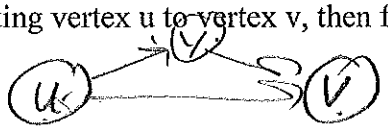
Approximation Algorithm for TSP with Triangular Inequality

Restrictions on the weighted, undirected graph G=(V, E):

1. There is an edge connecting every two distinct vertices.

2. Triangular Inequality: If W(u, v) denotes the weight on the edge connecting vertex u to vertex v, then for every other vertex y,

$$W(u, v) \leq W(u, y) + W(y, v).$$



NOTES:

- These conditions satisfy automatically by a lot of natural graph problems, e.g., cities on a planar map with weights being as-the-crow-flys (Euclidean distances).
- Even with these restrictions, the problem is still NP-hard.

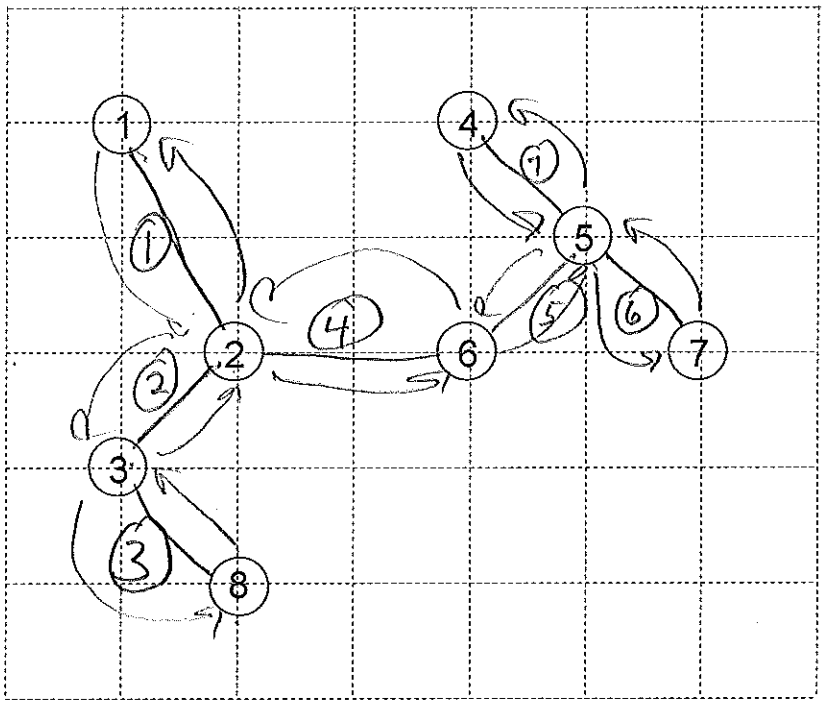
A simple TSP approximation algorithm:

Step 1. Determine a Minimum Spanning Tree (MST) for G (e.g., Prim's Algorithm section 7.8.3)

Step 2. Construct a path that visits every node by performing a preorder walk of the MST. (A *preorder walk* lists a tree node every time the node is encounter including when it is first visited and "backtracked" through.)

Step 3. Create a tour by removing vertices from the path in step 2 by taking shortcuts.

1) (Step 1) Determine a Minimum Spanning Tree (MST) for G (e.g., Prim's Algorithm) if we start with vertex 1 in the MST. (Assume edges connecting all vertices with their Euclidean distances)



Prim's algorithm is a greedy algorithm that performs the following:

a) Select a vertex at random to be in the MST.

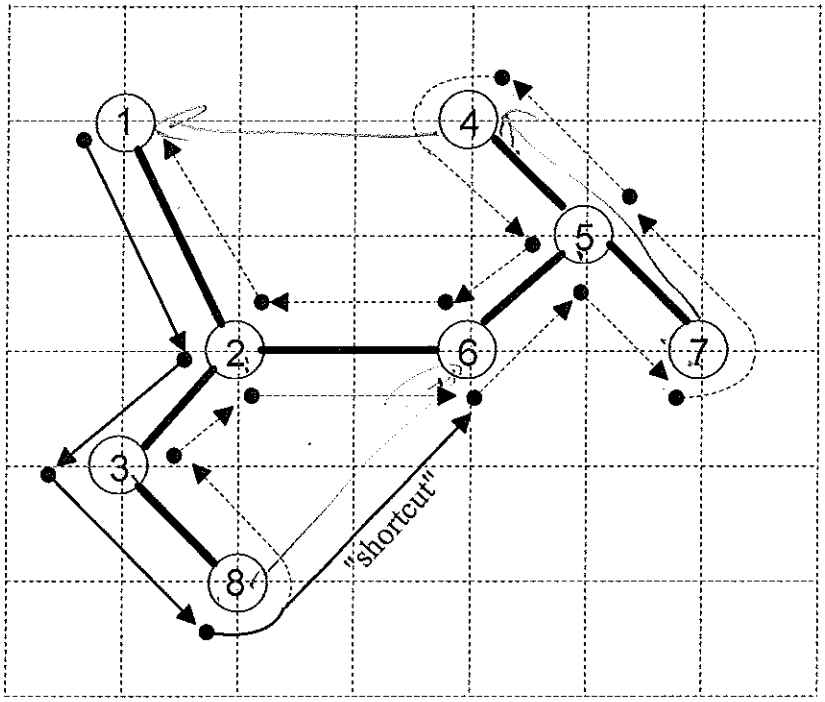
b) Until all the vertices are in the MST:
   - Find the closest vertex not in the MST, i.e., vertex closest to any vertex in the MST

Add this vertex using this edge to the MST

2) (Step 2) Determine the preorder walk of the MST.

1, 2, 3, 8, 3, 2, 6, 5, 7, 5, 4, 5, 6, 2, 1

3) (Step 3) Complete a tour by removing vertices from the path in step 2 by taking shortcuts.



a) Finish removing vertices from the preorder-walk path to create a tour by taking shortcuts:
[1, 2, 3, 8, ~~3~~, ~~2~~, 6, 5, 7, ~~5~~, 4, ~~6~~, ~~2~~, 1]

b) When scanning the above path, how did you know which vertices to eliminate to take a shortcut?

any already seen on path

4) Let's determine how close our approximation algorithm gets to the actual TSP tour.

a) If we take the optimal TSP tour and remove an edge, what do we have?

MST ≤ a spanning tree < opt tour

b) What is the relationship between the distance of the MST and the optimal TSP tour?

MST < opt tour

c) What is the relationship between the distance of the MST and the distance of the preorder-walk of the MST?

2 * MST = dist. of pre-order walk of MST

$\left(\frac{1}{2}\right)$ dist. of pre-order walk of MST = MST

d) What is the relationship between the distance of the preorder-walk of the MST and the tour obtained from the preorder-walk of the MST?

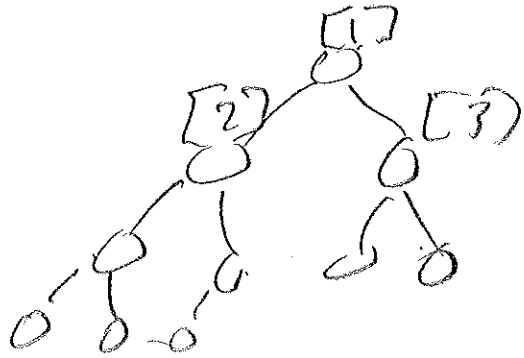
$\left(\frac{1}{2}\right)$ tour obtained from preorder walk of MST < $\left(\frac{1}{2}\right)$ dist. of preorder walk of MST = MST < opt tour

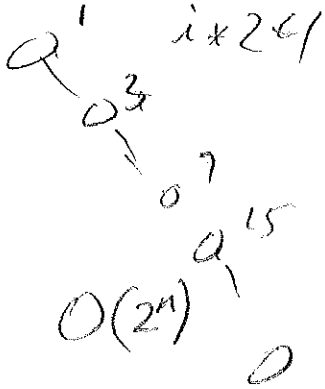
e) What is the relationship between the tour obtained from the preorder-walk of the MST and the optimal TSP tour?

$\left(\frac{1}{2}\right)$ tour obtained from preorder walk of MST < opt tour

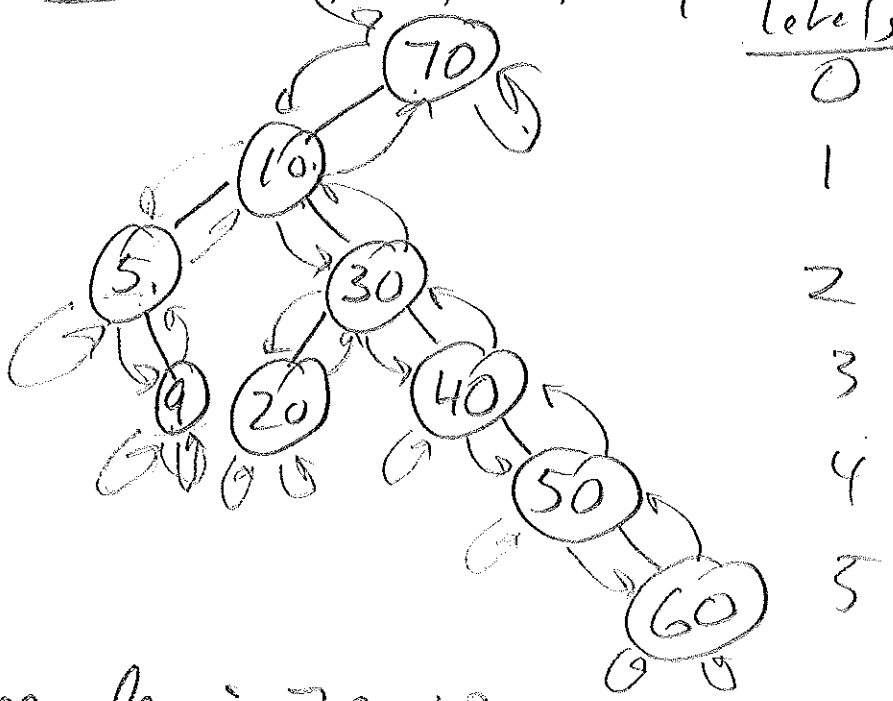
tour obtained from preorder walk of MST < 2 * opt. tour

Trees





BST: 70, 10, 30, 40, 50, 20, 60, 5, 9



levels

0

1

2

3

4

5

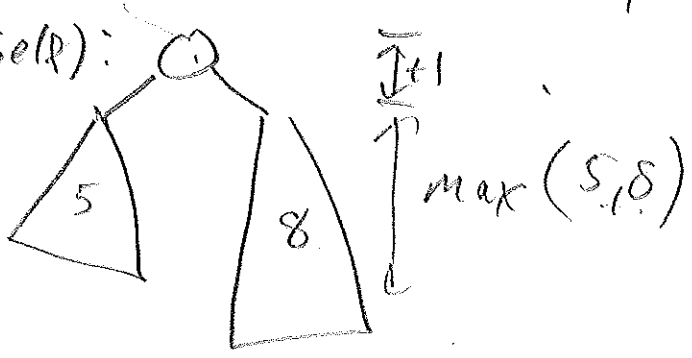
height = 5

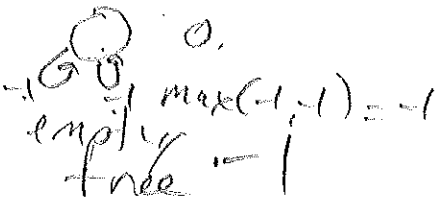
preorder: 70, 10, 5, 9, 30, 20, 40, 50, 60

inorder: 5, 9, 10, 20, 30, 40, 50, 60, 70

postorder: 9, 5, 20, 60, 50, 40, 30, 10, 70

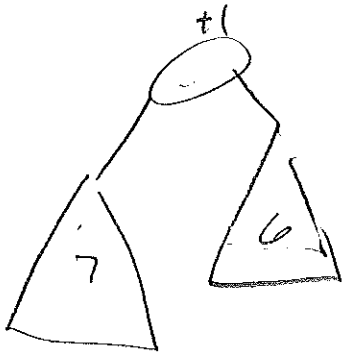
```
def height(self):
```



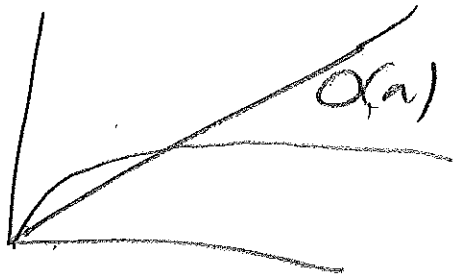


AVL tree - height-balance

AVL-tree v.s. BST
$1.44 \times O(\log_2 n)$? $O(n)$



balance factor = ht. left subtree − ht. right subtree



70, 10, 30, 40, 50, 20, 60, 5, 9