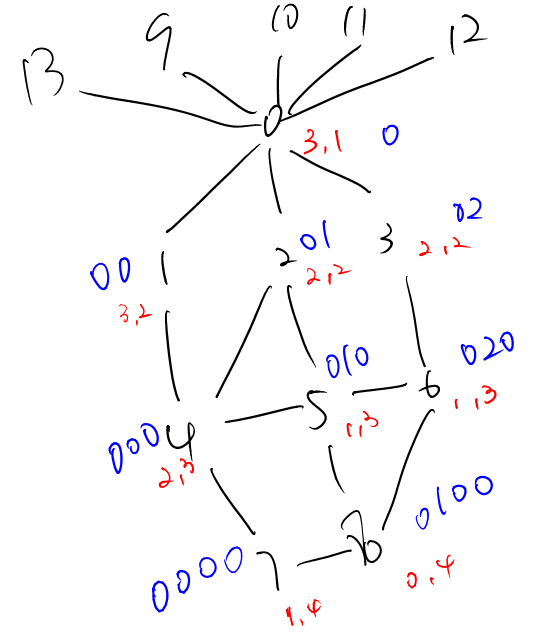
Combine tree structure with LCR

# The graph



Root of tree structure: Node 0

Landmarks: Node 8 and Node 9

The coding number of each node are write down with blue numbers.

The coordinates of LCR are write down with red numbers.

# Some Terms:

1. Coding distance:   
   src node code size + dst node code size - 2\* size of common part  
   For example, the coding distance of node 4 to node 6 in above graph is :  
   size of (0,0,0) + size of (0,2,0) – 2 \* size of (0) = 3 + 3 – 2 \* 1 = 4
2. Coordinate distance:  
   Weighted Euclidean distance  
   For example, the coordinate distance of node 4 to node 6 in above graph is:
   1. The weight: (2-1, 3-3) = (1, 0)
   2. The distance = (2-1) ^ 2 \* 1 ^ 2 + (3-3) ^ 2 \* 0 ^ 2 = 1 + 0 = 1

# Combine policy 1:

1. If there are neighbors with coordinate distance shorter than current node by more than a certain threshold, choose the neighbor with shortest coordinate distance.
2. Else, choose the neighbor with shortest coding distance

But this policy will give us **infinite loop** some times, for example from node 1 to node 6:

The threshold for coordinate is set to 1 in this experiment:

1. 1st step:  
   6 to 7: 6(5, 1): 3[4, 4] 5[5, 1] 8[6, 0]

This line means current node is node 6, its neighbor are 3, 5, 8. The blue number are coding distance to node 6, the red number are coordinate distance to node 6.  
So node 6 has coding distance 5 and coordinate distance 1. Since none of its neighbor has coordinate distance shorter than node 6’s coordinate distance (1) by more than the threshold 1, so the neighbor with shortest coding distance will be chosen, that is **node 3** in this case.

1. 2nd step:  
   6 to 7: 3(4, 4): 0[3, 9] 6[5, 1]  
   So we go to node 3 and check its neighbor 0, 6 in this step. Since node 6 have a coordinate distance which is shorter than node 3 by 3, so **node 6** will be chosen for next step.

Here the infinite loop starts.

# Combine policy 2:

1. If there are neighbors with coordinate distance shorter than current node by more than a certain threshold, choose the neighbor with shortest coordinate distance.
2. Else, choose the neighbor with shortest coding distance **and with coordinate distance no longer than current node’s coordinate distance plus the threshold.**

No infinite loop anymore, but there are some pair that cannot find a path. For example,   
6 to 7: 6(5, 1): 3[4, 4] 5[5, 1] 8[6, 0]  
No neighbor of node 6 meet both rule above.

Although set the threshold even lower will make this pair work, but there are other pairs won’t work.

# Combine policy 3:

1. If there are neighbors with coordinate distance shorter than current node by more than a certain threshold, choose the neighbor with shortest coordinate distance.
2. Else, choose the neighbor with shortest coding distance
3. **Never choose a node which is already in the path**

Same problem with policy 2, some pair may fail to find a path.

# Other important observations:

Put aside the infinite loop and the paths that are failed. There are still some problem of this combined method, for example:

8 to 1: 8(4, 97): 5[3, 40] 6[3, 40] 7[2, 52]

Node 5 and Node 6 have exactly same coordinate distance to node 6. Although they both lead to a suboptimal path, but it gives me a feeling that such situation must be very common in large networks and it may misleading the algorithm. Note that even in this small network, they still have coordinate distance shorter than optimal node (Node 7 in this case) by 12

Similar situation in this graph:

8 to 3: 8(4, 32): 5[3, 8] 6[1, 8] 7[4, 20]

4 to 6: 5(4, 0): 2[3, 1] 4[4, 1] 6[0, 0] 8[5, 1]

The main reason for such a problem is that there are too few landmarks, but consider the scale of the network, only 14 nodes and relatively sparse. So I really don’t think 4 or 5 landmarks or even tens of landmarks will help the decision on larger networks.

# More landmarks

I have raised the number of landmarks to 4, but there still are either infinite loop or failed pairs.

I will attach some results if you want them.