



BuildShortestPathTree method:

Reset Graph State

Start from endpoint vertex

Set distance for endpoint vertex = 0

Create a queue for pending edges to evaluate (called “the fringe” in some texts)

Loop

If there is pending vertex to evaluate

Traverse all incoming and valid edges for this vertex

Store them into the fringe

Extract next shortest weighted edge from the fringe

If there is no pending edge

Finished!

If tail (vertex) of extracted edge has not been evaluated

Distance from vertex to endpoint = edge’s weight + distance from edge’s head to endpoint

Mark this edge as a shortest path edge for current vertex

else

Do not traverse this vertex in next loop

End Loop

BuildSidetracksHeap method:

Create a heap (queue) to store sidetracks for each path (weight is the sum of deltas of sidetracks) \*

Store in queue an empty sidetrack list, representing the shortest path

Start to evaluate from start vertex in shortest path

Loop

For each sidetrack outcoming from current vertex

Create a copy of previous vertex’s sidetrack list

Add sidetrack to list

Store sidetrack list into the heap

End for

If there is a next vertex in shortest path

Recursively evaluate sidetracks for next vertex

else

Finished!

End Loop

Sidetracks are those outcoming edges that are not part of the shortest path

Delta of a sidetrack is the extra distance to take comparing with the shortest path

Shortest path edges always have a delta = 0

Internal storage of PathsHeap

Each path is represented uniquely by its sidetracks collection.

Sidetracks are represented here by its delta values. Rest of edges in each path are extracted from shortest path (S-D-E-F-J-T)

Empty collection (first) represents shortest path

Eppstein’s original example graph

2

9

18

20

10

8

14

25

11

13

27

14

15

15

20

12

7

Shortest path tree

Other paths

Distances to endpoint and sidetracks

3

9

4

1

10

6

Modified example

2

9

18

20

10

8

14

25

11

13

27

14

15

15

20

12

7

5

16

35

30