data

** Street map from (openstreetmap.org) as an html file having places and distances between them.

** Dictionary of places and their beauties provided by the user: the smaller the number, the higher the beauty. Some nodes have no beauty specified

Approach

** Extended existing code for Dijkstra

** Create an weighted undirected graph from the html file with nodes as places, edges as paths between places, and weights representing distances

** Incorporate beauty rating into graph by adding it as a feature of a node

Approach continued...

** Graph is represented as a dictionary of dictionaries to store edge and node values

** The code then adds 100 to the edges that link the nodes that have no beauty as input.

** It then adds the beauty given by the user to the edges that come from the node that has a user inputted beauty. Hence creating weighted directed graph

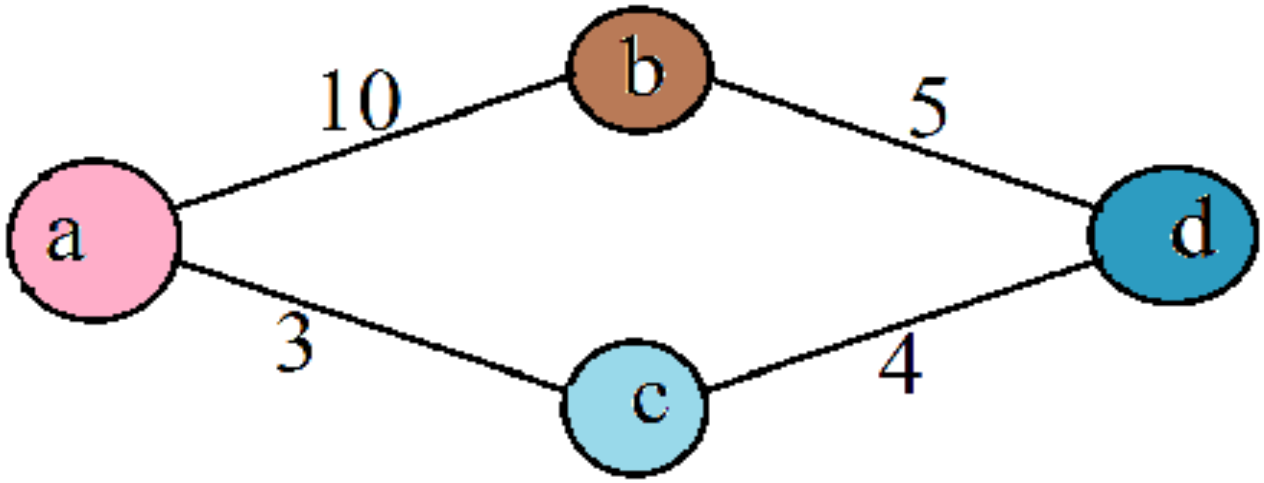
Computing a path:

After the graph is altered as shown above,

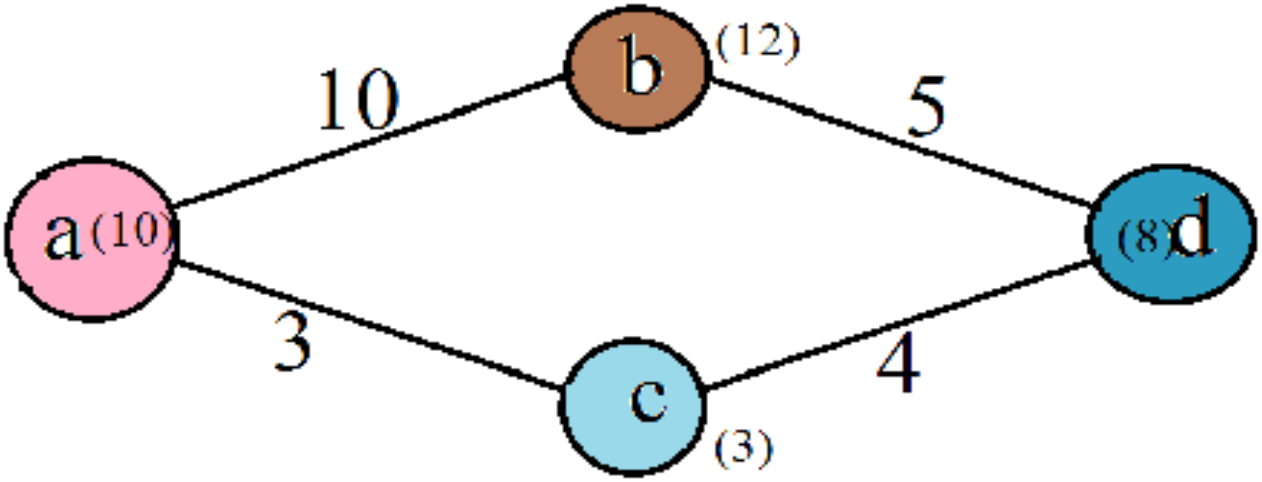
Dijkstra is run on the code and a path is found

to check that it runs and give the best path I ran the code on some user inputed graphs.

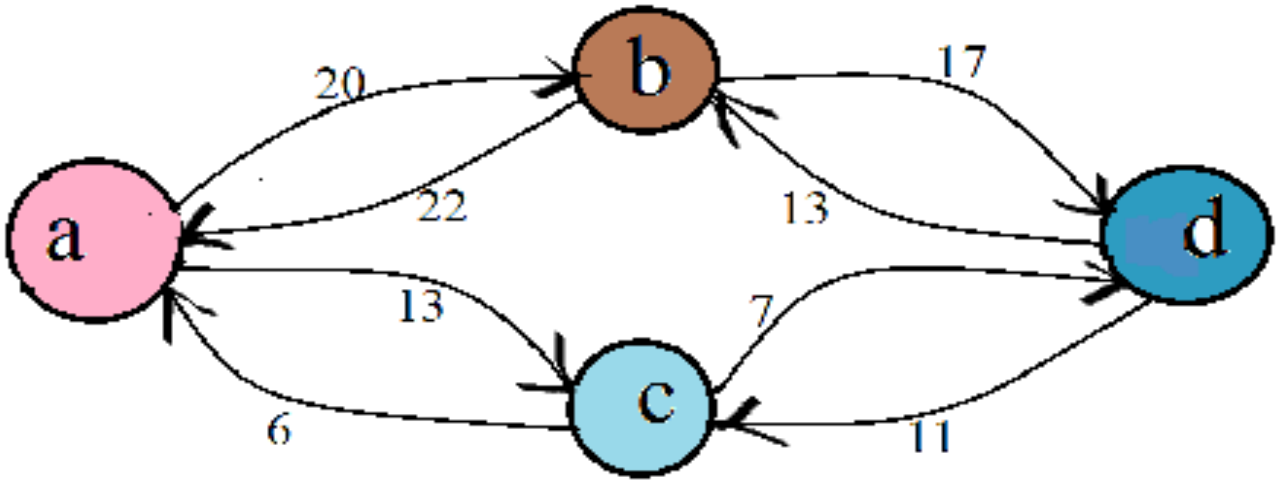
Example:: (original graph)



Graph with user inputted beauty ::



Graph with added beauty ::



Example in code

Enter Graph - with distances "example: {"a":{"b": 12.00}, "b": {"a": 11.99}} : {'a':{'b': 10, 'c': 3} , 'b': {'a':10,'d':5}, 'c': {'a':3, 'd':4}, 'd': {'b': 5, 'c': 4}}

('Graph', "{'a': {'c': 3, 'b': 10}, 'c': {'a': 3, 'd': 4}, 'b': {'a': 10, 'd': 5}, 'd': {'c': 4, 'b': 5}}")

Enter Graph - with beauty "example: {"a":1.00(very beautiful), "b":x(not beautiful) x < 100.00}" : {'a': 10.00, 'b': 12.00, 'c': 3.00, 'd': 8.00}

('Graph Beauty ', "{'a': 10.0, 'c': 3.0, 'b': 12.0, 'd': 8.0}")

Enter "StartNode": 'a'

('StartNode', 'a')

Enter "EndNode": 'd'

('EndNode', 'd')

{'a': {'c': 13.0, 'b': 20.0}, 'c': {'a': 6.0, 'd': 7.0}, 'b': {'a': 22.0, 'd': 17.0}, 'd': {'c': 12.0, 'b': 13.0}}

dist from: a:{'a': 0, 'c': 13.0, 'b': 20.0, 'd': 20.0}

path : {'c': 'a', 'b': 'a', 'd': 'c'}

shortest_path: ['a', 'c', 'd']

shortest_path_distance: 20.0 meters

Enter Graph - with distances "example: {"a":{"b": 12.00}, "b": {"a": 11.99}} : {'a':{'b': 10, 'c': 3} , 'b': {'a':10,'d':5}, 'c': {'a':3, 'd':4}, 'd': {'b': 5, 'c': 4}}

('Graph', "{'a': {'c': 3, 'b': 10}, 'c': {'a': 3, 'd': 4}, 'b': {'a': 10, 'd': 5}, 'd': {'c': 4, 'b': 5}}")

Enter Graph - with beauty "example: {"a":1.00(very beautiful), "b":x(not beautiful) x < 100.00}" : {'a': 10.00, 'b': 12.00, 'c': 3.00, 'd': 8.00}

('Graph Beauty ', "{'a': 10.0, 'c': 3.0, 'b': 12.0, 'd': 8.0}")

Enter "StartNode": 'd'

('StartNode', 'd')

Enter "EndNode": 'a'

('EndNode', 'a')

{'a': {'c': 13.0, 'b': 20.0}, 'c': {'a': 6.0, 'd': 7.0}, 'b': {'a': 22.0, 'd': 17.0}, 'd': {'c': 12.0, 'b': 13.0}}

dist from: d:{'a': 18.0, 'c': 12.0, 'b': 13.0, 'd': 0}

path : {'a': 'c', 'c': 'd', 'b': 'd'}

shortest_path: ['d', 'c', 'a']

shortest_path_distance: 18.0 meters

Credits

Got idea from

<http://www.technologyreview.com/view/528836/forget-the-shortest-route-across-a-city-new-algorithm-finds-the-most-beautiful>

Basic Dijkstra code

<https://github.com/nvictus/priority-queue-dictionary/blob/master/examples/dijkstra.py>

Data

<https://www.openstreetmap.org/export#map=19/42.38635/-71.07889>.