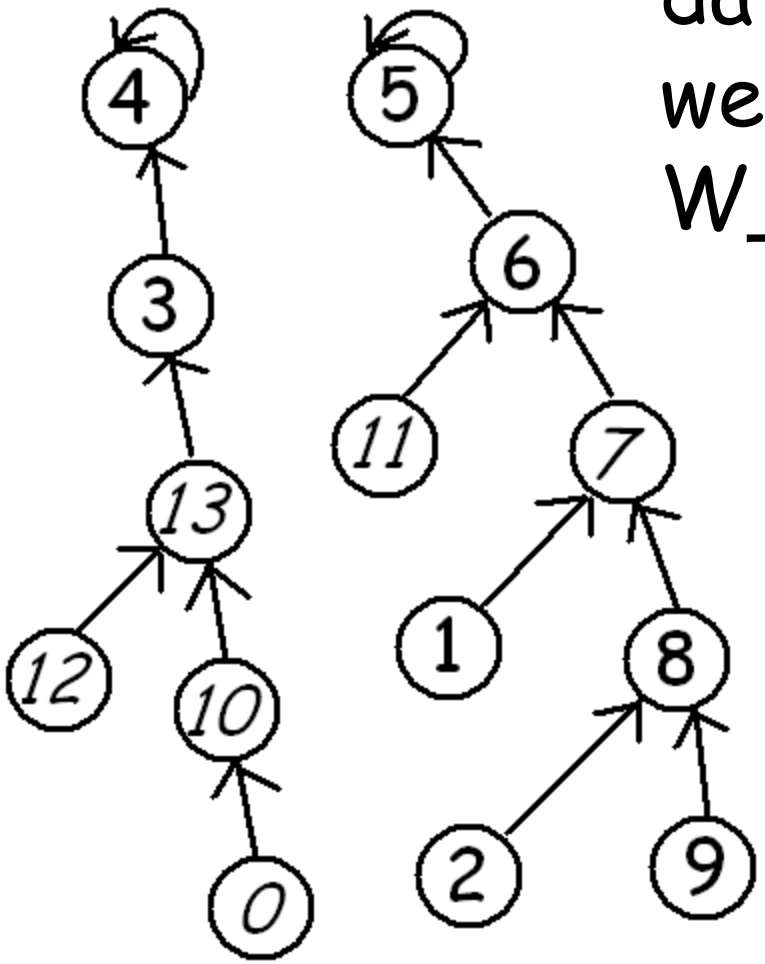


1. Draw the union find data structure after the weighted union W_UNION(0,8).

Use a weighted union that calls a collapsing find.

2. What is in the parent array after the union?

Final exam tutorial time:
Thursday Dec. 5: from 11am until all questions are answered. ECS 116.
Programming Assignment #4B: Upload to connex by Tues. Nov. 26, 11:55pm.
I gave you a program for generating random MST problems for testing.
Written Assignment #5: due Wed. Dec. 4, beginning of class.
Please take the time to give some feedback to your lab instructor.

My random spanning tree problem generator will crash if your computer does not have enough space.

To ask for less space:

#define NMAX 1024

change to

#define NMAX 128

and recompile.

To get more space, on unix/csh:

limit stacksize unlimited

under cygwin:

set ulimit –s unlimited

Google for instructions for other machines.

Minimum weight spanning tree:

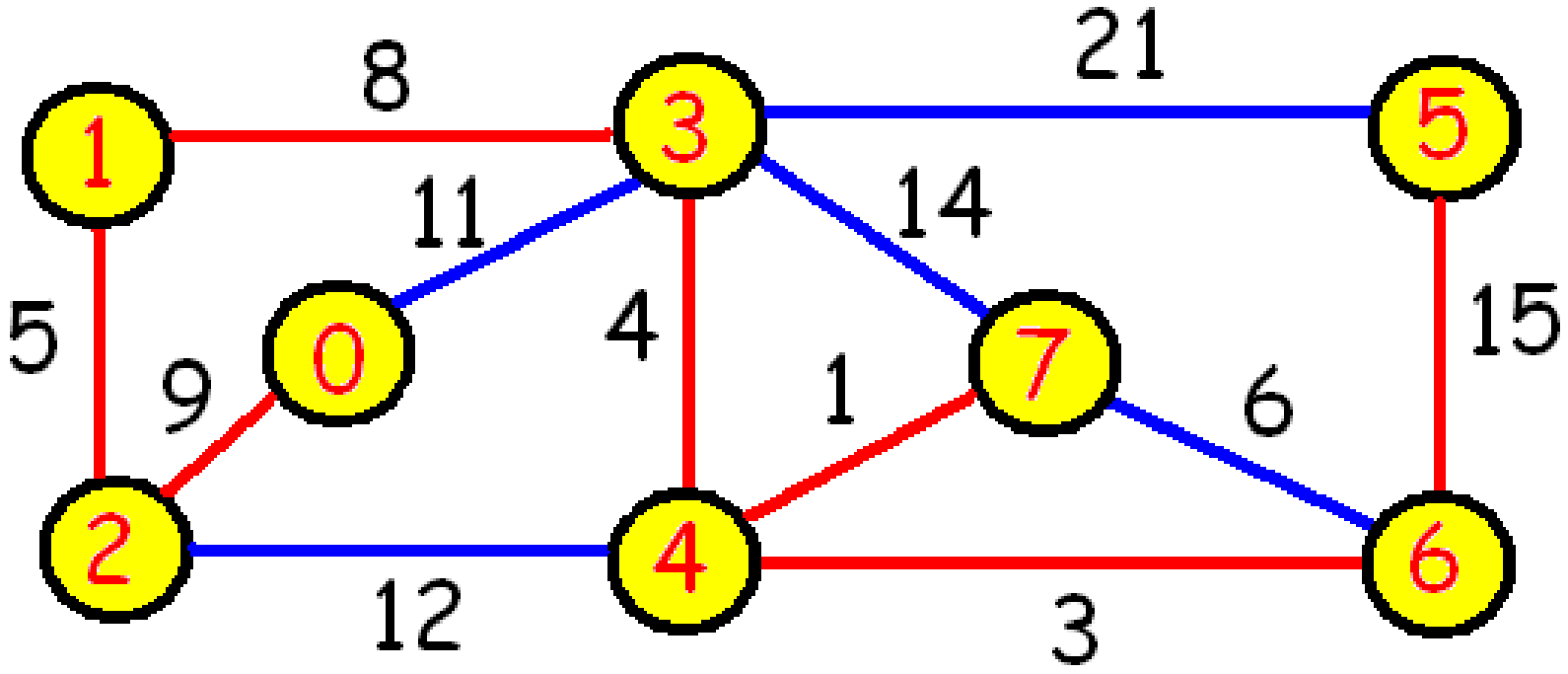
Put a non-negative weight on each edge.

Weight of a tree: sum of weights of its edges.

Problem: find a spanning tree of graph G with minimum weight.

Kruskal's algorithm: sort edges by weight.

Then for each edge: add it to the tree if its endpoints are in different components.



Edge weights:

1, 3, 4, 5, 6, 8, 9, 11, 12, 14, 15, 21

A throwaway approach for a researcher:

Presort edges by weight using a system sort (mergeSort) before using them for input to the program.

To help ensure correctness- use a flat union/find.

How much time does this approach take?

The Dijkstra/Prim algorithm for finding a minimum weight spanning tree (MST).

We have already seen Kruskal's algorithm.

The Dijkstra/Prim algorithm is a second approach. A very similar algorithm can be used to find a shortest path from a fixed vertex s to all the other vertices in the graph.

High level description:

Put a vertex in the tree.

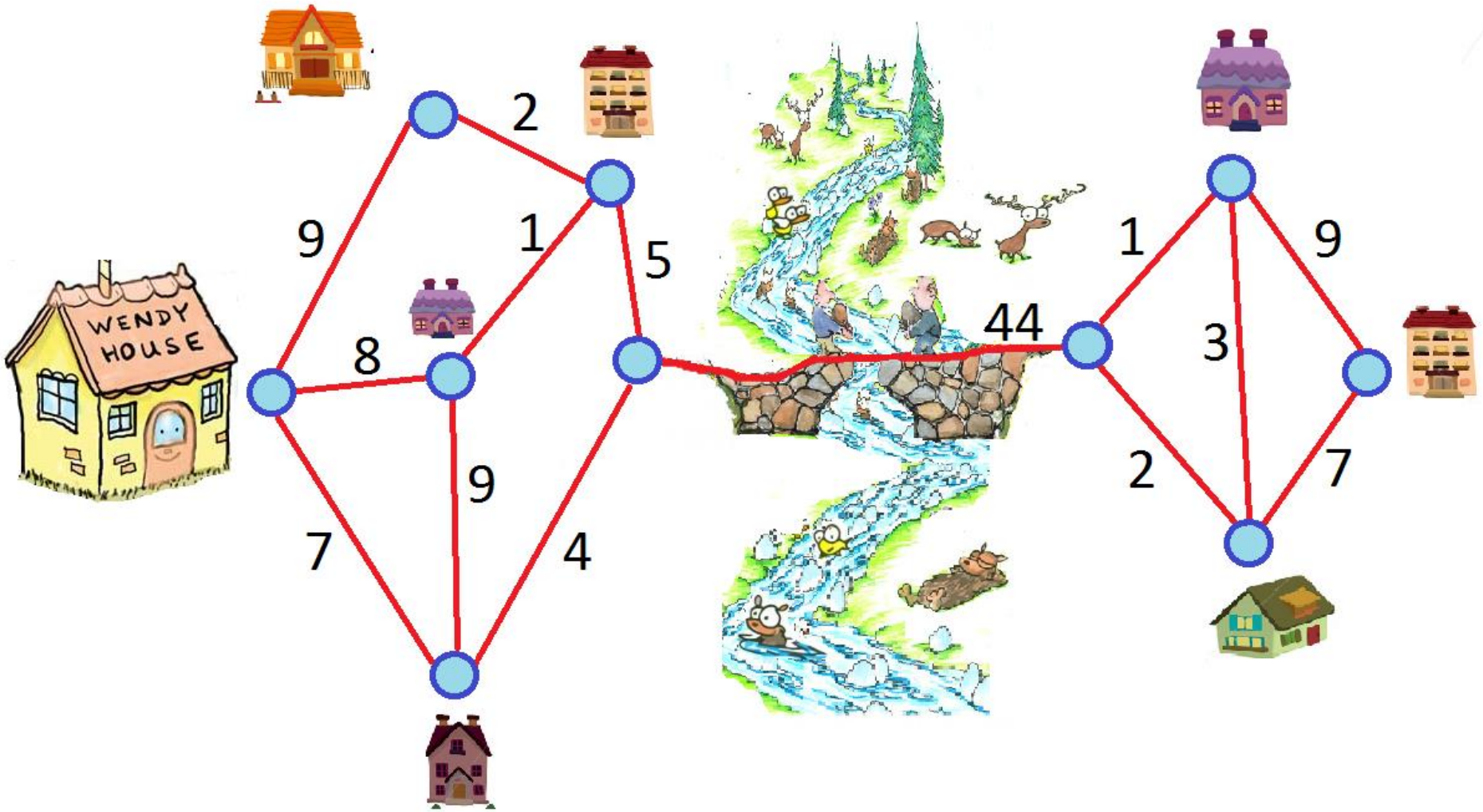
Repeat

1. Let e be a minimum weight edge from a tree vertex to a vertex not in the tree.

2. Add e to T and update the data structures.

Until n-1 edges have been selected.

The Muddy City Problem



Pave roads starting at my house without getting muddy.

Crucial Observation: Consider the edges which go from tree vertices to a vertex v which is not yet in the tree: only the minimum weight one is a candidate for adding to the MST.

The data structures used to keep track of the minimum weight edge from each tree vertex to a non-tree vertex are:

tree[i]= true if i is in the tree and false otherwise.

min_wt[i]= weight of minimum weight edge to vertex i

closest[i]= the vertex in the tree which is closest to vertex i.

Initialization:

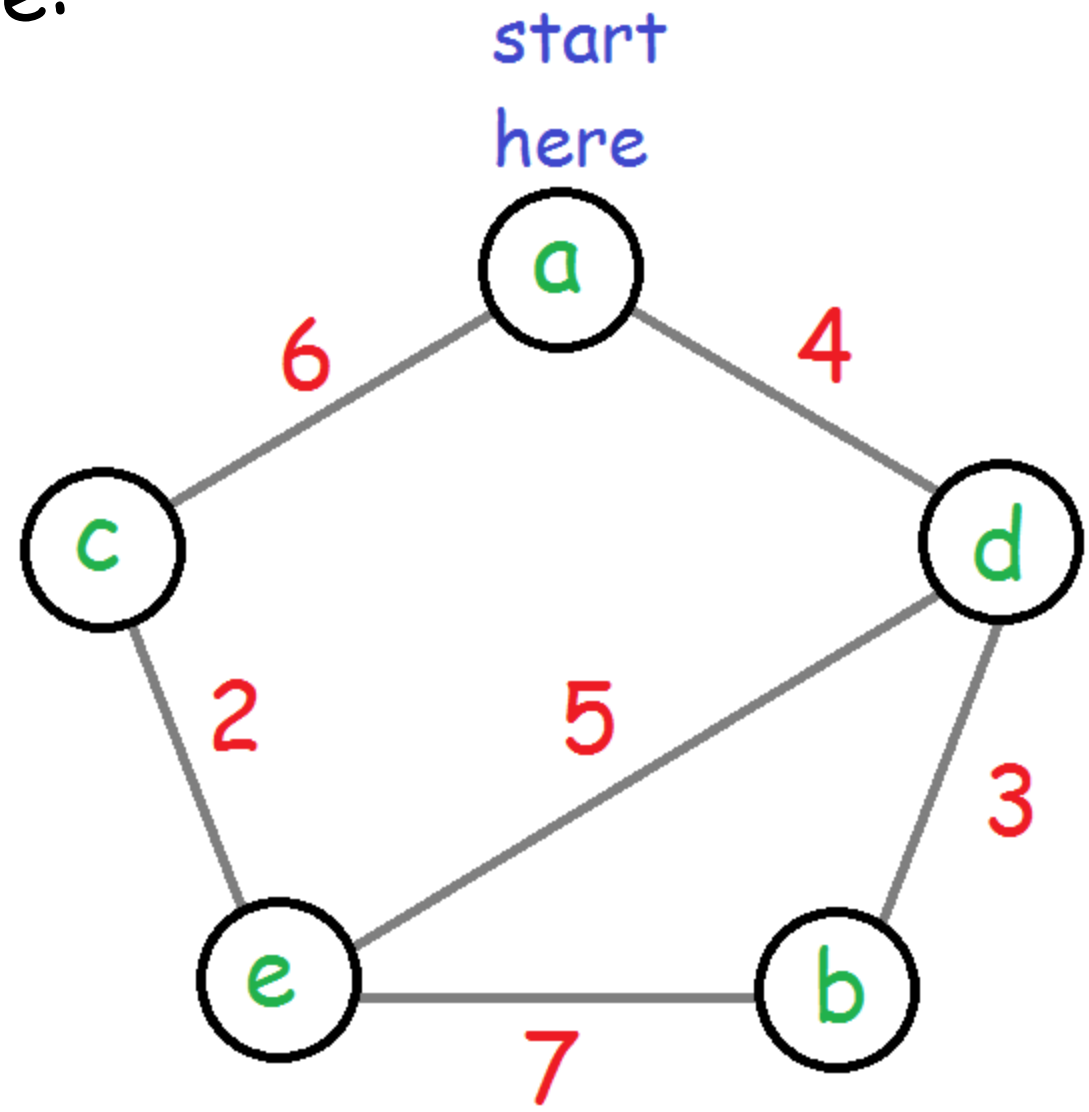
```
for (i=0; i < n; i++)
{
    min_wt[i]= maximum weight +1;   // ∞
    closest[i]= -1;                 // NULL = Λ
    tree[i]= false;
}
```

To add vertex i to the tree:

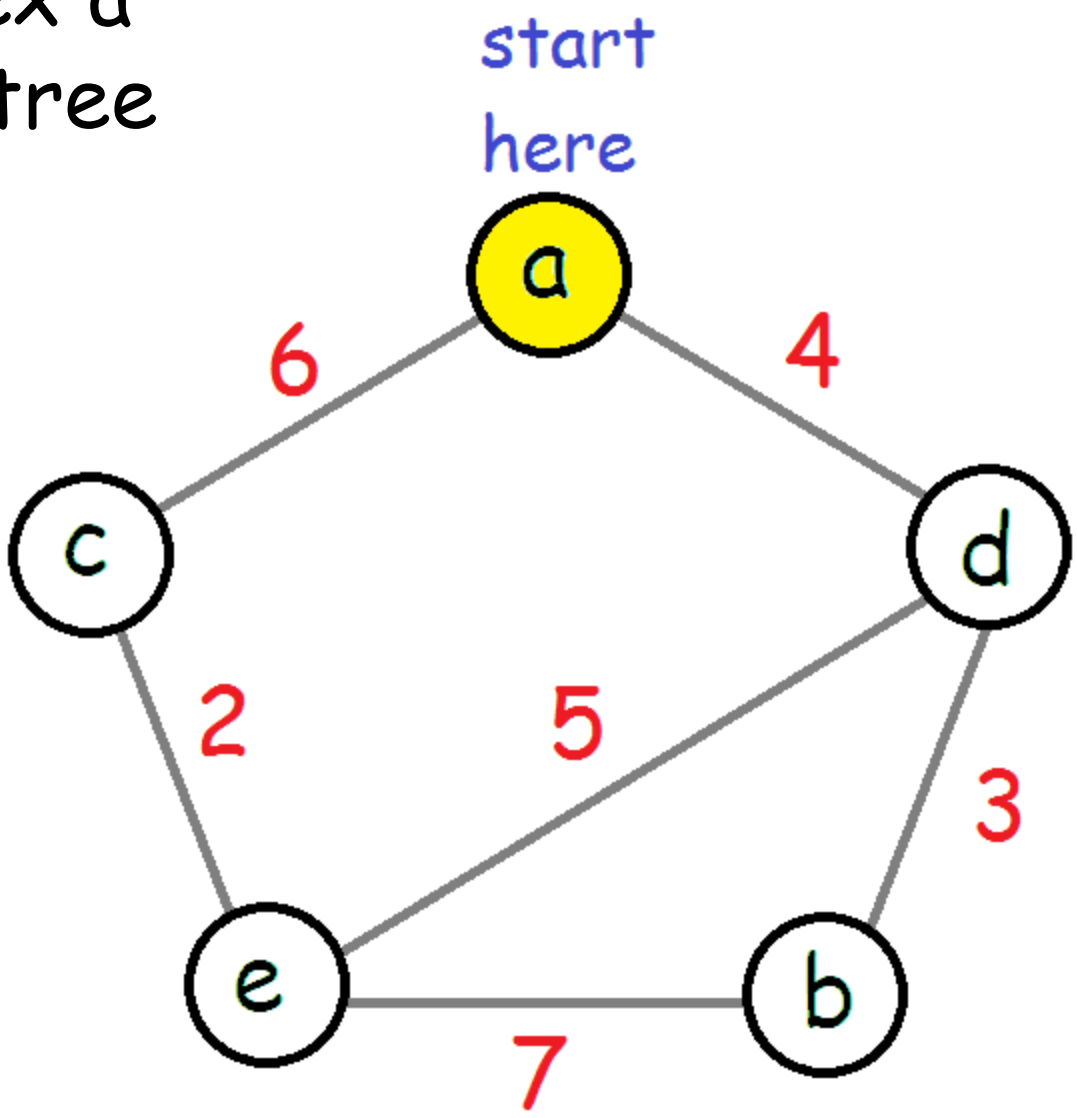
```
tree[i]= true
for each neighbour j of i do
{
    if ( not tree[j]) then
        if min_wt[j] > the weight of edge (i,j)
        {
            min_wt[j]= weight of edge (i,j)
            closest[j]= i
         }
}
```

To choose the minimum weight edge from a tree vertex to a vertex not in the tree, we can simply scan through the min_wt values for each of the non-tree vertices and select the minimum in O(n) time.

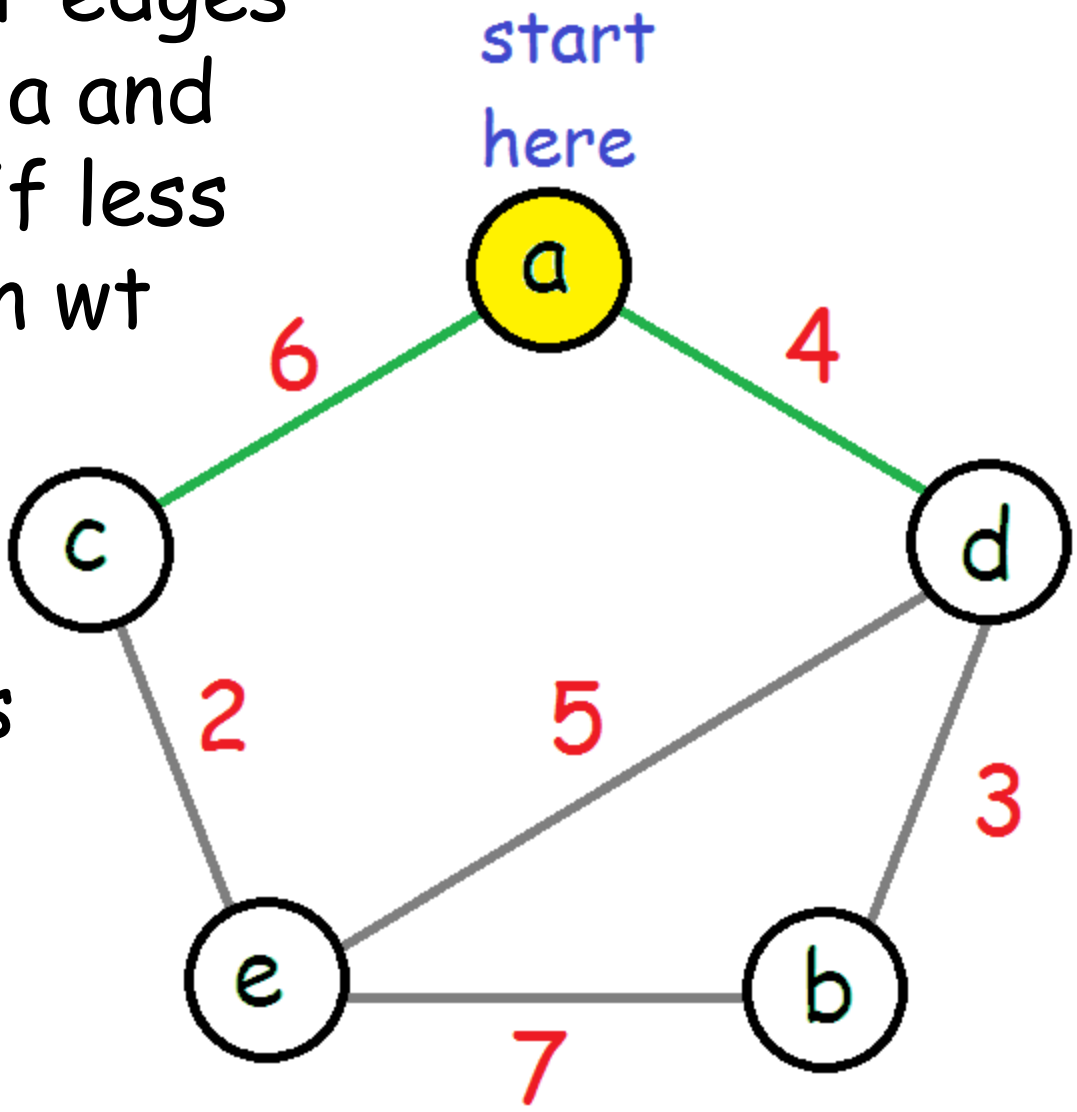
Example:



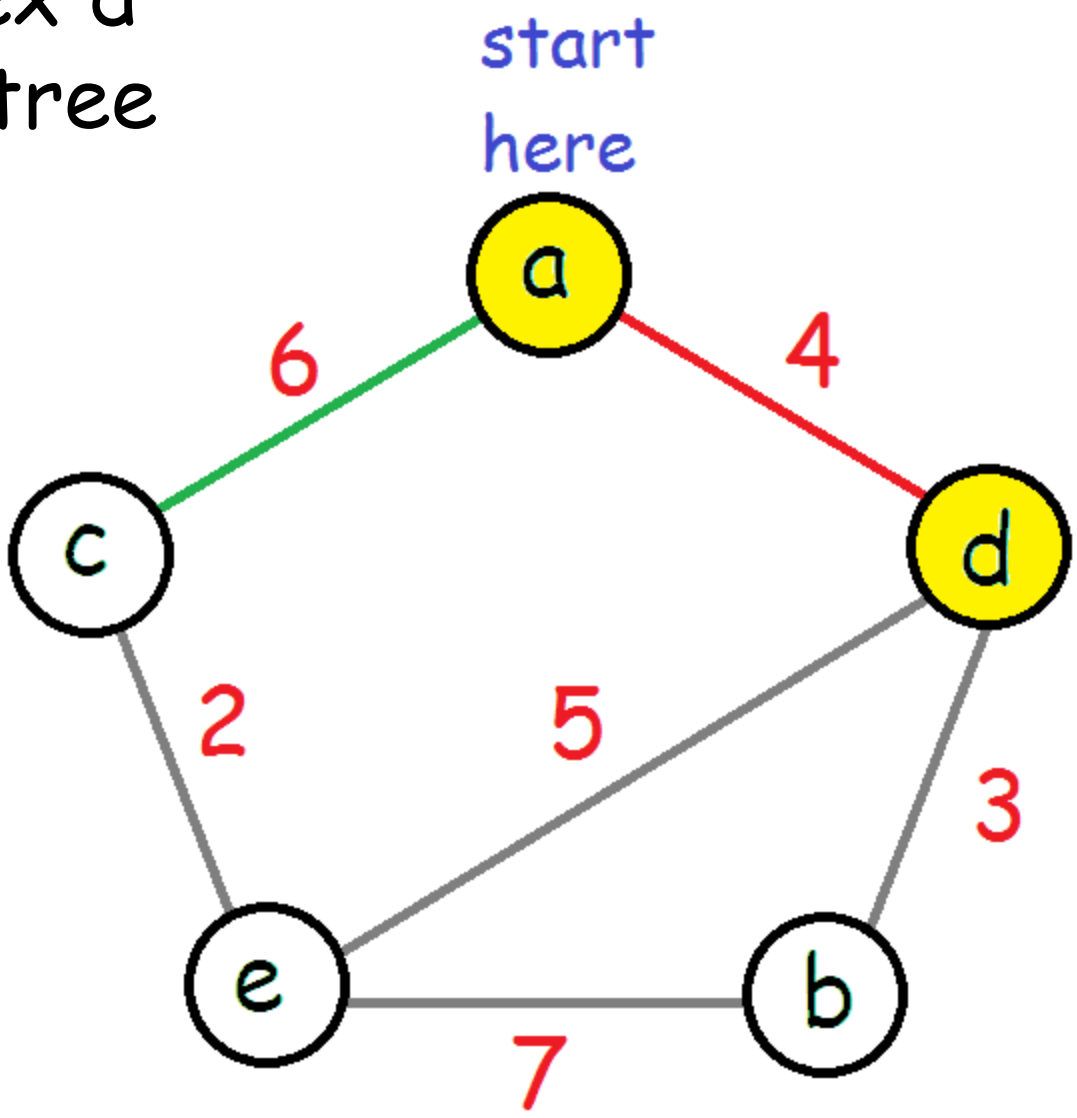
Put vertex a into the tree



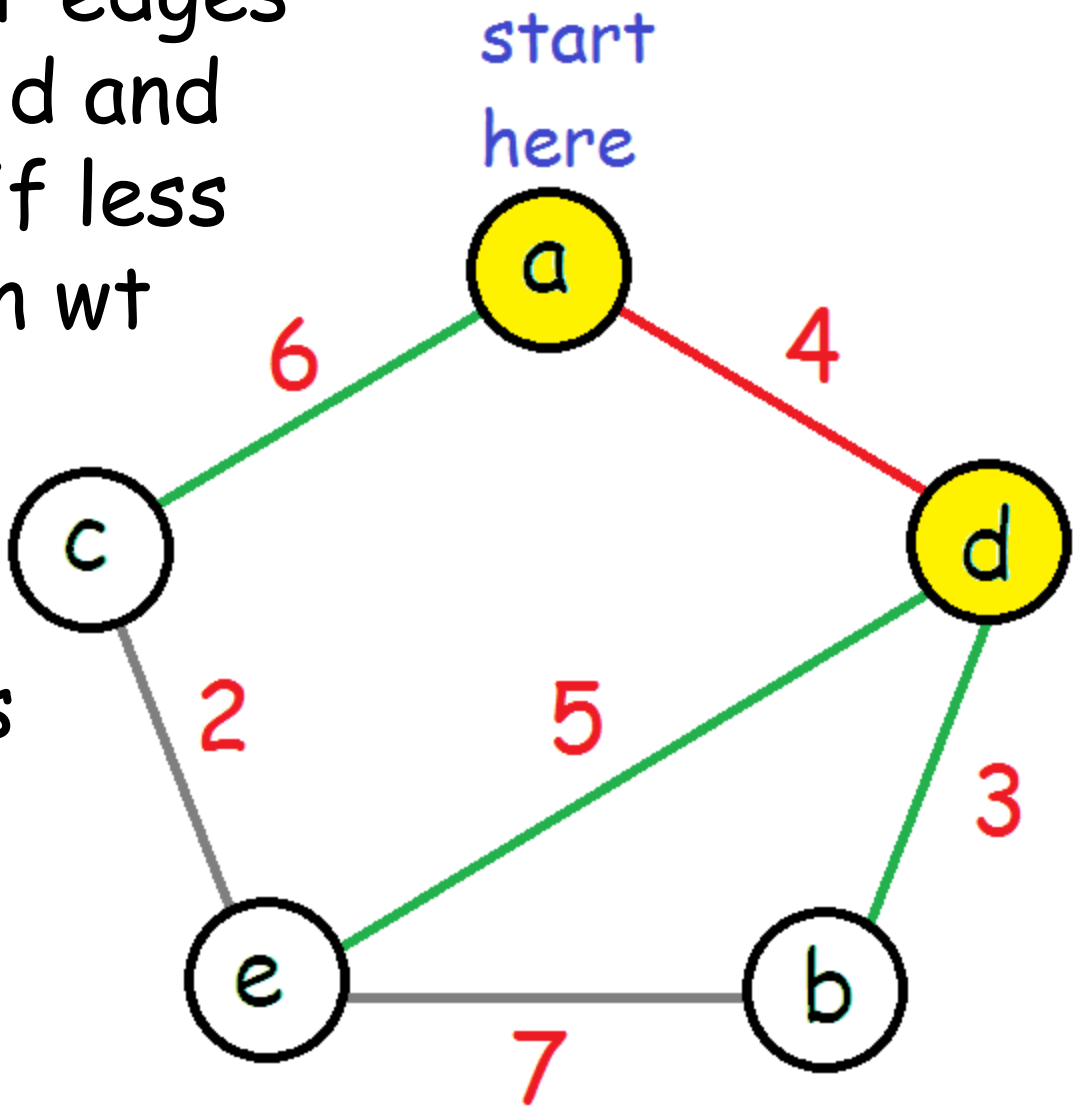
Consider edges next to a and record if less than min wt edges to those vertices



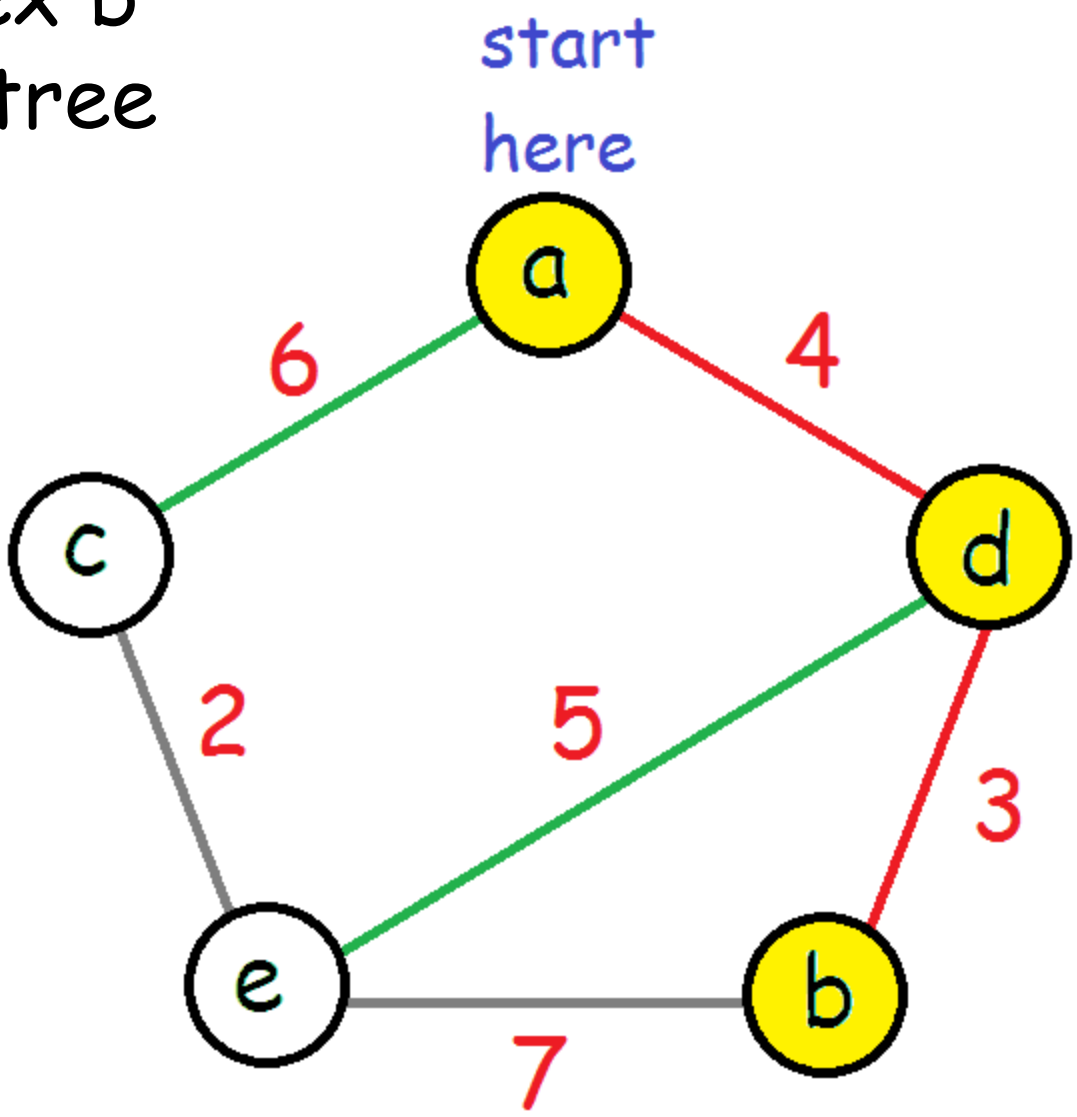
Put vertex d into the tree



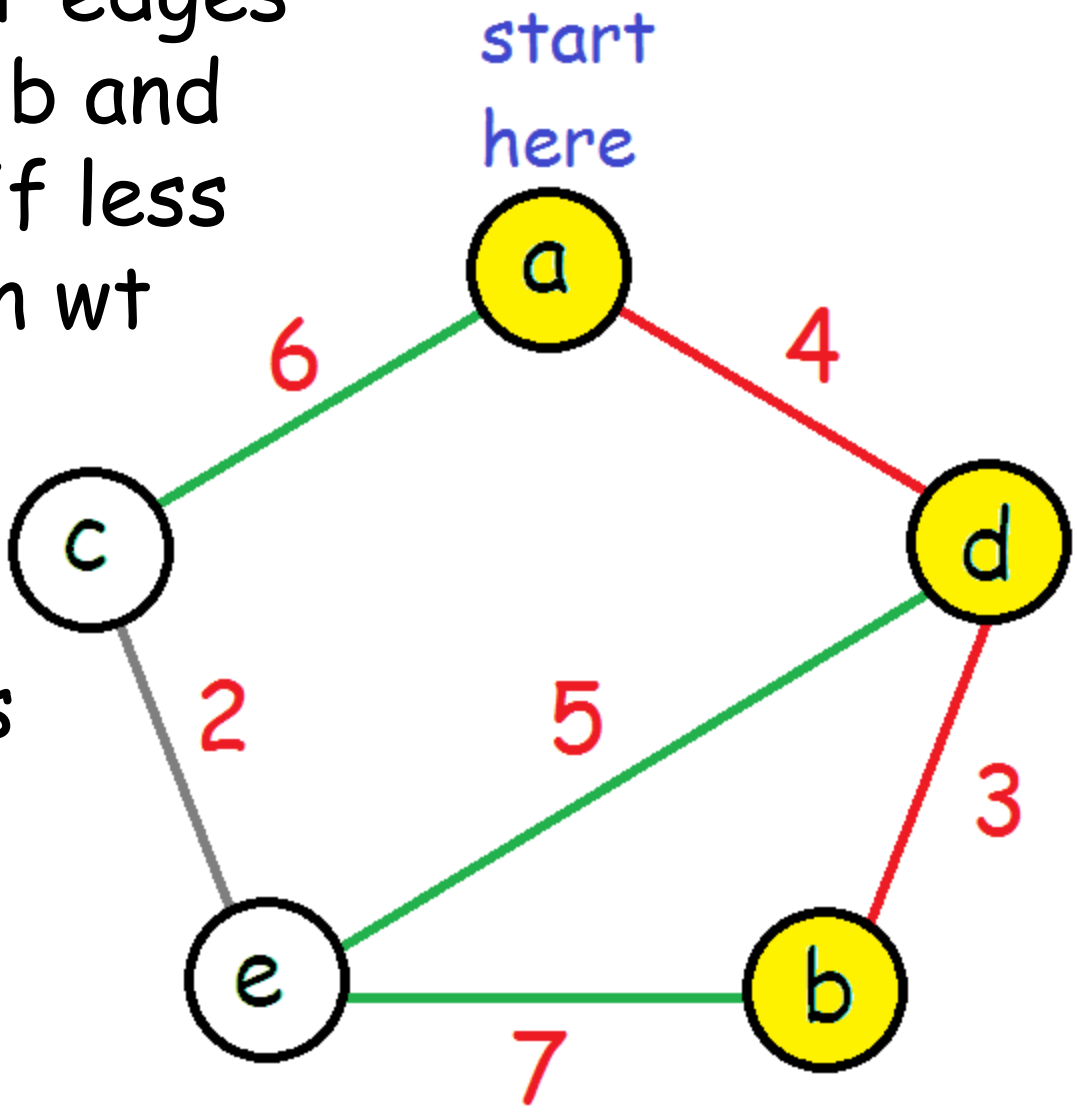
Consider edges next to d and record if less than min wt edges to those vertices



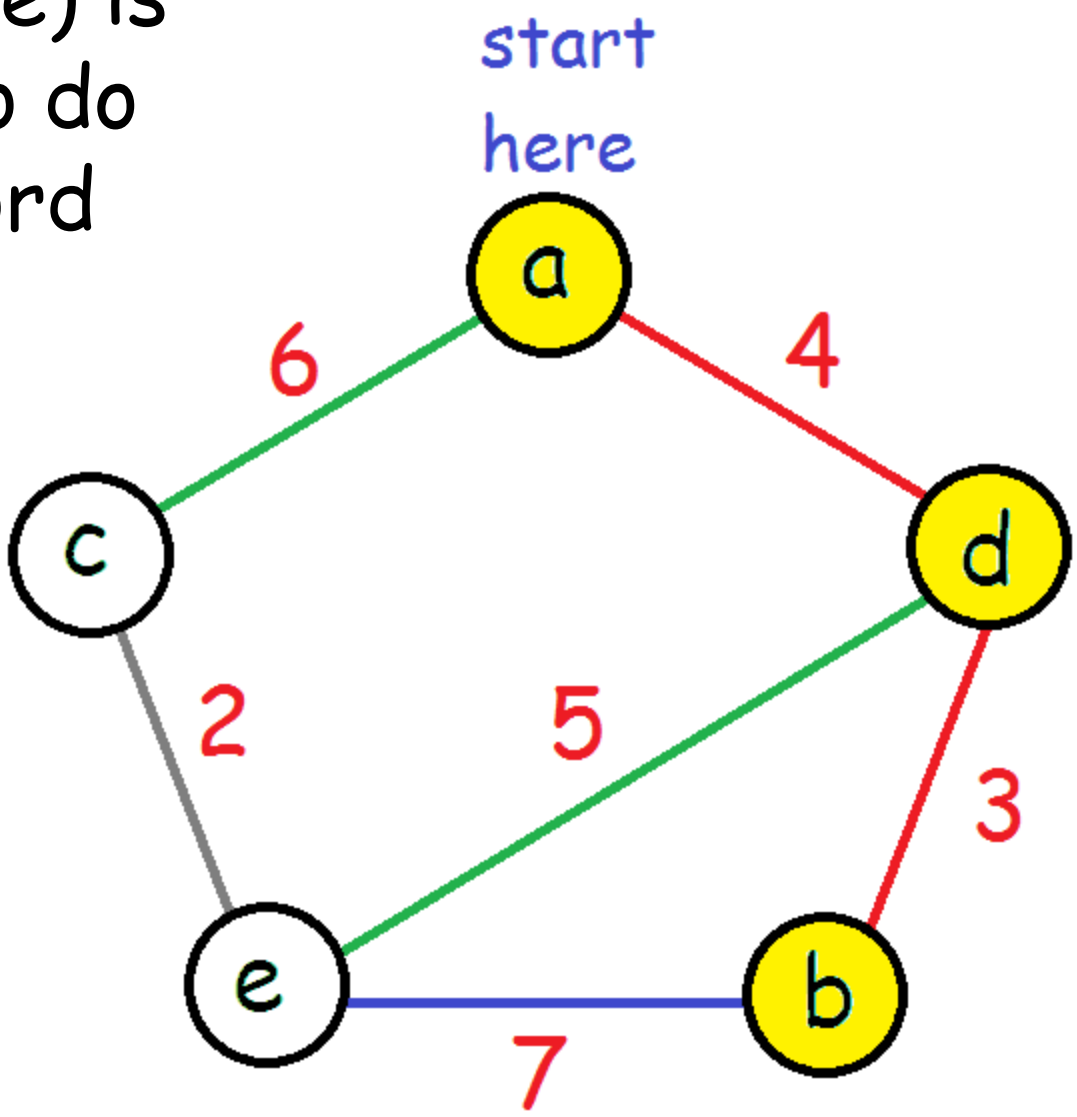
Put vertex b into the tree



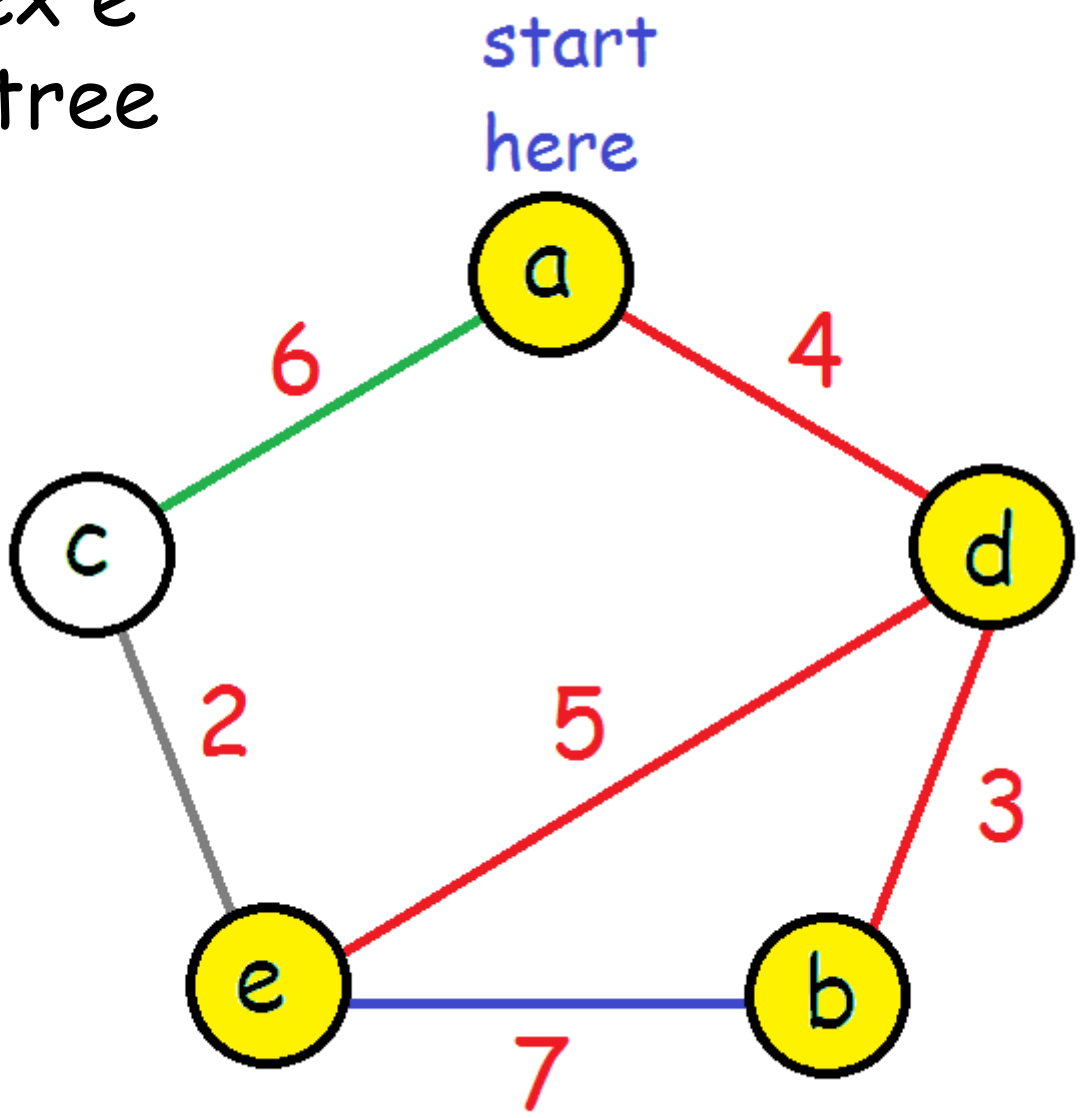
Consider edges next to b and record if less than min wt edges to those vertices



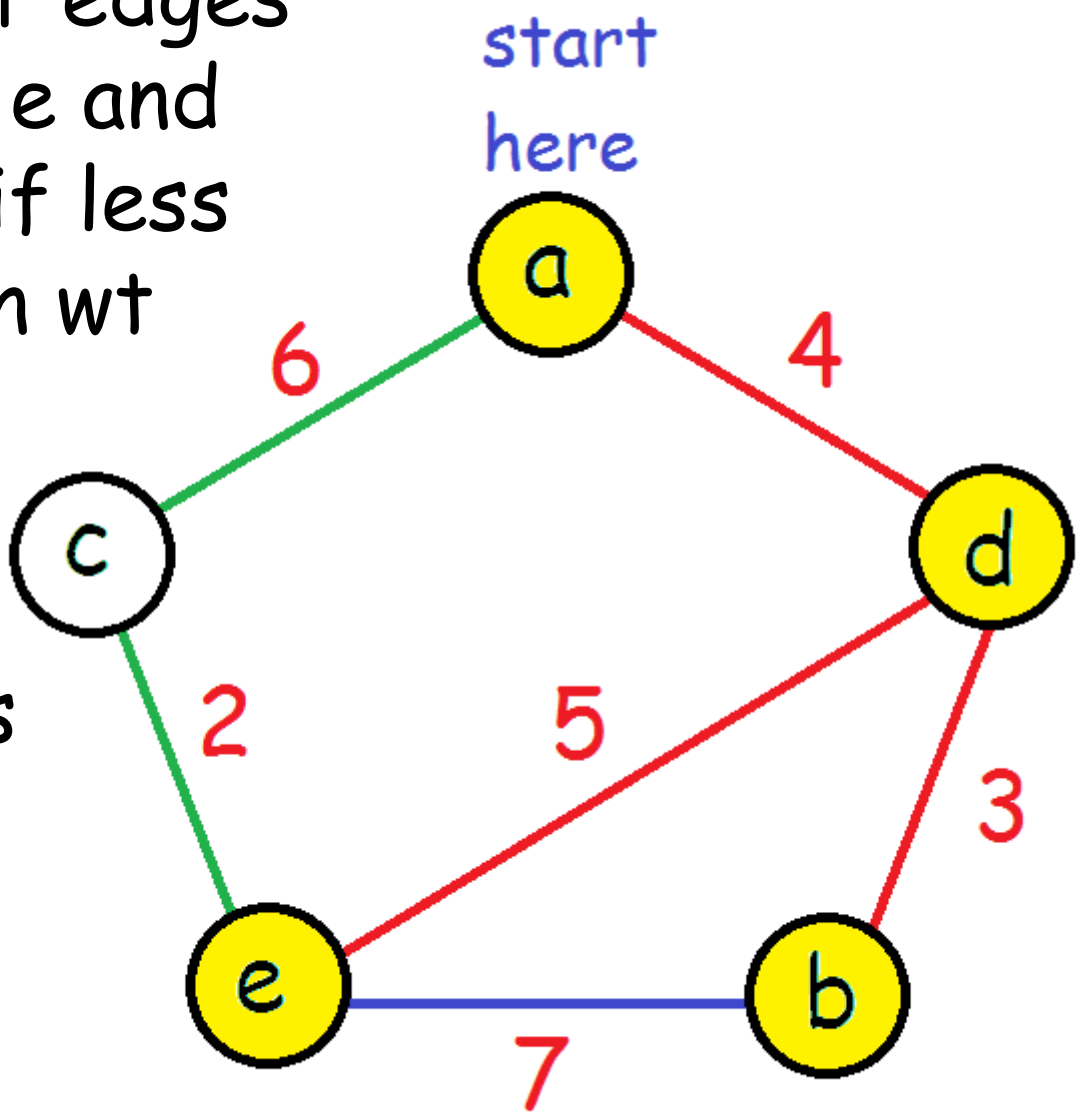
Edge (b,e) is worse so do not record it.



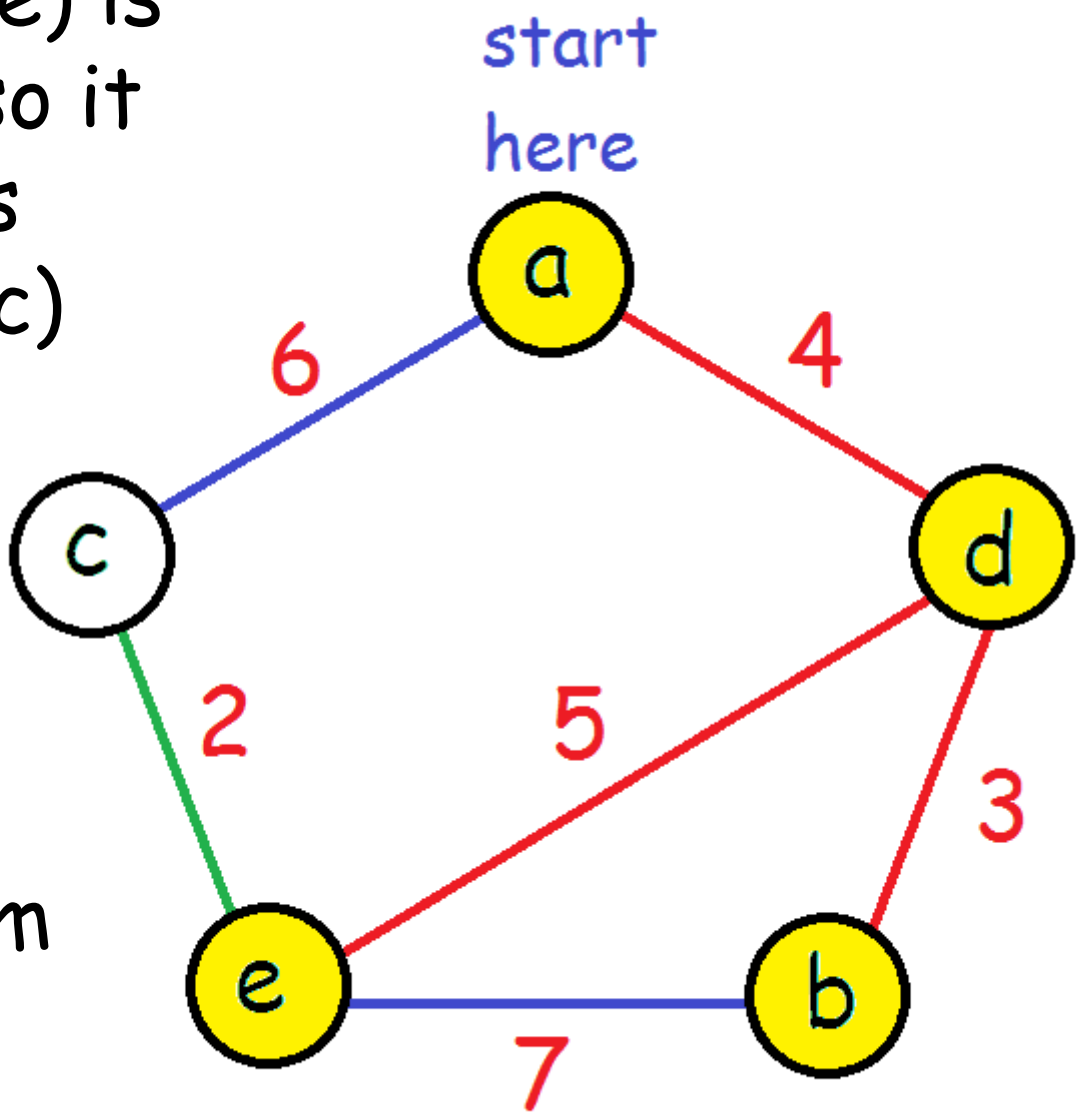
Put vertex e into the tree



Consider edges next to e and record if less than min wt edges to those vertices

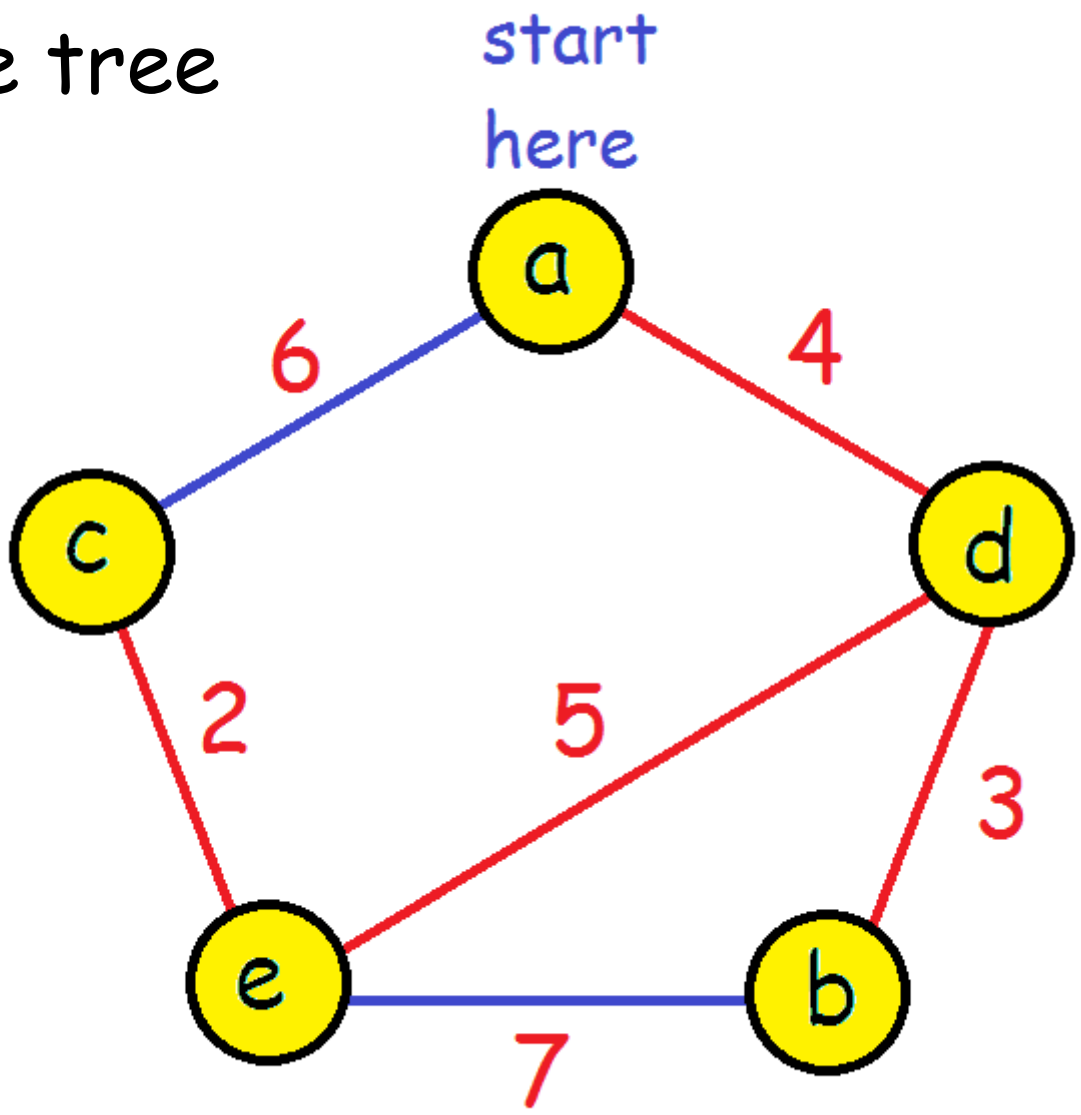


Edge (c,e) is better so it Replaces edge (a,c) as the best option for getting to c from a tree vertex



Put vertex e into the tree

MST:



High level description:

Put a vertex in the tree.

Repeat

1. Let e be a minimum weight edge from a tree vertex to a vertex not in the tree.

2. Add e to T and update the data structures.

Until n-1 edges have been selected.

Time complexity: Initialization: $\Theta(n)$

The outer loop is repeated n-1 times.

Step 1: $\Theta(n)$ at each iteration for a total of $\Theta(n^2)$.

Step 2: takes no more than $O(m)$ time in total (amortized analysis) if we use adjacency lists or $O(n^2)$ time with adjacency matrices because the neighbours of a vertex are considered at most one time.

The total amount of work is in $\Theta(n^2)$.

Under the comparison model, Kruskal's algorithm is $\Theta(m \log_2 m)$ in the worst case.

Thought question:
When is the Dijkstra/Prim algorithm which is $\Theta(n^2)$ faster than Kruskal's algorithm which is $\Theta(m \log m)$?