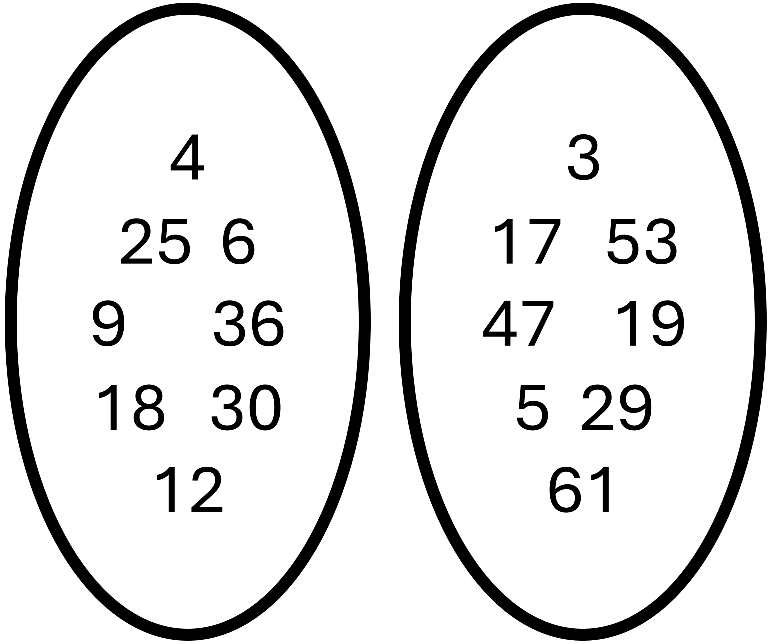
Minimum Spanning Trees

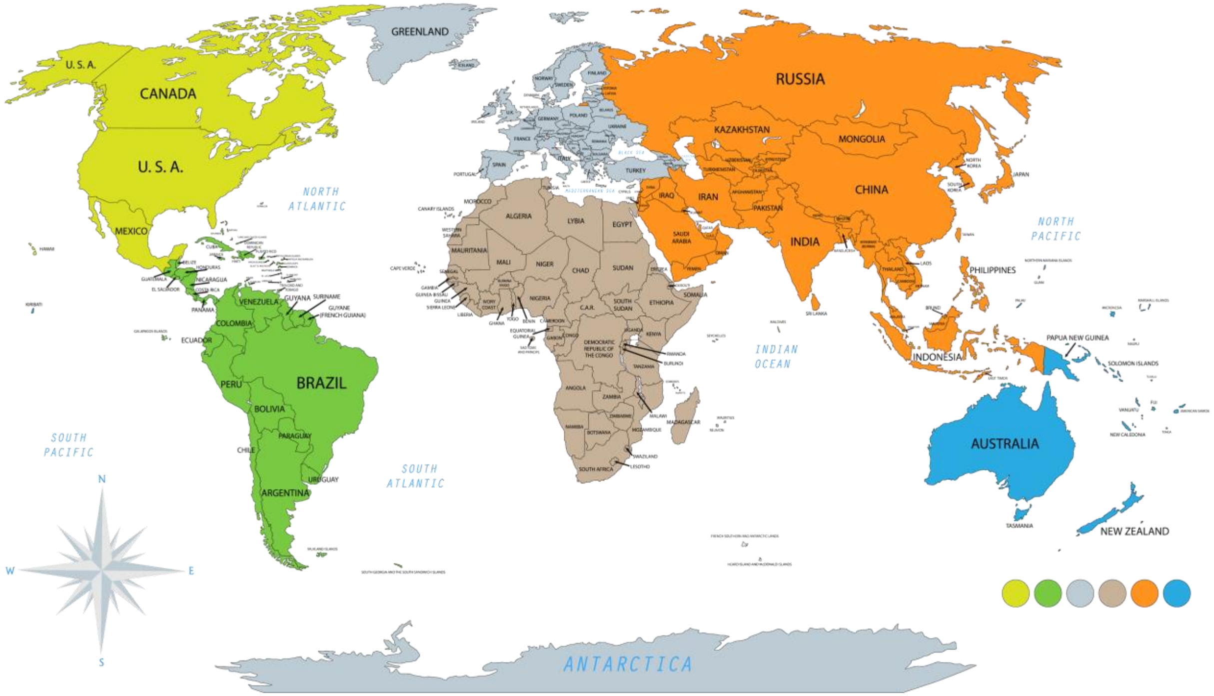
CS 2100: Program Design and Implementation 1

Northeastern University Khoury College of Computer Sciences

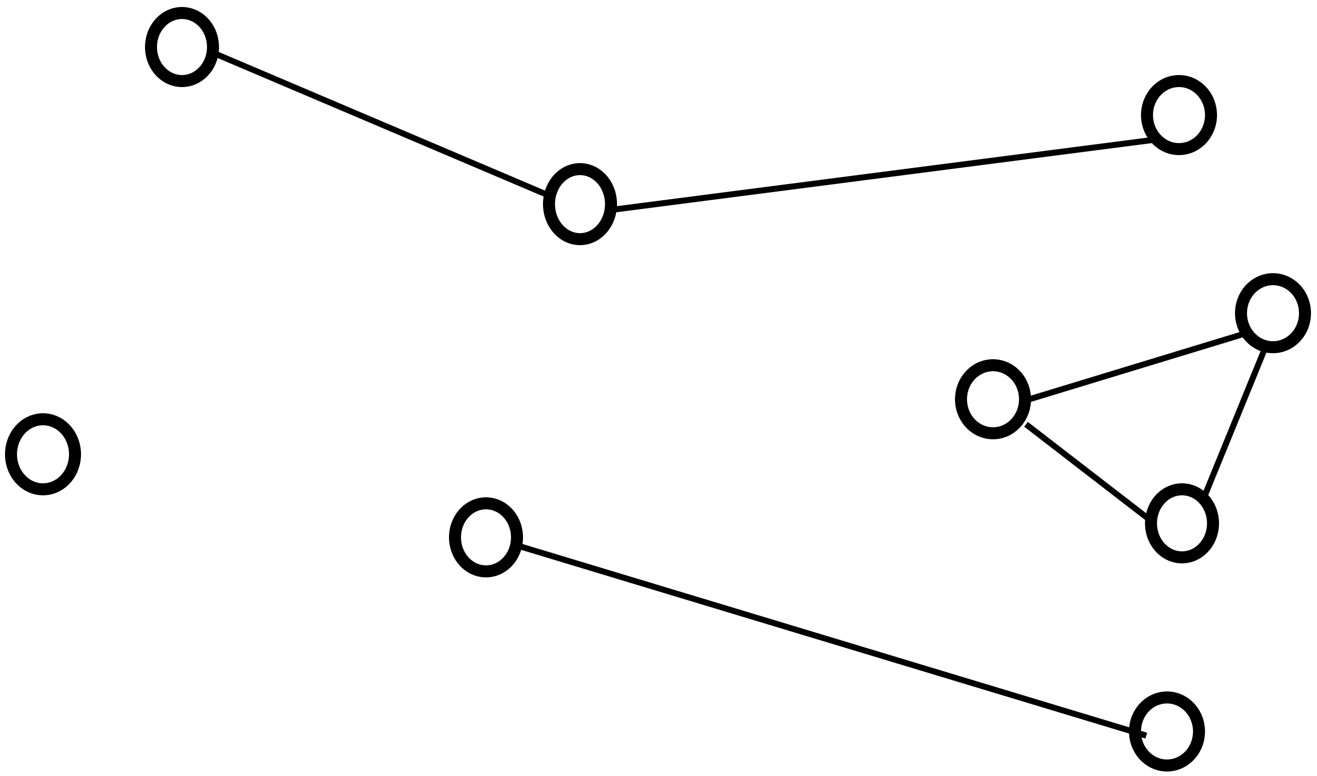
Review: The Union-Find algorithm finds disjoint sets.

Disjoint sets are sets that have no elements in common.



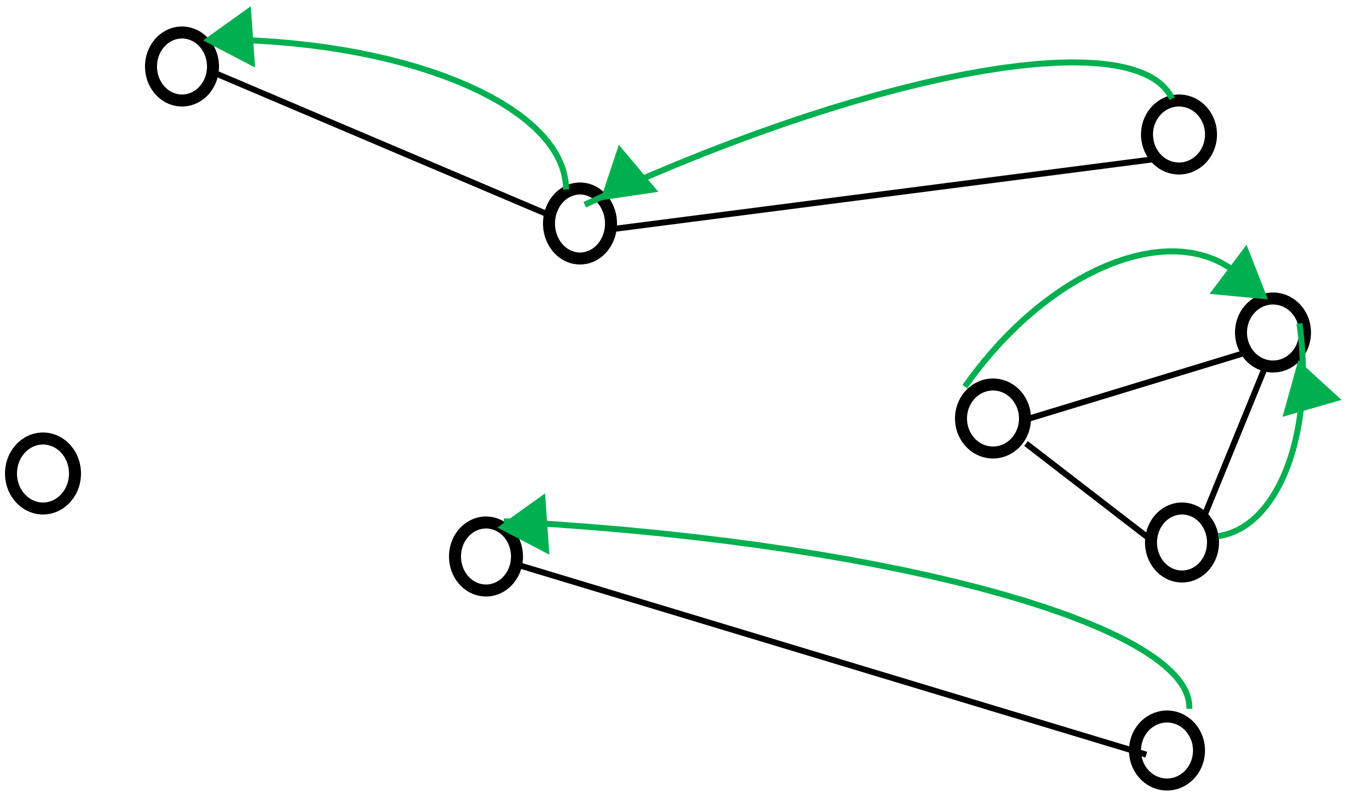


https://www.worldatlas.com/geography/continents-by-number-of-countries.html



Given: a bunch of items, and some requirements (edges) for which pairs of items need to be in the same set

Goal: put the items into disjoint sets



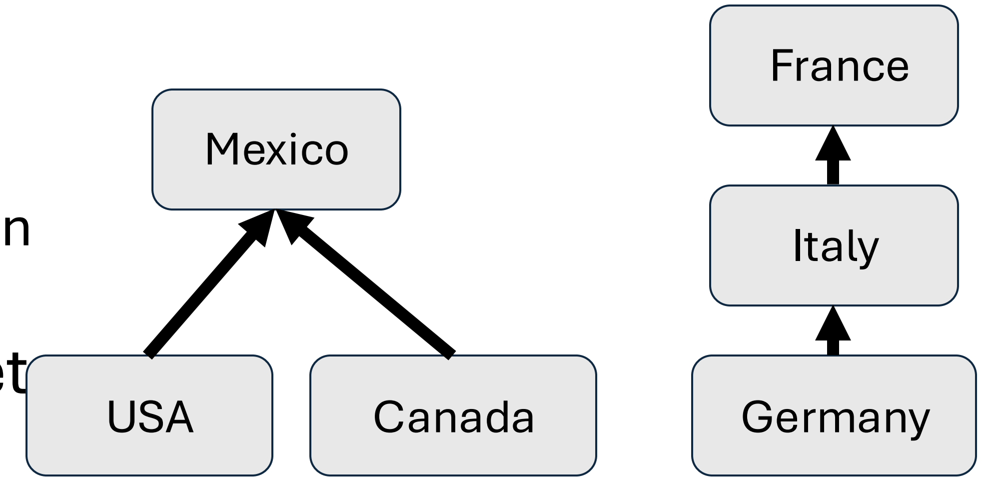
Given: a bunch of items, and some requirements (edges) for which pairs of items need to be in the same set

Goal: put the items into disjoint sets

Representing disjoint sets

- The goal of Union-Find is to result in a representation of disjoint sets:
  - Each item is a node in a graph
  - Each disjoint set is a tree within a *forest*
  - Each disjoint set has one element which is the *representative element*
    - That item is the root of the tree.

- The algorithm for finding / managing disjoint sets is called Union-Find
  - union: combine two sets
    - Do this when we find that those two sets have an element in common
  - find: find the representative element of a set
    - Given an element, find the root of the tree that it belongs to



Union-Find Summary

- The Union-Find algorithm finds disjoint sets
  - It groups nodes into sets such that, for each set:
    - The nodes in that set are connected to each other
    - None of the nodes in that set have an edge with a node outside that set

- Each node keeps track of:
  - Its data (the element)
  - Its parent (another node)
    - `node.parent == node` if node is a root
    - `node.parent == node2` if node2 is its parent
- Operations
  - `union(node1, node2)` combines the sets containing node1 and node2 into a single set
  - `find(node)` returns node's root

```
find (node):
    while (node.parent != node):
        node = node.parent
    return node

union(node1, node2):
  root1 = find(node1)
  root2 = find(node2)
  if root1 != root2:
    if root1.rank > root2.rank:
      root2.parent = root1
    else:
      root1.parent = root2
      if root1.rank == root2.rank:
        root2.rank++
```

New data structure: Priority Queue

Priority Queue = a list which stores things in order

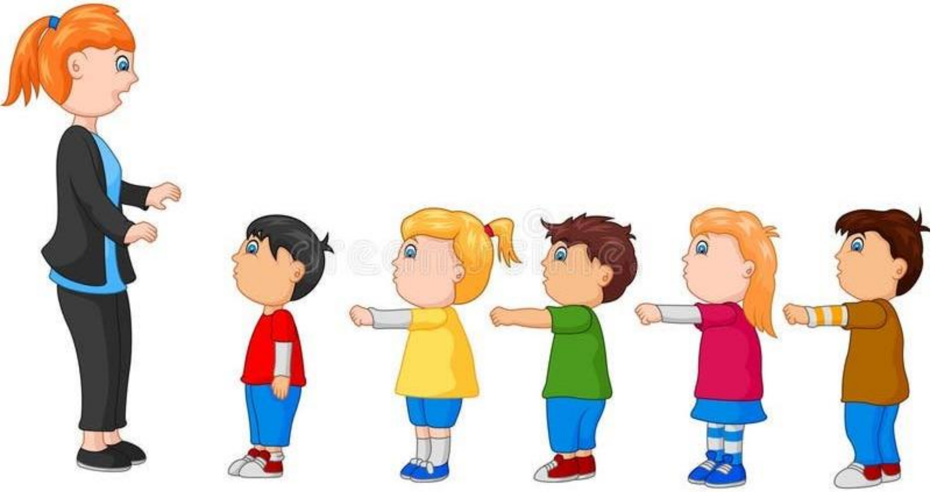
The order according to `__eq__()` / `__lt__()` / etc., not the order in which they were added

```
import heapq

priority_queue: list[int] = []
print(priority_queue) # []

heapq.heappush(priority_queue, 3)
heapq.heappush(priority_queue, 1)
heapq.heappush(priority_queue, 4)
heapq.heappush(priority_queue, 2)

print(priority_queue) # [1, 2, 4, 3]
```



https://www.dreamstime.com/illustration/standing-line.html

Poll: What is output?

```
priority_queue: list[int] = []

heapq.heappush(priority_queue, 5)
heapq.heappush(priority_queue, 3)
heapq.heappush(priority_queue, 6)

print(heapq.heappop(priority_queue))
```

A. 3
B. 5
C. 6

Poll: A tree is a type of graph. What are the differences between a tree and a graph?

A. Each node in a tree must have exactly one parent (unless it is the root)

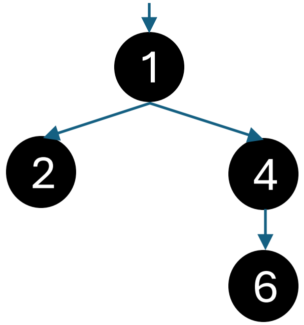
B. Each node in a tree must have at least one child

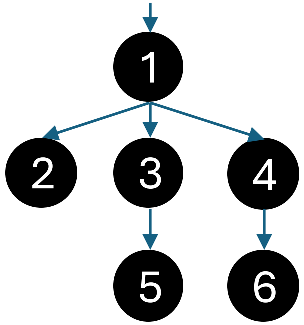
C. Trees are required to have at least one cycle

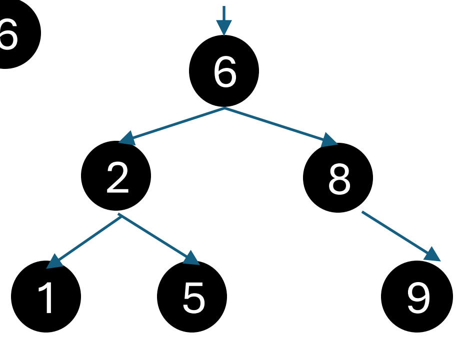
D. Trees are not allowed to have cycles

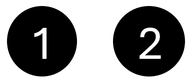
E. Edges in a tree must be weighted

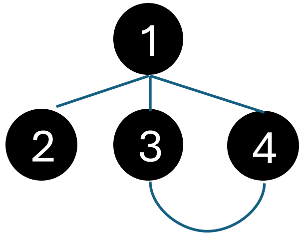
F. Edges in a tree cannot be weighted

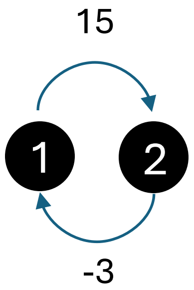








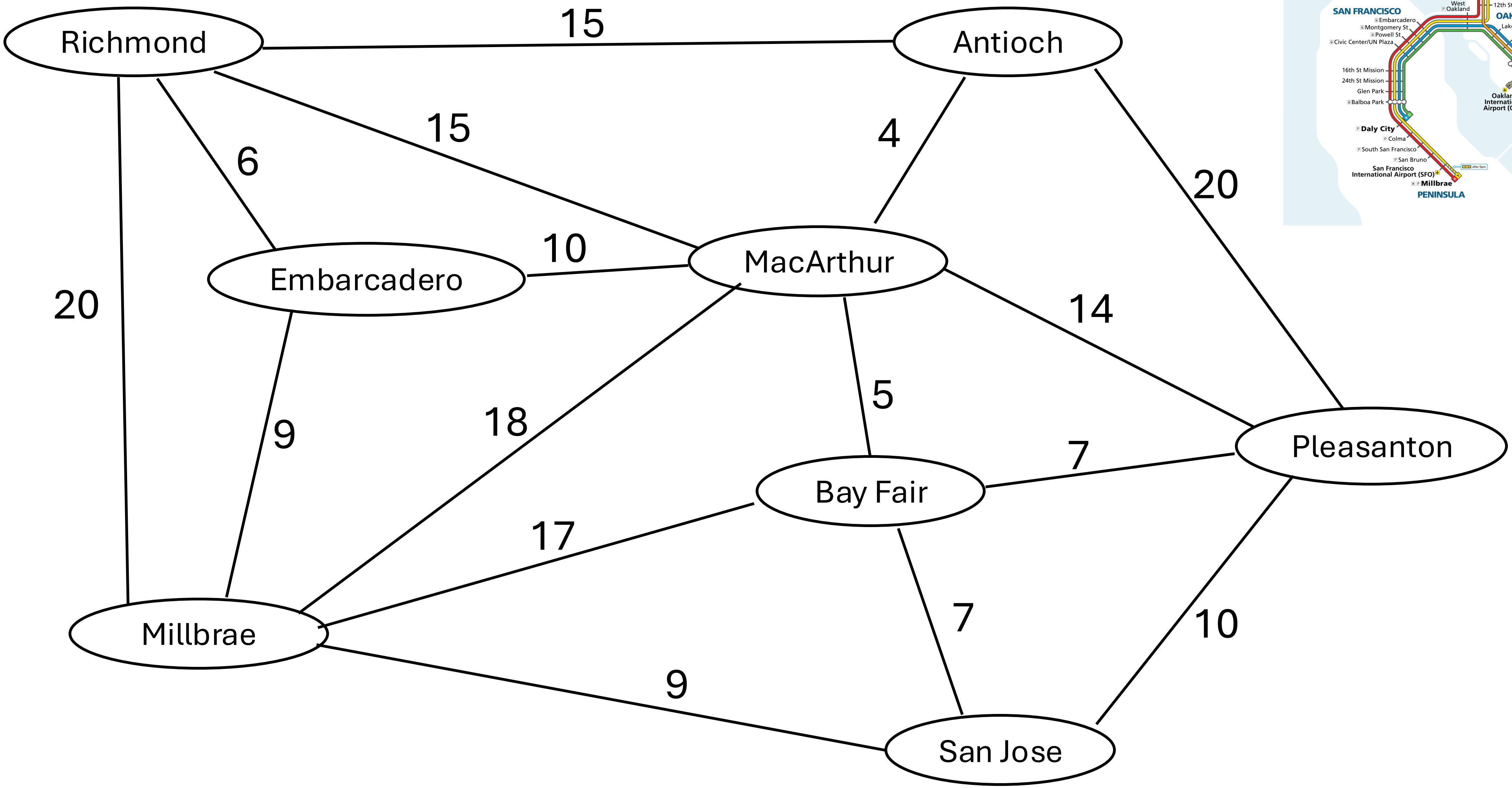




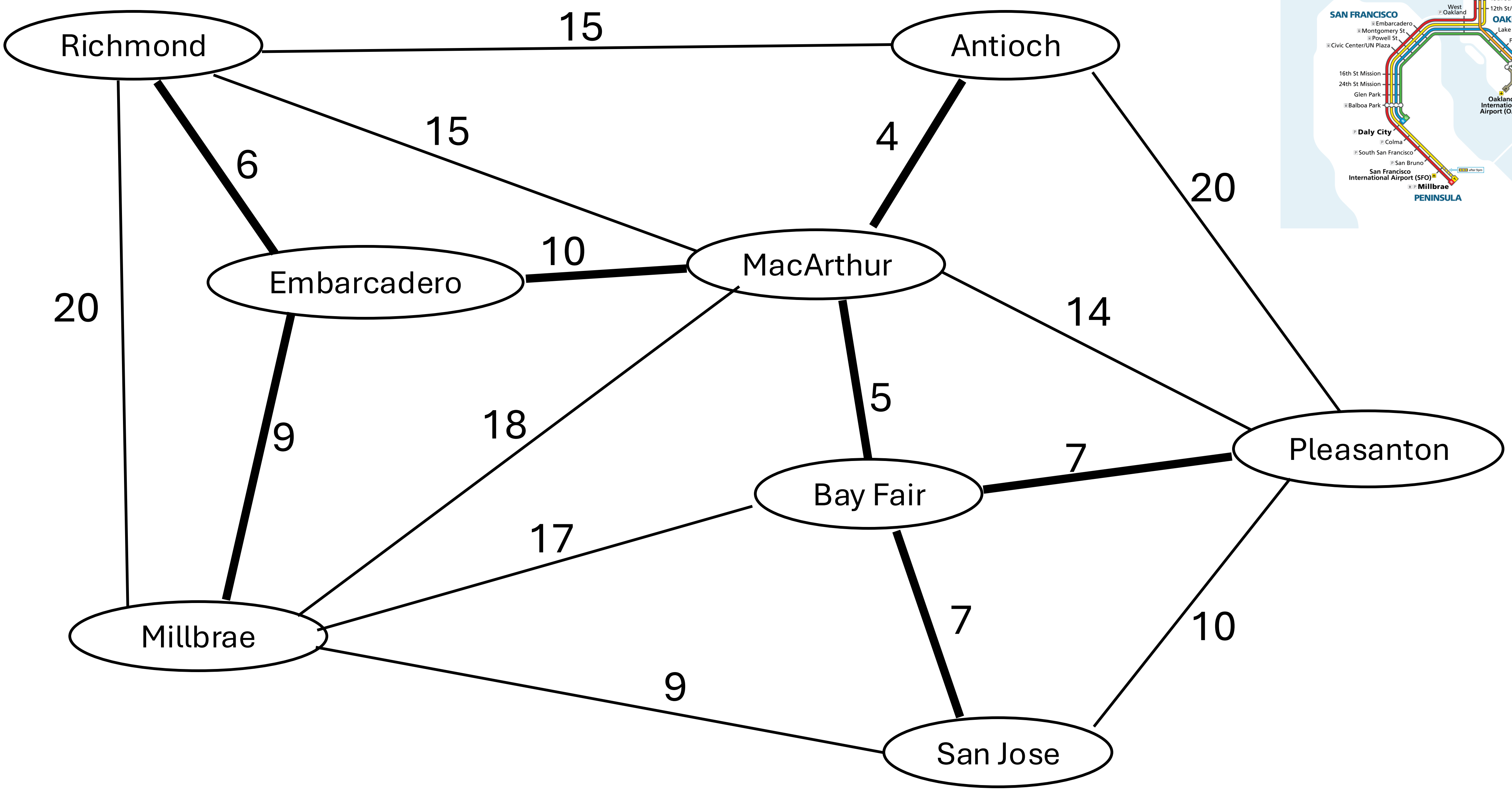
Let's say we're designing the BART system...

- Given:
  - Location of each station
  - Cost of building tracks from each station to each other station
- Goal:
  - Find the cheapest way to build tracks such that we can get from any station to any other station

https://www.bart.gov/system-map


Richmond
Antioch
Embarcadero
MacArthur
Pleasanton
Bay Fair
Millbrae
San Jose
15
15
6
4
20
10
20
14
9
18
5
7
17
7
10
9

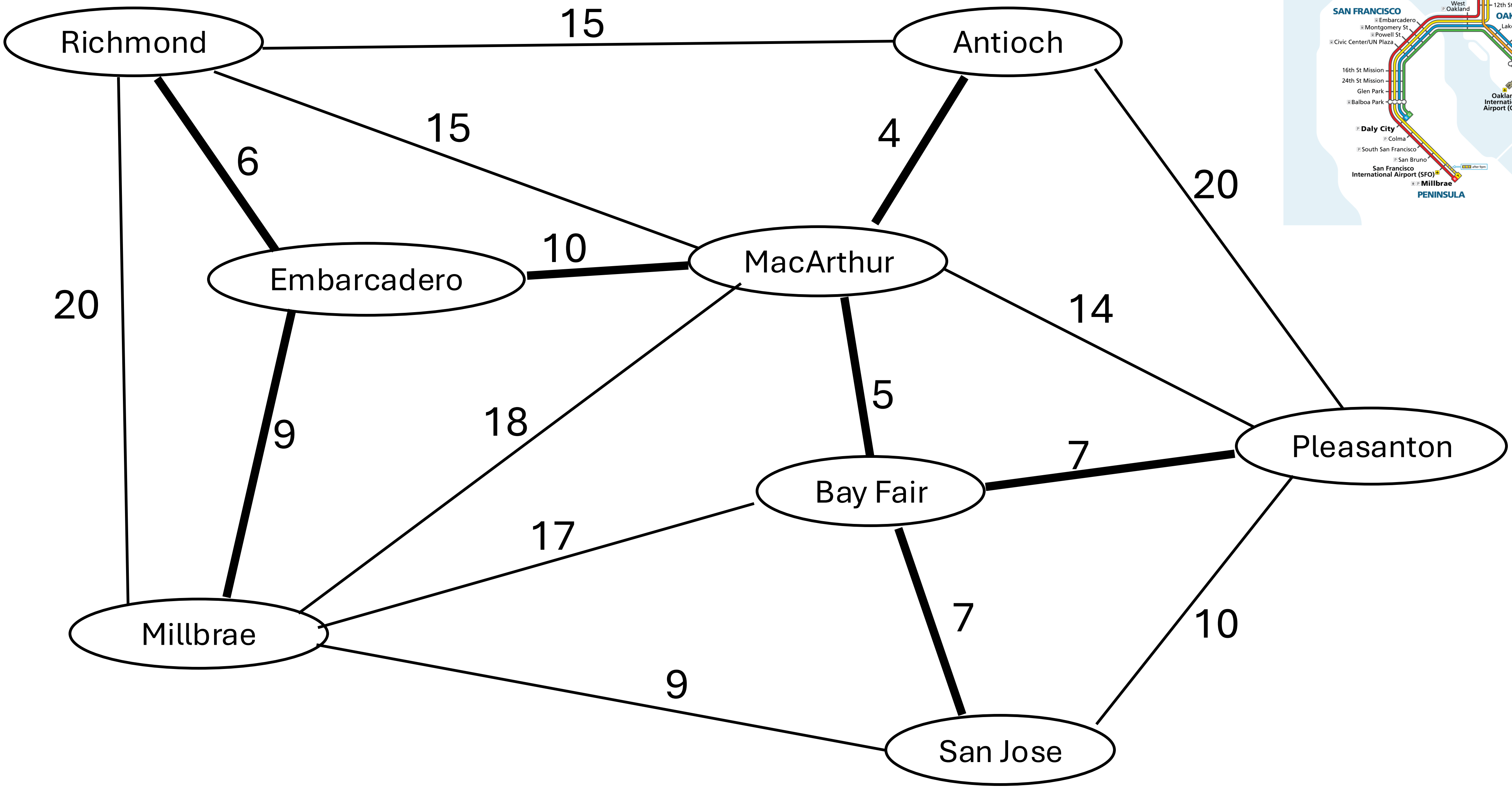

Richmond
Antioch
EAST BAY
SAN FRANCISCO
OAKLAND
PENINSULA
SAN JOSE
Daytime Service
Red Line
Orange Line
Yellow Line
Green Line
Blue Line
OAK Airport
Embarcadero
MacArthur
Bay Fair
Millbrae
Dublin/
Pleasanton
Berryessa/North San José


Richmond
Antioch
Embarcadero
MacArthur
Pleasanton
Bay Fair
Millbrae
San Jose
15
15
6
4
20
10
20
14
9
18
5
7
17
7
10
9

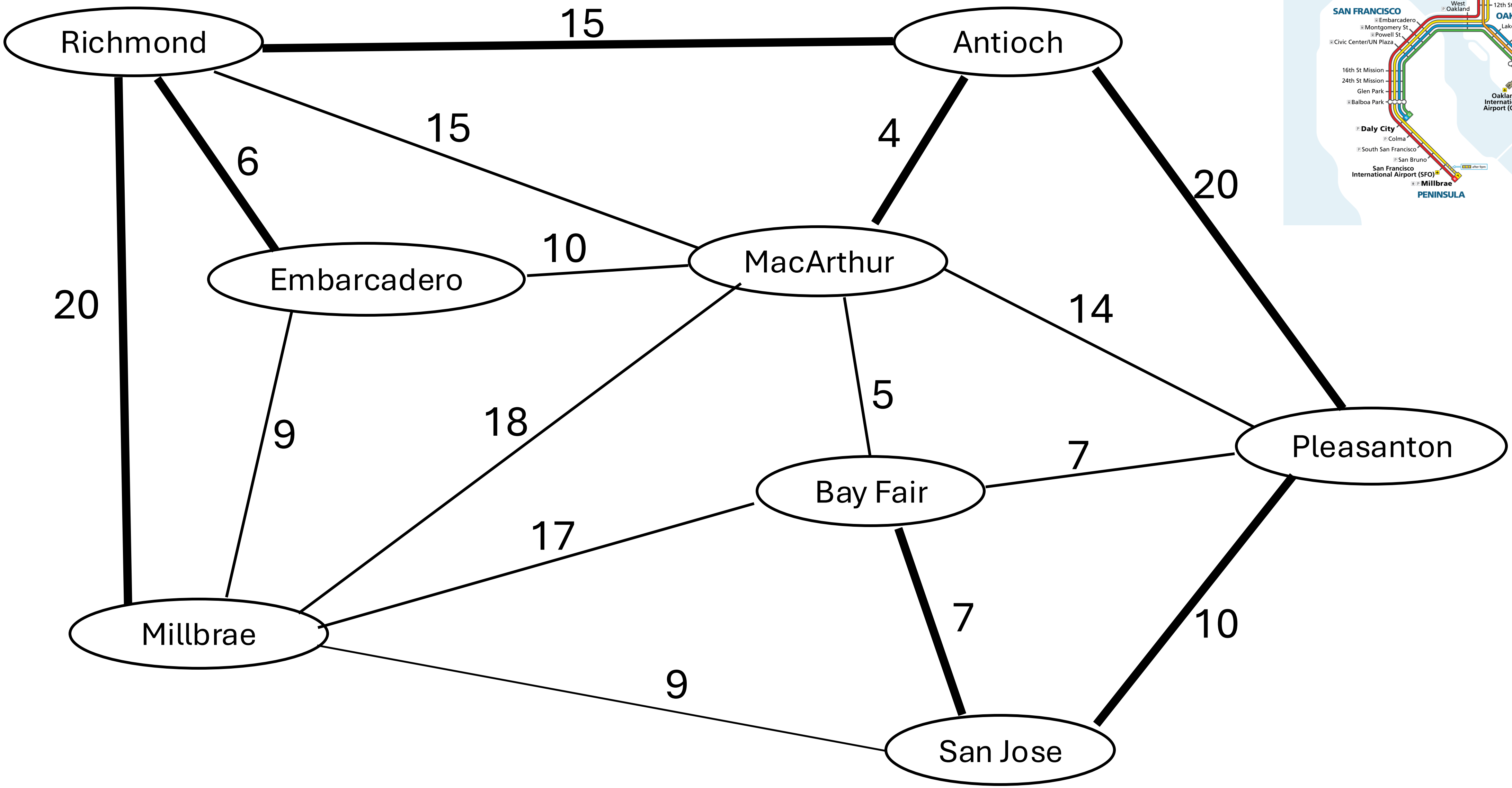

North Concord/ Martinez
Pittsburg/ Bay Point
Pittsburg Center
Antioch
Change trains at transfer platform
Richmond
El Cerrito del Norte
El Cerrito Plaza
North Berkeley
Downtown Berkeley
Ashby
Concord
Pleasant Hill/Contra Costa Centre
Walnut Creek
Lafayette
Orinda
Rockridge
EAST BAY
Daytime Service (Every day until 9pm)
Red Line Richmond - Millbrae
Orange Line Richmond - Berryessa/North San José
Yellow Line Antioch - SFO Int'l Airport
Green Line Daly City - Berryessa/North San José
Blue Line Daly City - Dublin/Pleasanton
OAK Airport Coliseum - Oakland Int'l Airport
MacArthur TIMED TRANSFER (SOUTHBOUND)
19th St/Oakland TIMED TRANSFER (NORTHBOUND)
12th St/Oakland City Center
West Oakland
SAN FRANCISCO
OAKLAND
Embarcadero
Montgomery St
Powell St
Civic Center/UN Plaza
Lake Merritt
Fruitvale
Coliseum
San Leandro
Bay Fair
Castro Valley
West Dublin/ Pleasanton
Dublin/ Pleasanton
16th St Mission
24th St Mission
Glen Park
Balboa Park
Oakland International Airport (OAK)
Hayward
South Hayward
Union City
Fremont
Daly City
Colma
South San Francisco
San Bruno
San Francisco International Airport (SFO)
after 9pm
Millbrae
PENINSULA
Warm Springs/South Fremont
Milpitas
Berryessa/North San José
SAN JOSE

Spanning tree

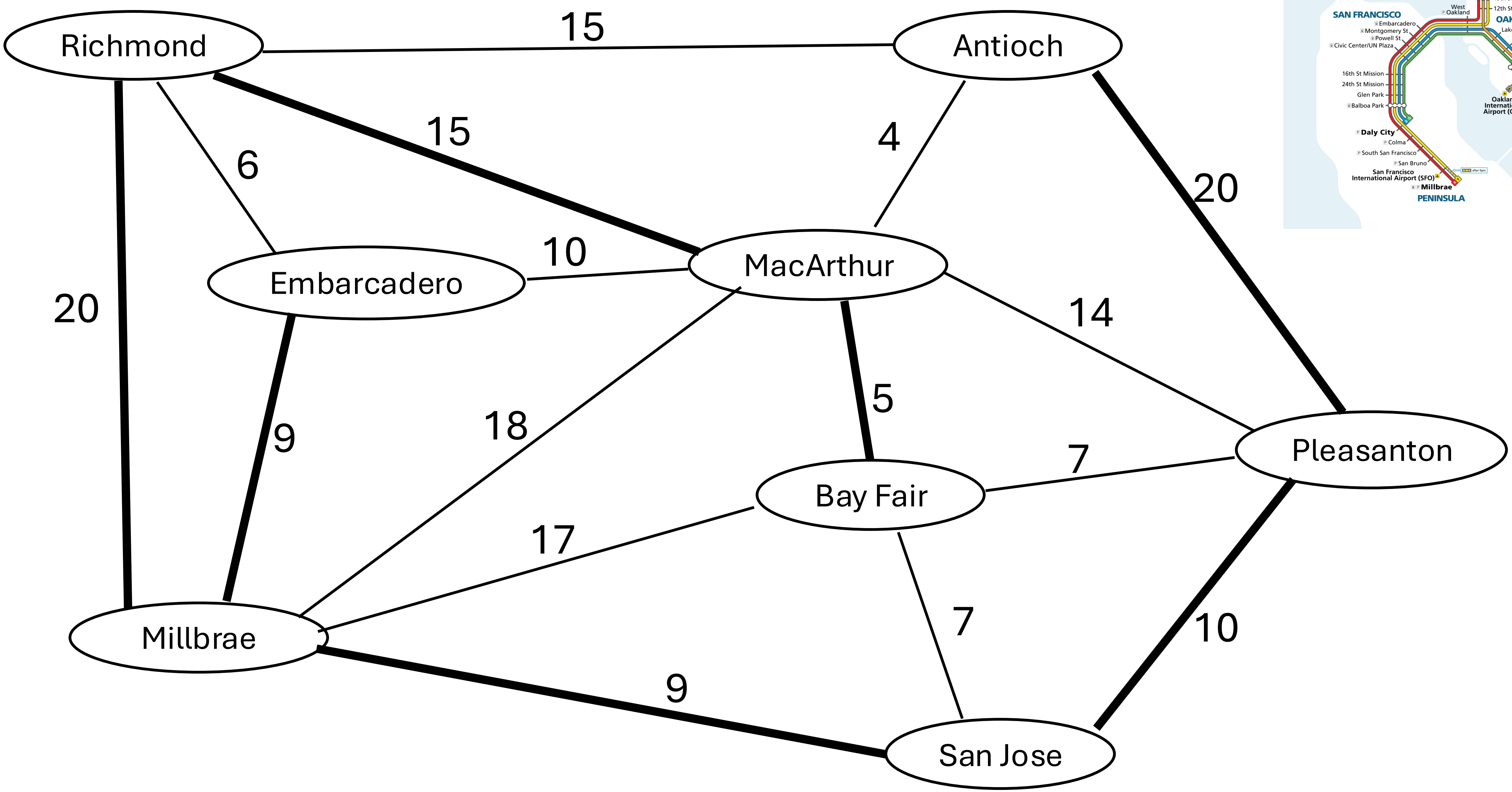
A spanning tree of a graph G is any tree that contains all the vertices in G


Richmond
Antioch
Embarcadero
MacArthur
Pleasanton
Bay Fair
Millbrae
San Jose
15
15
6
4
20
10
20
14
5
18
9
7
17
7
10
9


Richmond
Antioch
EAST BAY
SAN FRANCISCO
OAKLAND
PENINSULA
SAN JOSE
Millbrae
Dublin/Pleasanton
Berryessa/North San José
Daytime Service
Red Line
Orange Line
Yellow Line
Green Line
Blue Line
OAK Airport


Richmond
Antioch
Embarcadero
MacArthur
Pleasanton
Bay Fair
Millbrae
San Jose
15
15
6
4
20
10
20
14
18
5
9
7
17
7
10
9


EAST BAY
SAN FRANCISCO
OAKLAND
PENINSULA
SAN JOSE
Daytime Service


Richmond
Antioch
Embarcadero
MacArthur
Pleasanton
Bay Fair
Millbrae
San Jose
15
15
6
4
20
10
20
14
5
9
18
7
17
7
10
9


Richmond
Antioch
EAST BAY
SAN FRANCISCO
OAKLAND
PENINSULA
SAN JOSE
Daytime Service
Red Line
Orange Line
Yellow Line
Green Line
Blue Line
OAK Airport
MacArthur
Embarcadero
Bay Fair
Daly City
Millbrae
Dublin/ Pleasanton
Berryessa/North San José

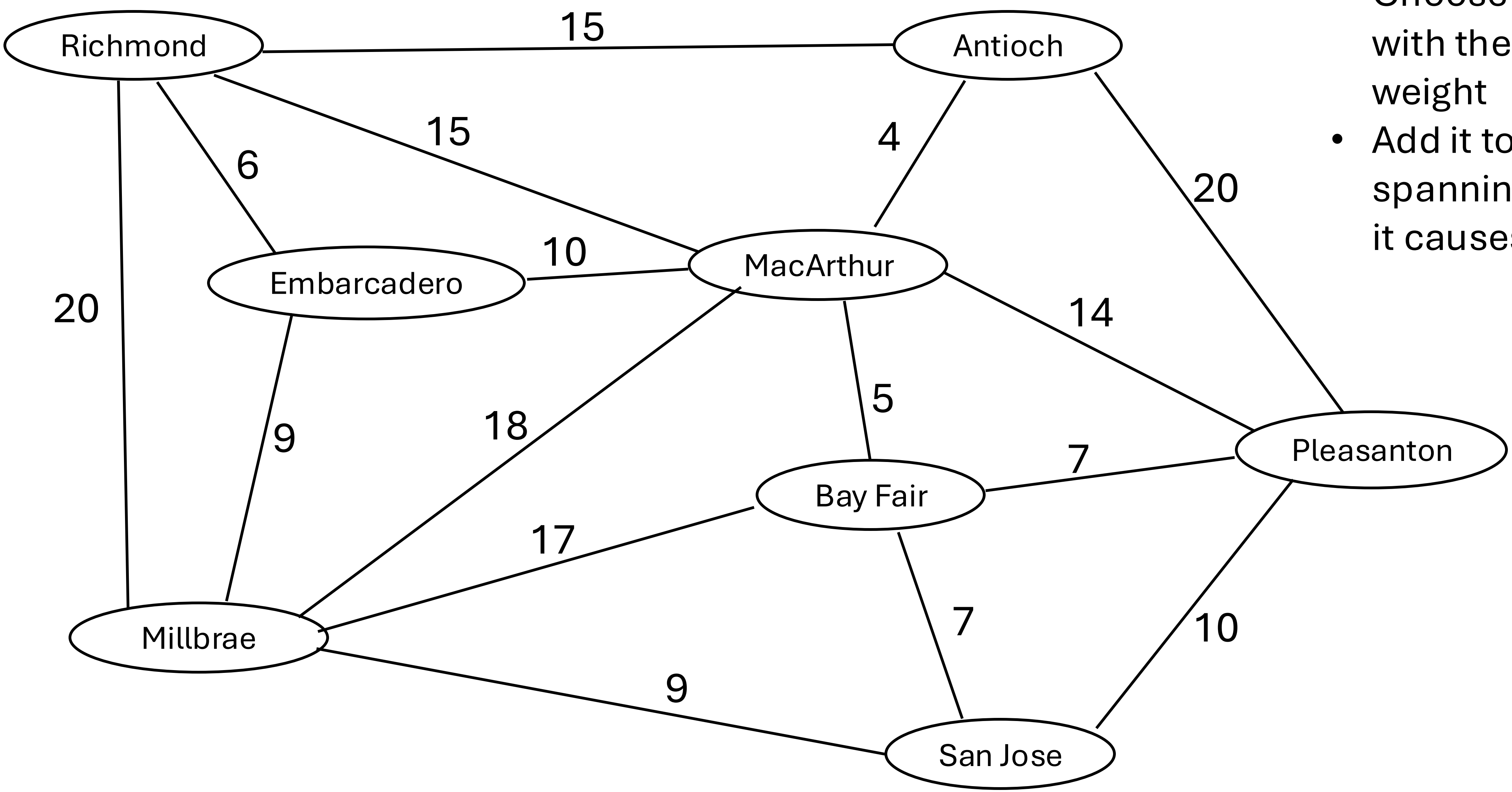
Minimum Spanning Tree

An MST of a graph G is a tree that connects all the vertices of G with the minimum possible edge weight sum

Kruskal's Algorithm

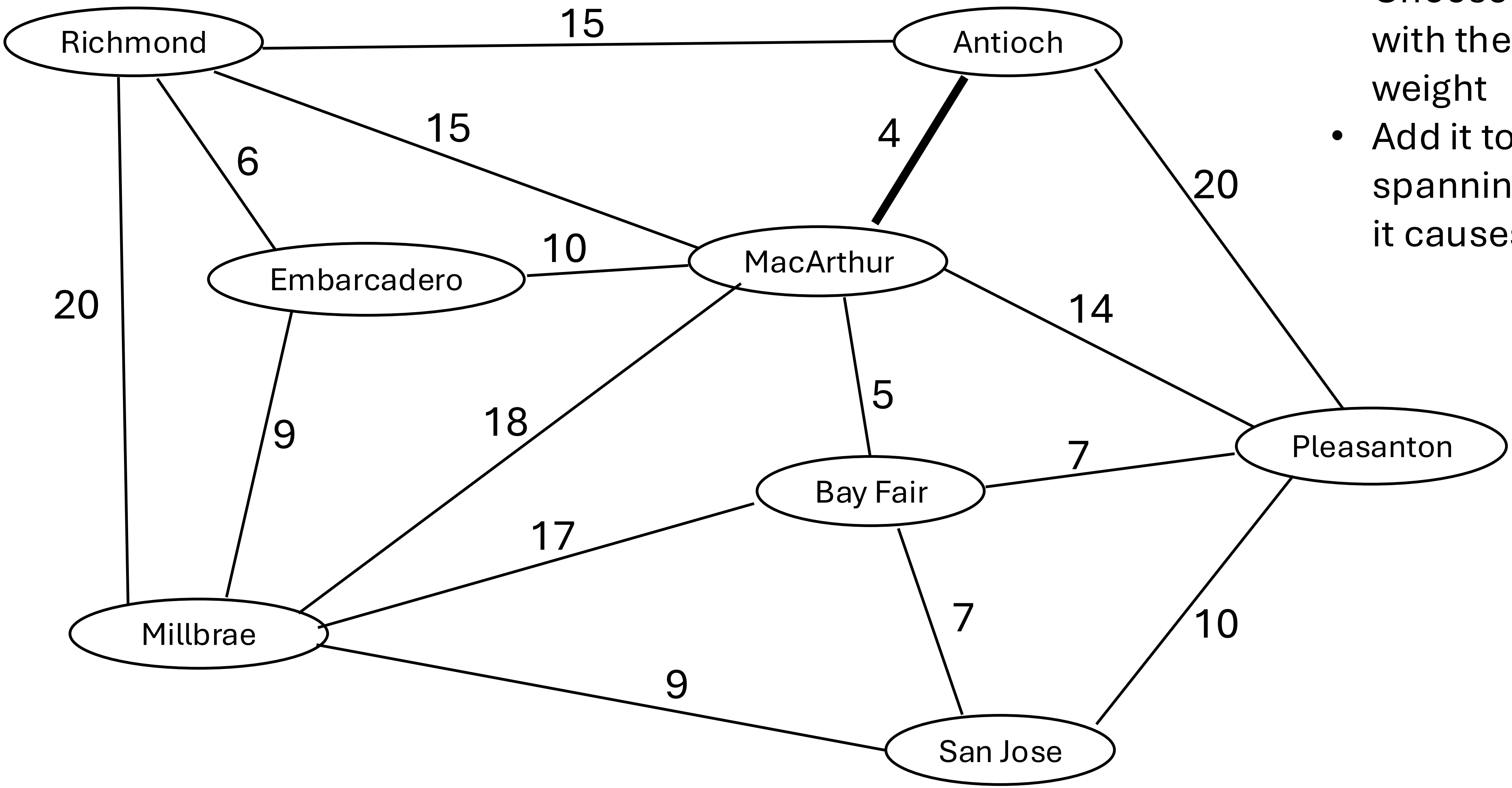
- Until all the vertices are connected:
  - Choose the edge with the minimum weight
  - Add it to the spanning tree unless it causes a cycle

- This algorithm is:
  - Greedy
  - Optimal



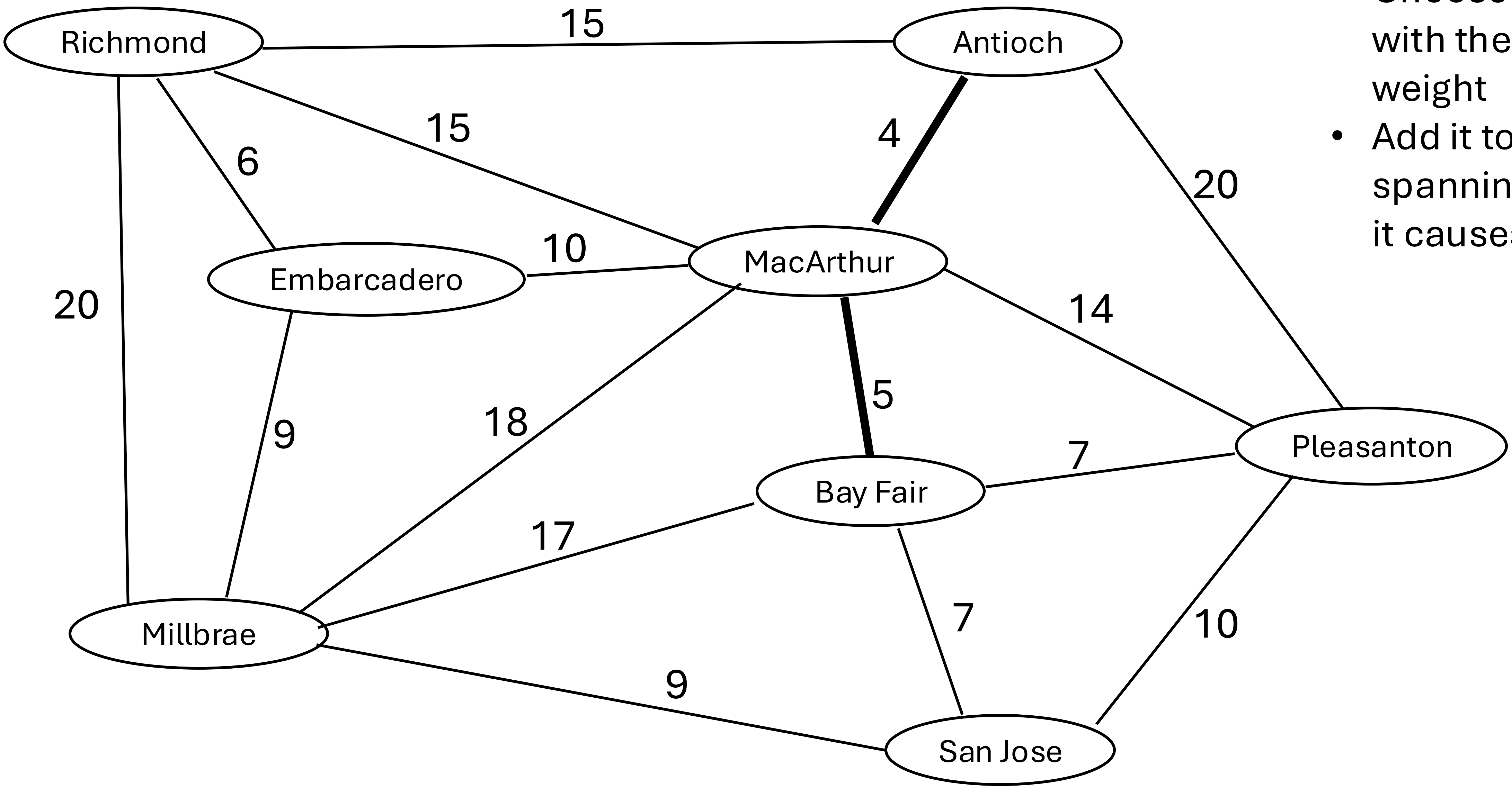
Until all the vertices are connected:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle



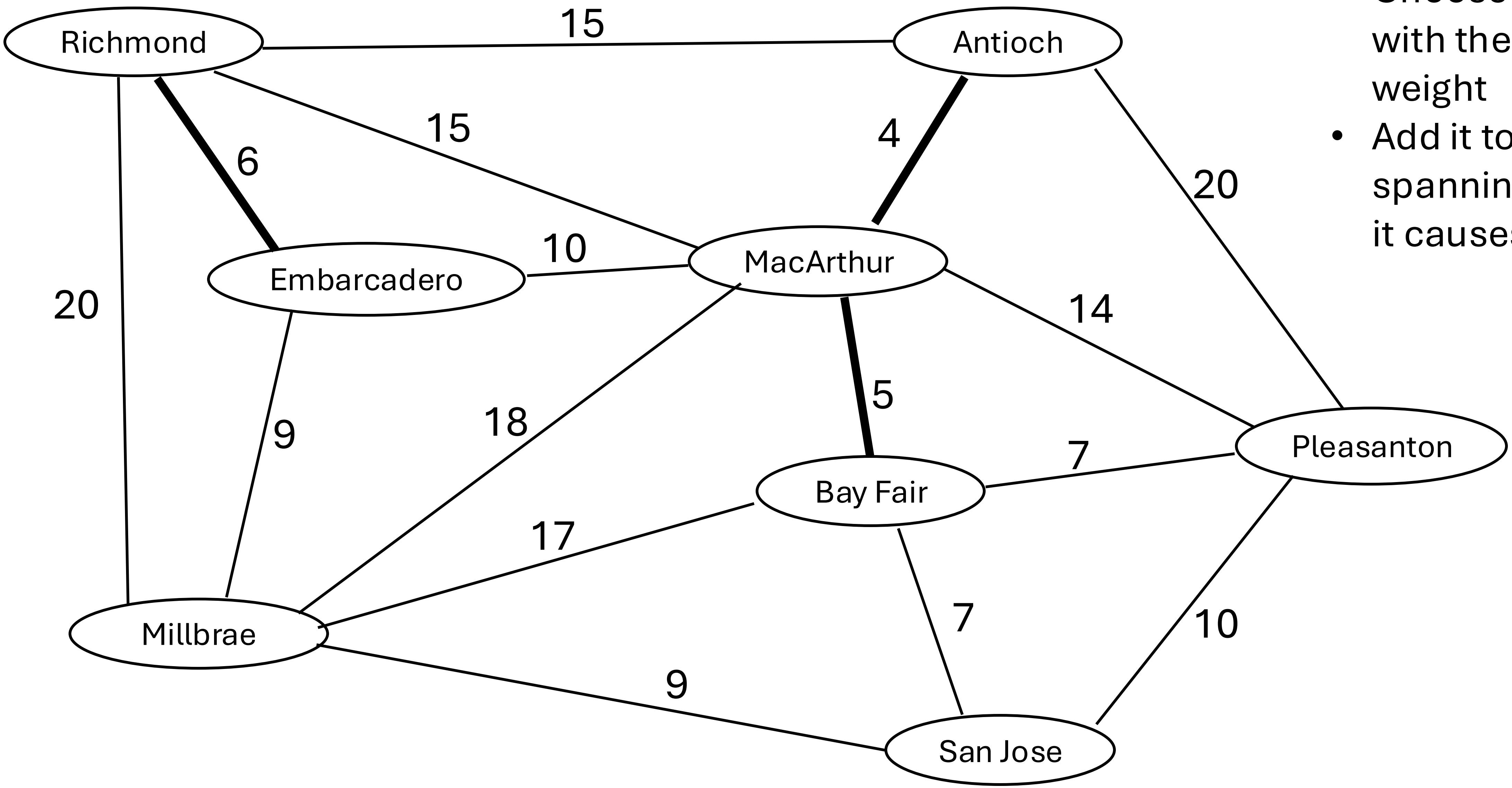
Until all the vertices are connected:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle



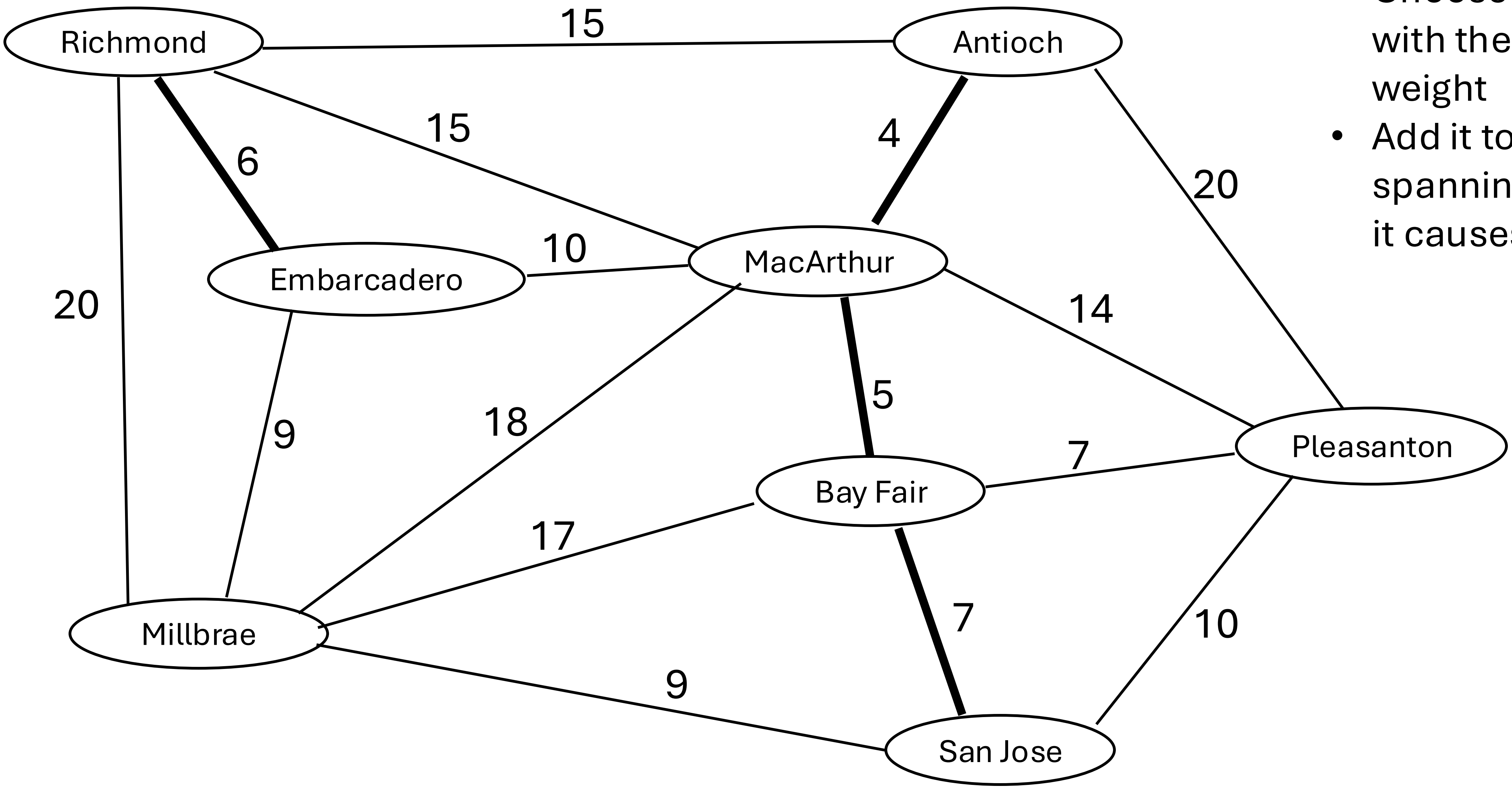
Until all the vertices are connected:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle



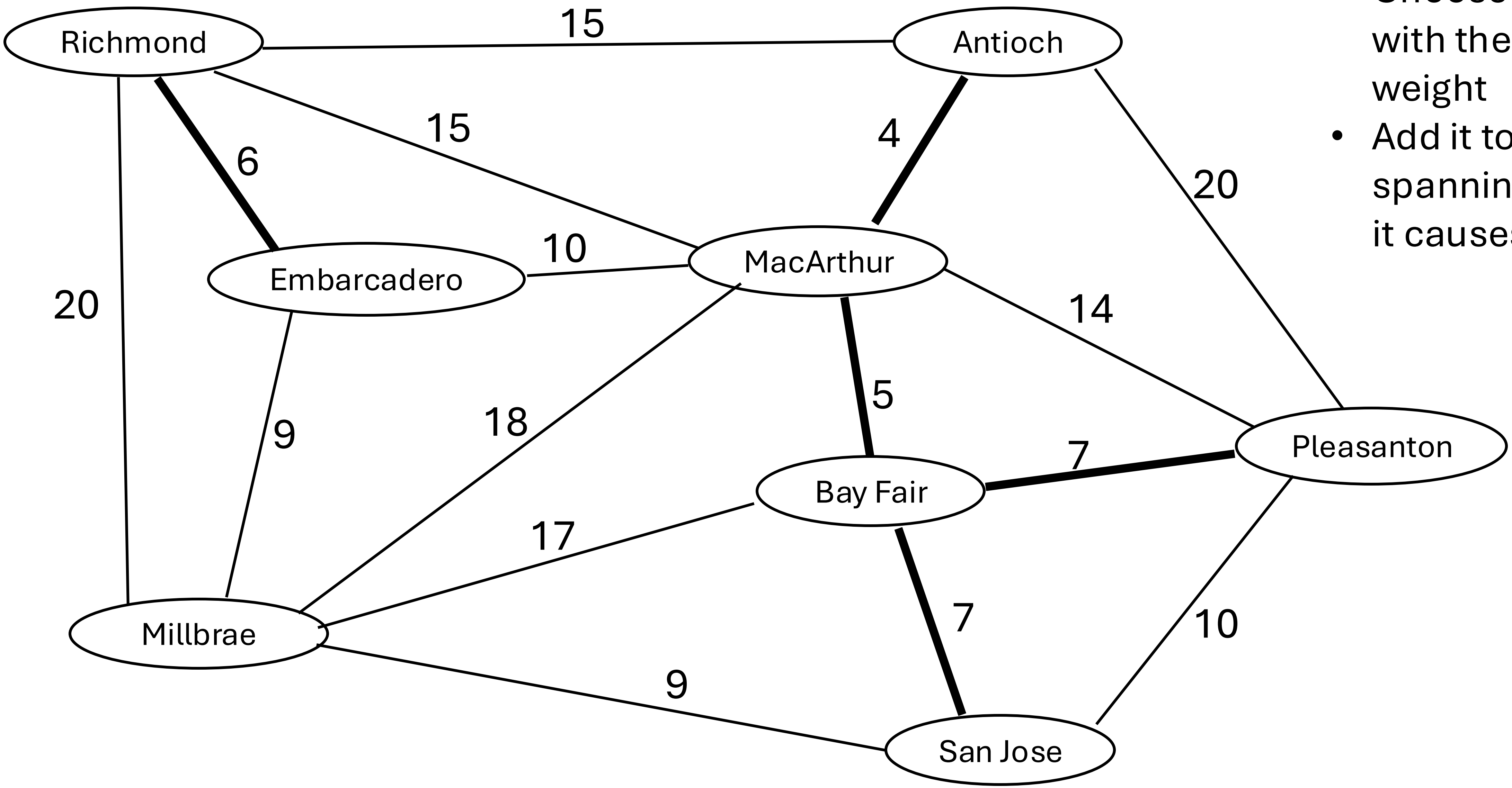
Until all the vertices are connected:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle



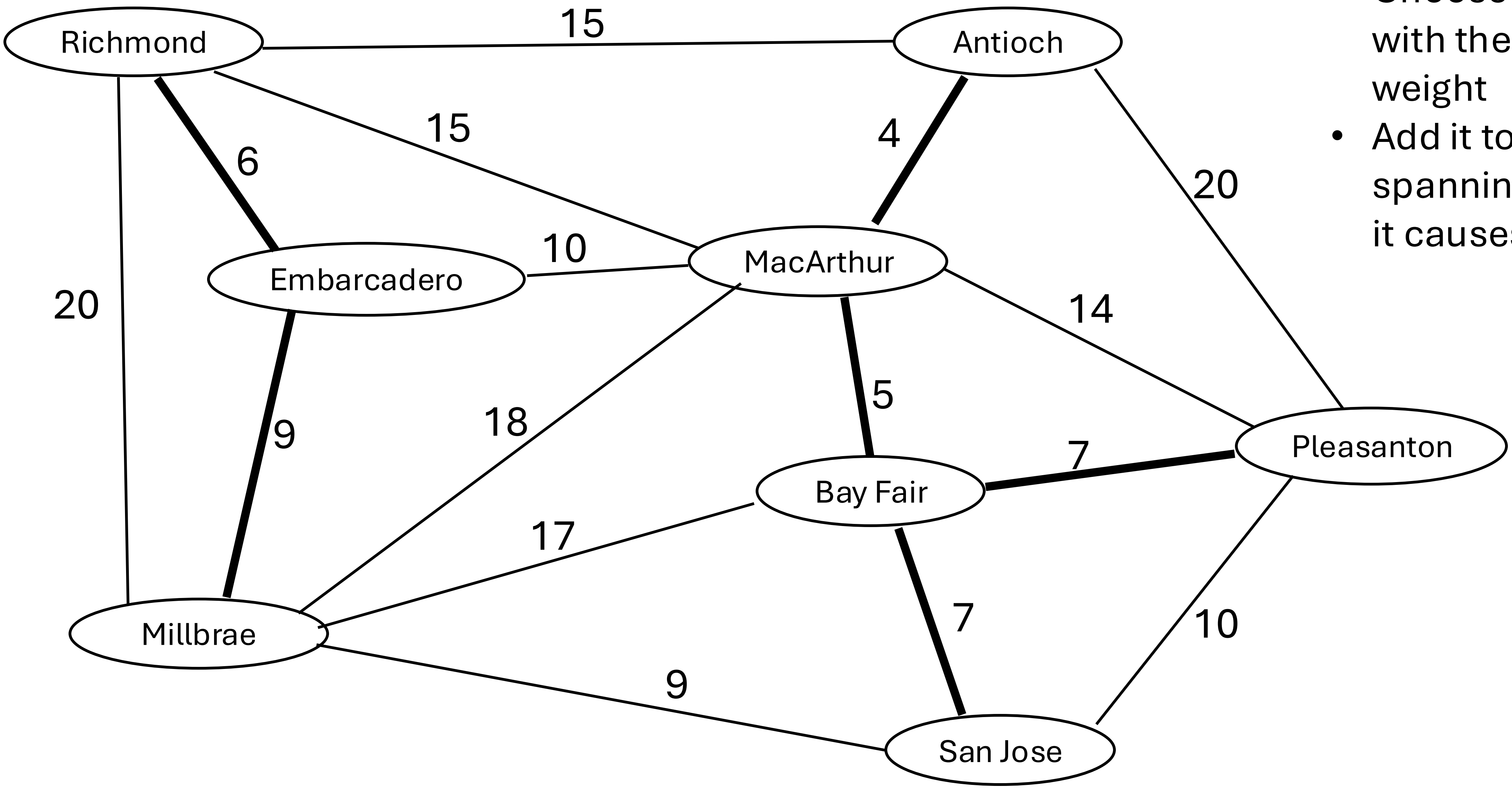
Until all the vertices are connected:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle



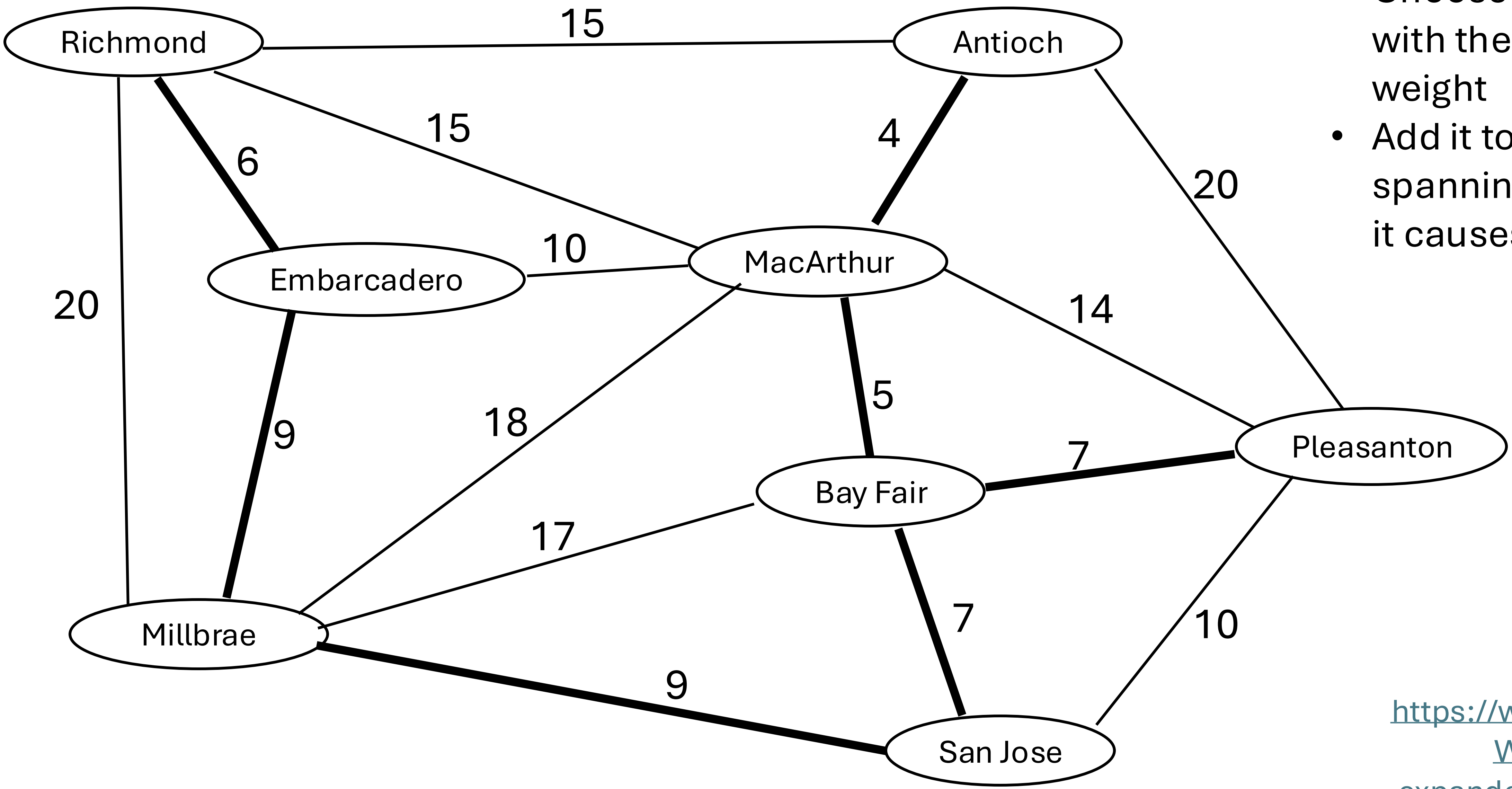
Until all the vertices are connected:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle



Until all the vertices are connected:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle



Until all the vertices are connected:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle

https://www.quora.com/Why-hasnt-BART-expanded-to-the-South-Bay

Poll: How can we get the edge with the smallest weight at each step?

A. Store the edges in a Priority Queue, and remove the front item one by one
B. Store the edges in a regular list, and search for and remove the smallest item one by one
C. Store the edges in a Binary Search Tree (SortedSet) and remove the leftmost node one by one
D. Store the edges in a Hash Table and remove the leftmost node one by one

Poll: How can we tell if an edge (between node1 and node2) would cause a cycle?

A. Union-find: If the `find` operation returns the same root for both node1 and node2, then that edge would cause a cycle

B. Union-find: If the `find` operation for node1 returns node2, or vice versa, then that edge would cause a cycle

C. Recursive backtracking: starting at node1, search for all nodes that can be reached. If node2 is in that set, then that edge would cause a cycle

Kruskal's Algorithm (formal)

1. Create a forest F (a set of trees), where each vertex in the graph is a separate tree.

2. Create a sorted set S containing all the edges in the graph.

3. While S is nonempty and F is not yet spanning
   a. Remove the edge with the minimum weight from S.
   b. If the removed edge connects two different trees, then add it to the forest F, combining two trees into a single tree.

https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

Kruskal's Algorithm (formal)

Create a node for each BART station, where the data is the station and the parent is itself (root)

1. Create a forest F (a set of trees), where each vertex in the graph is a separate tree.

Put the edges in a priority queue

2. Create a sorted set S containing all the edges in the graph.

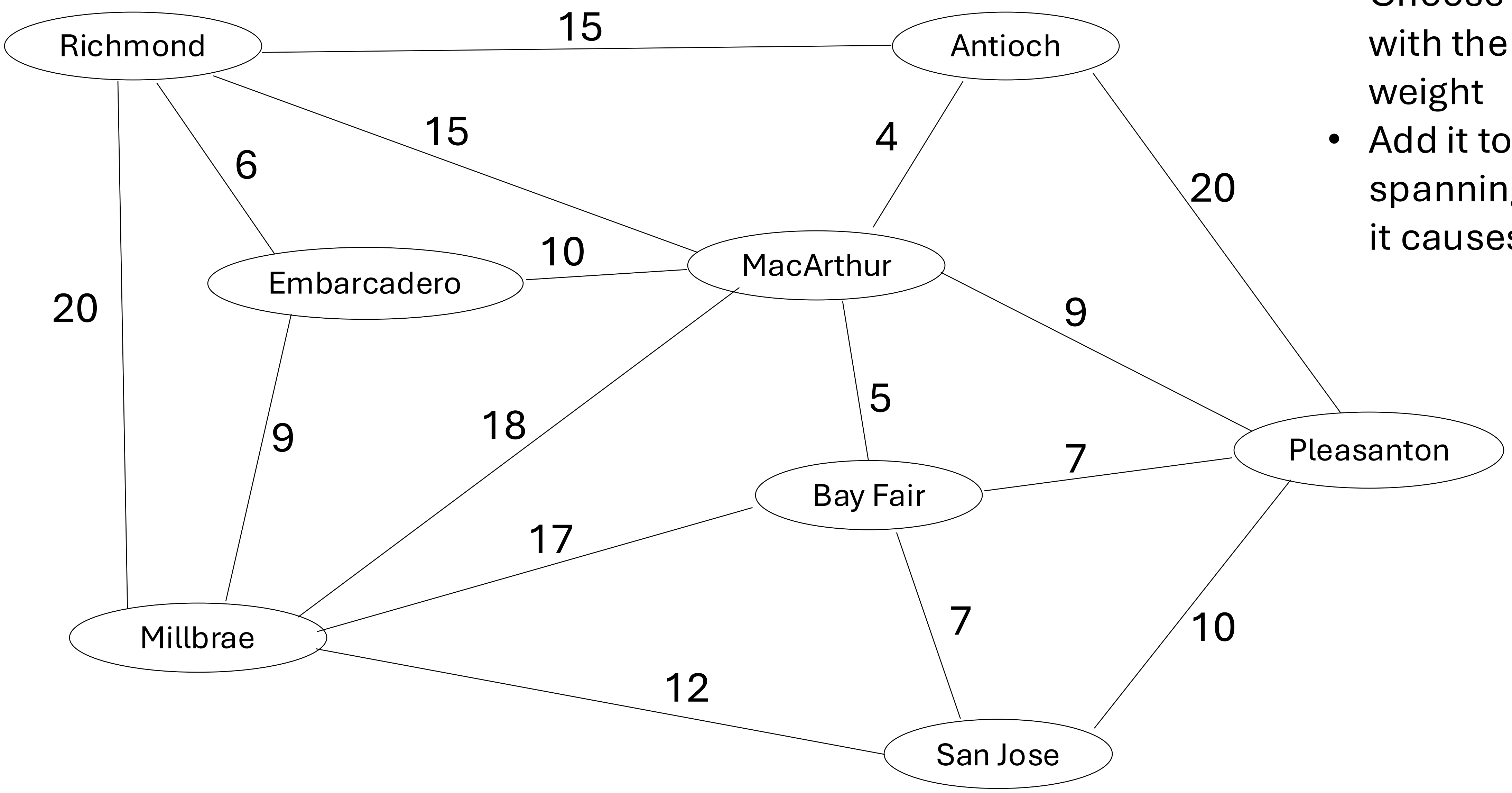
There are some stations s1 and s2 such that the train cannot get from s1 to s2

Meaning: the Union-Find still has multiple trees in it

3. While S is nonempty and F is not yet spanning
   a. Remove the edge with the minimum weight from S.
   b. If the removed edge connects two different trees, then add it to the forest F, combining two trees into a single tree.

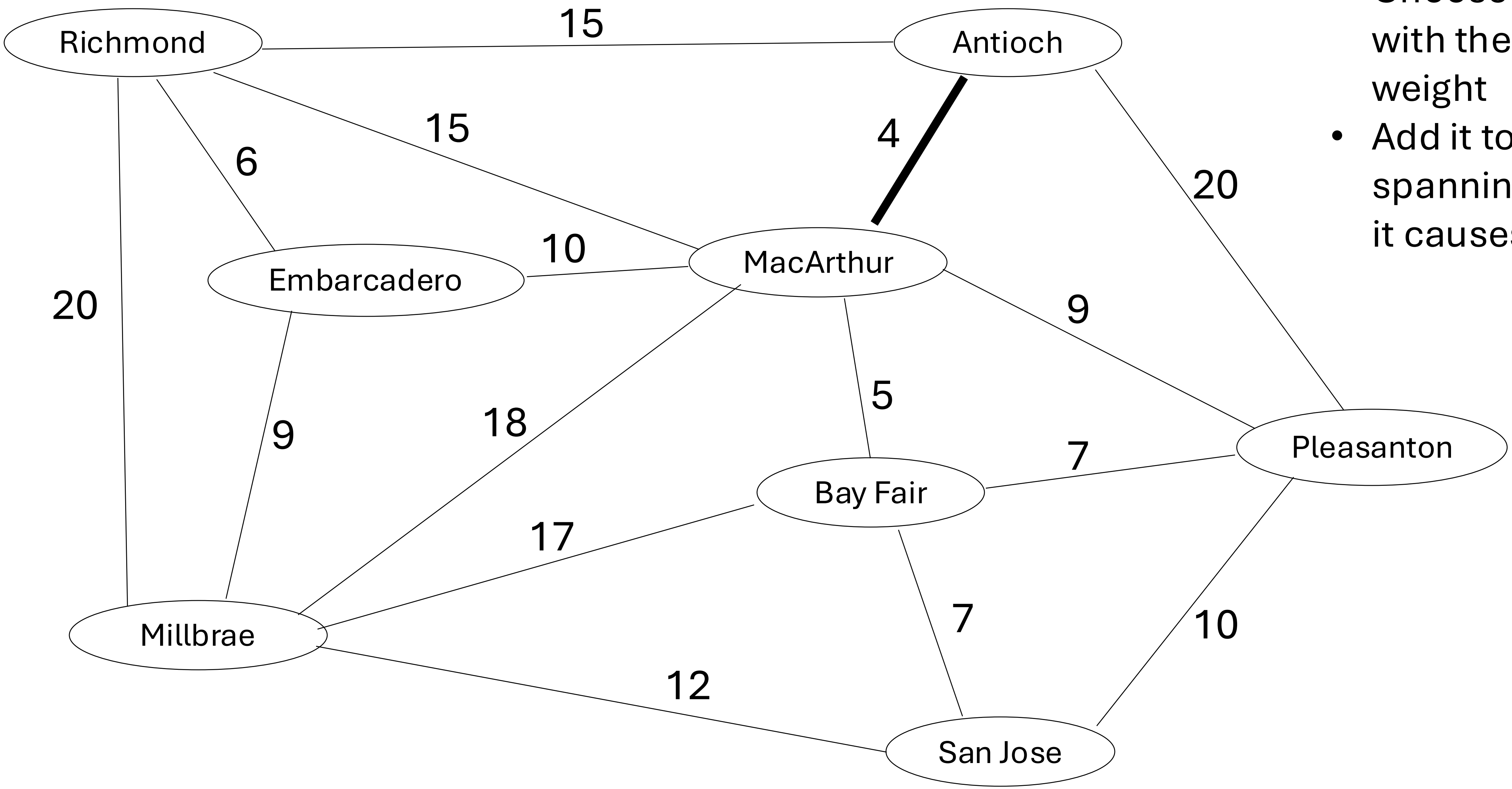
https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

Another example MST (different weights)



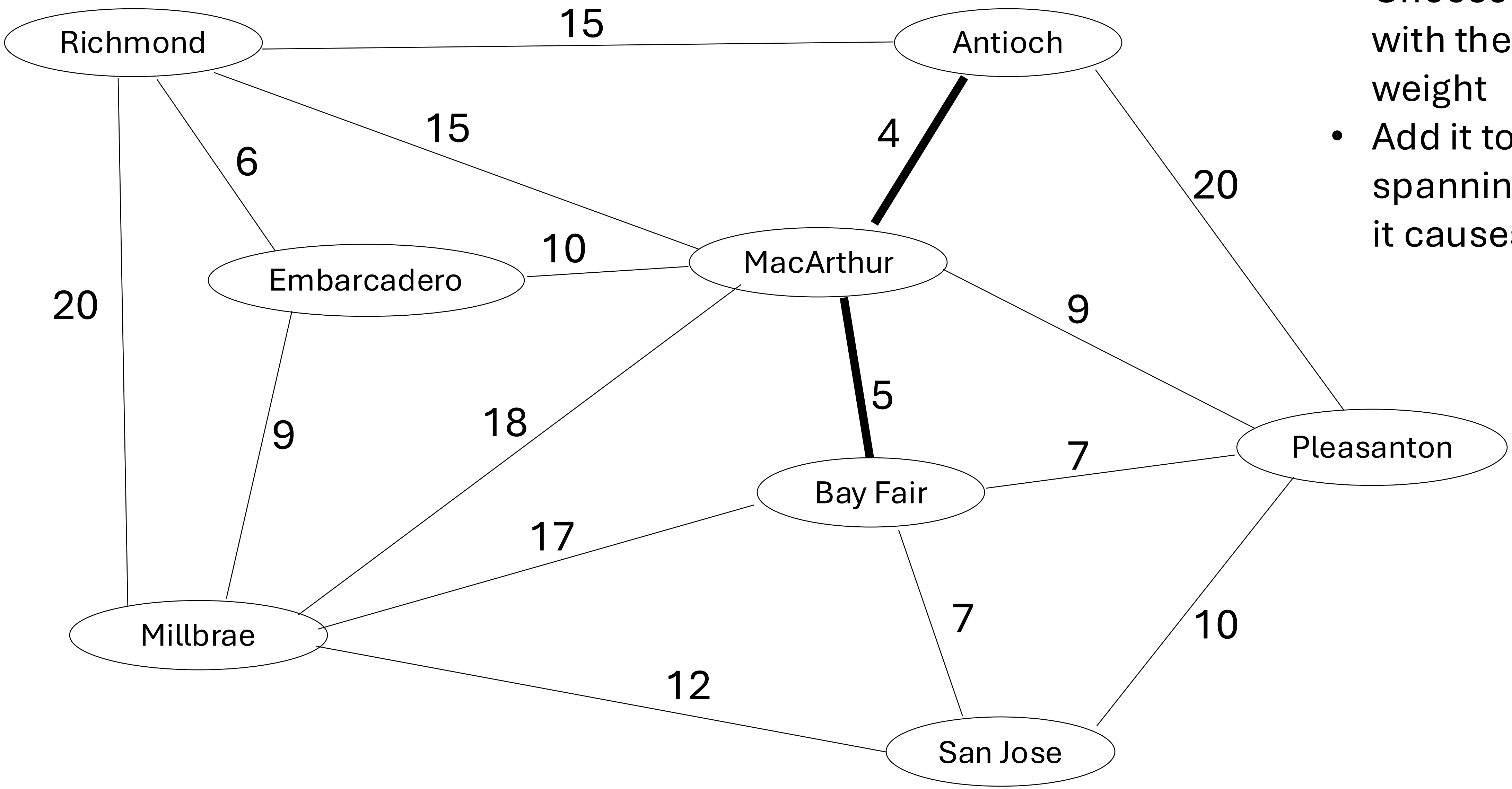
Until all the vertices are connected together:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle



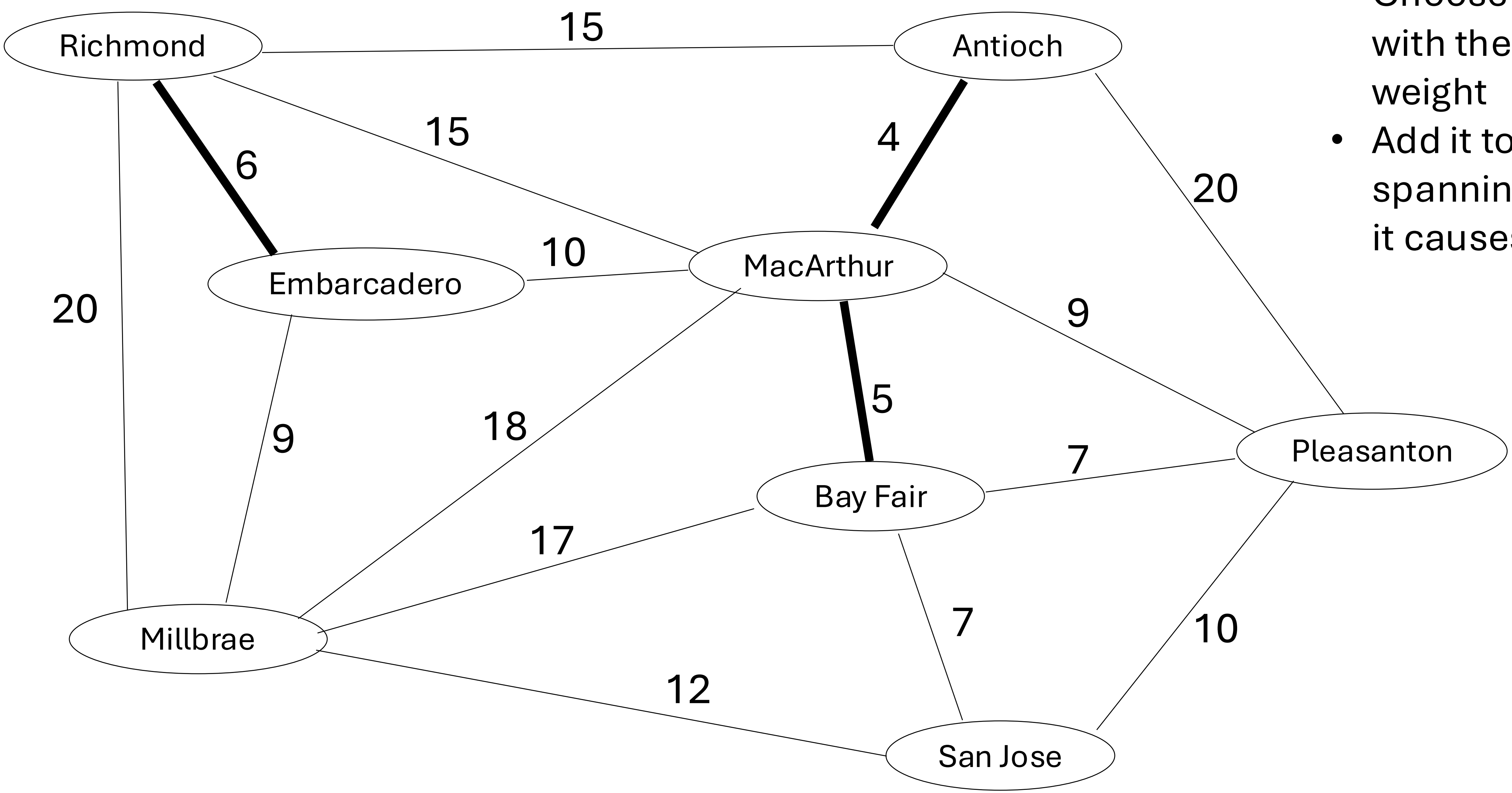
Until all the vertices are connected together:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle



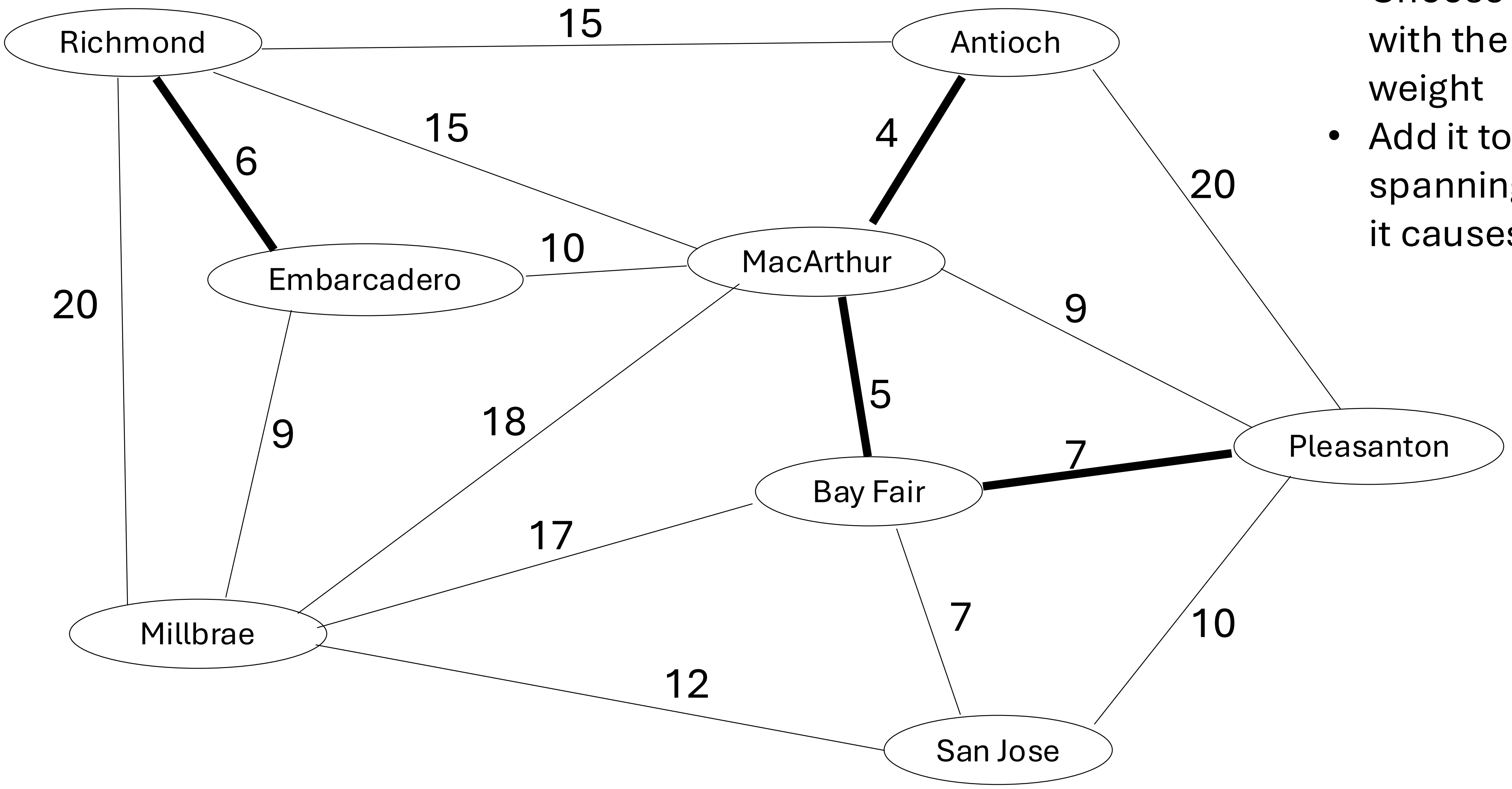
Until all the vertices are connected together:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle



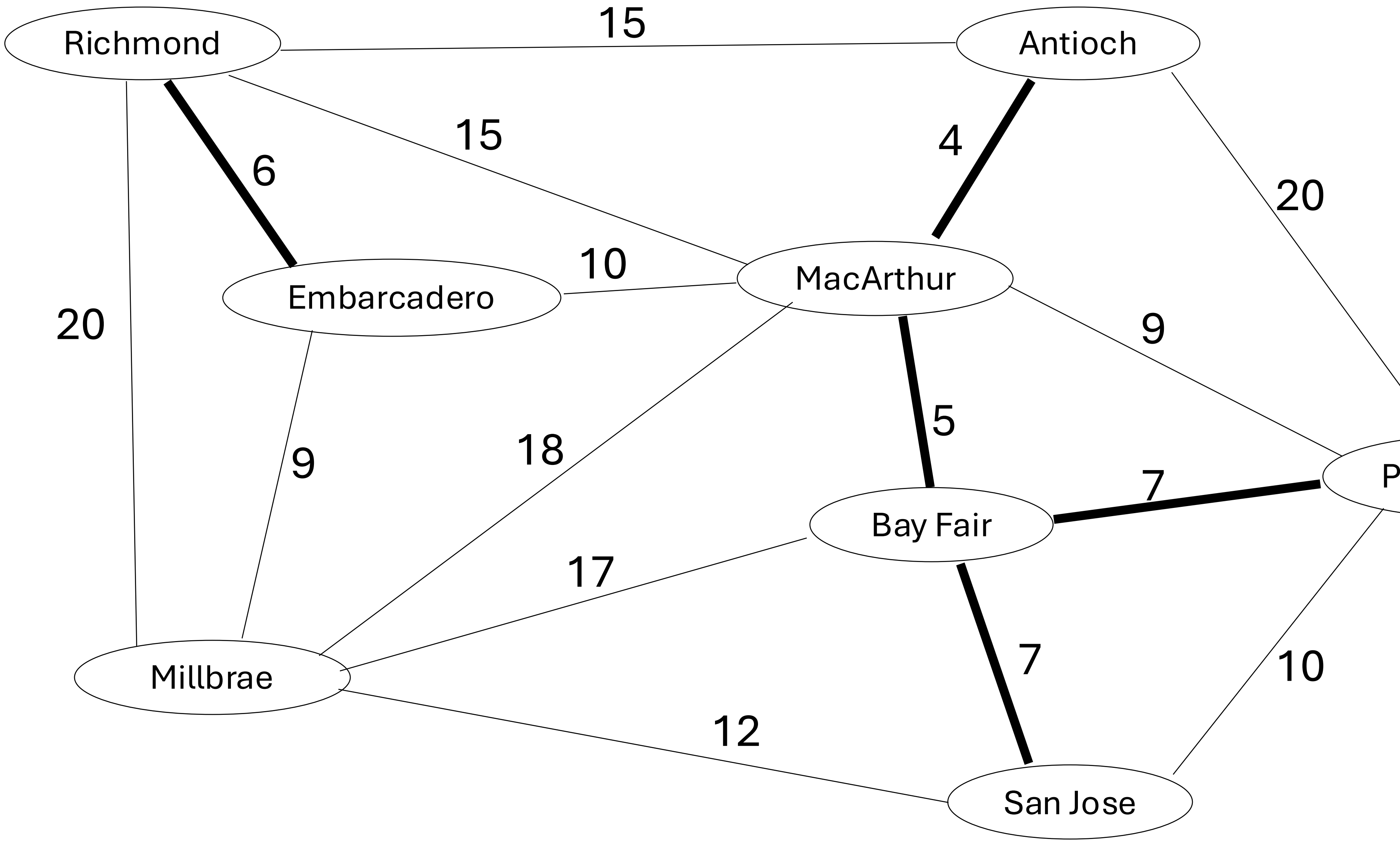
Until all the vertices are connected together:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle



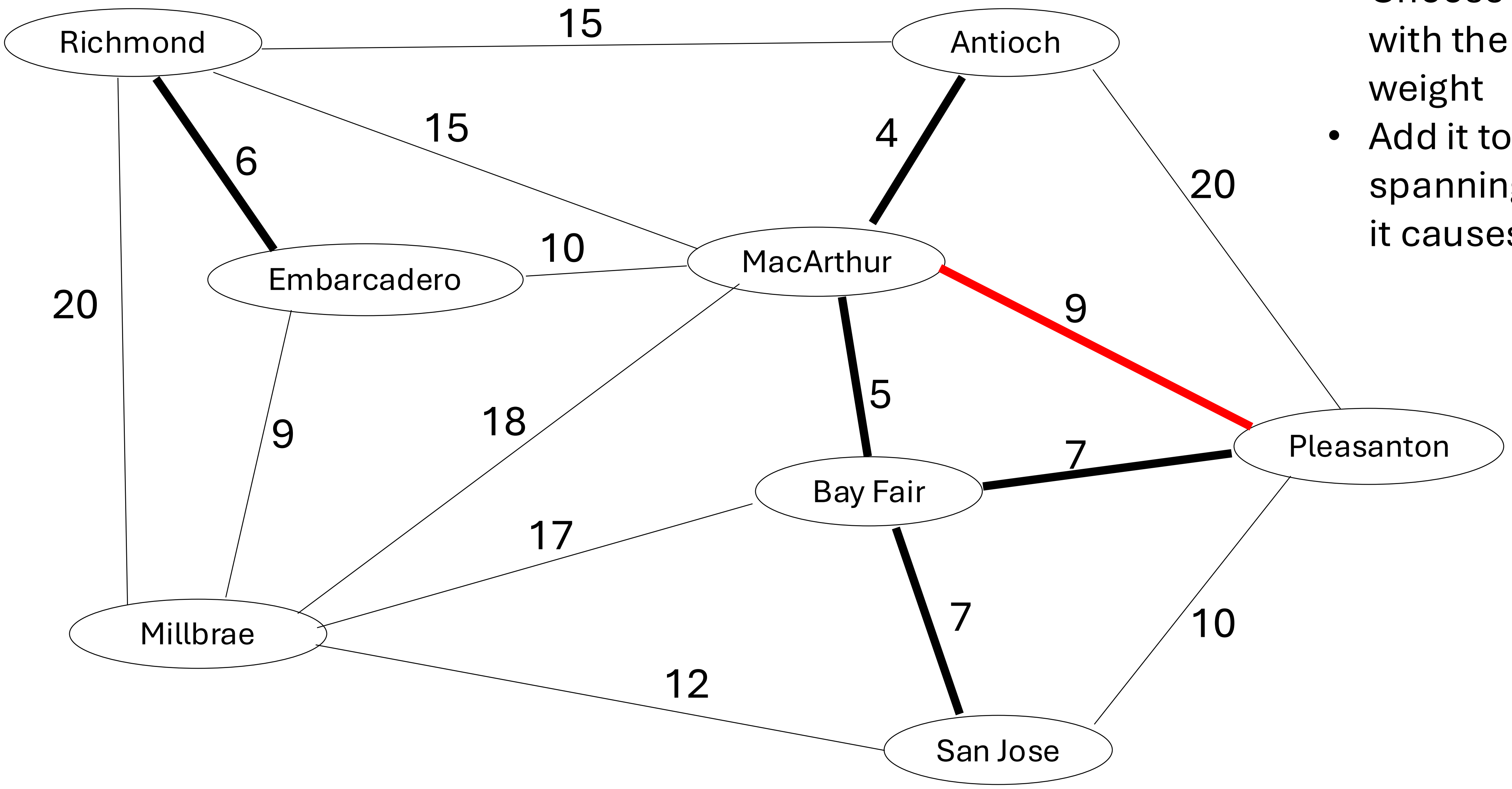
Until all the vertices are connected together:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle



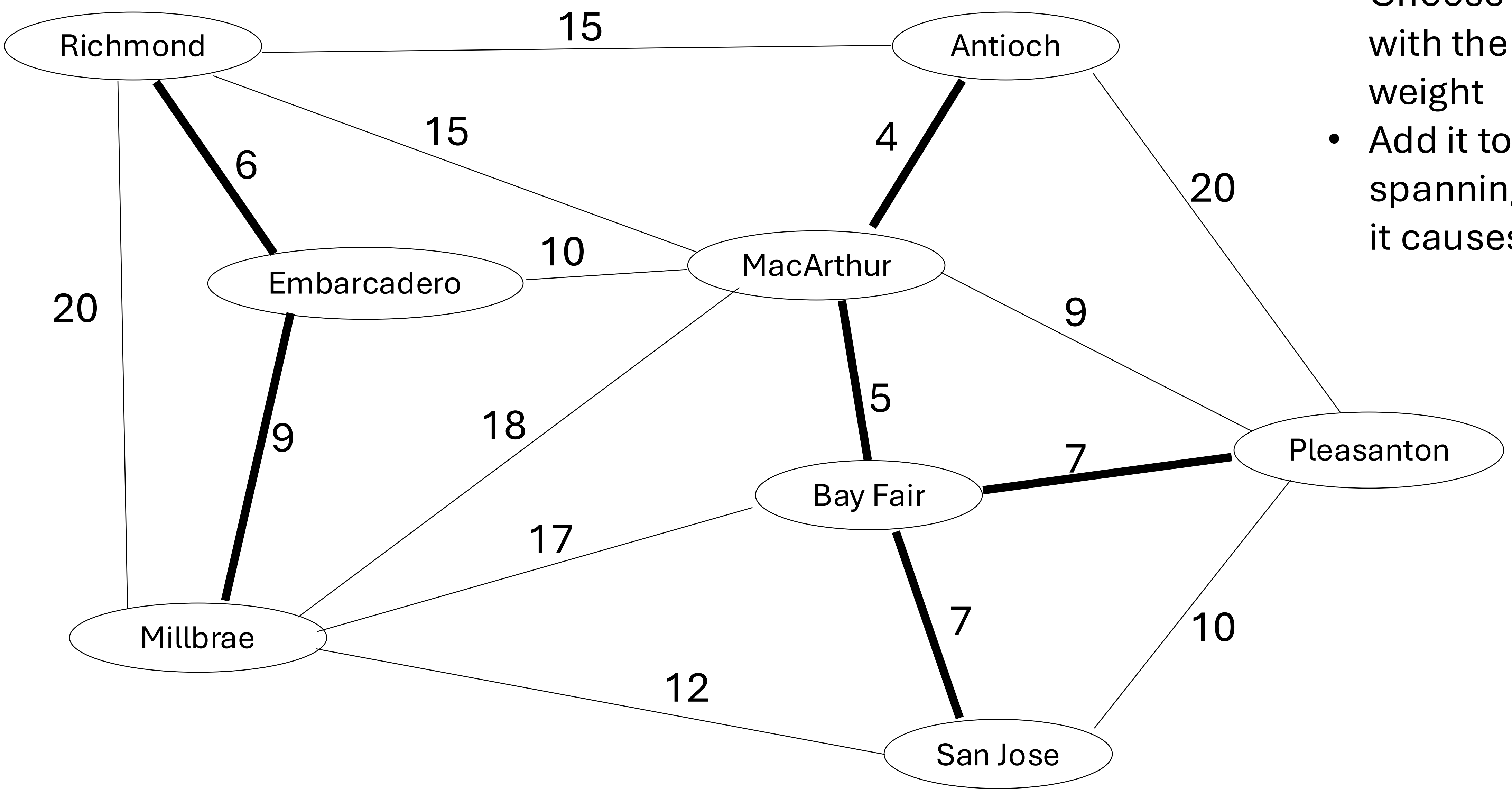
Until all the vertices are connected together:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle



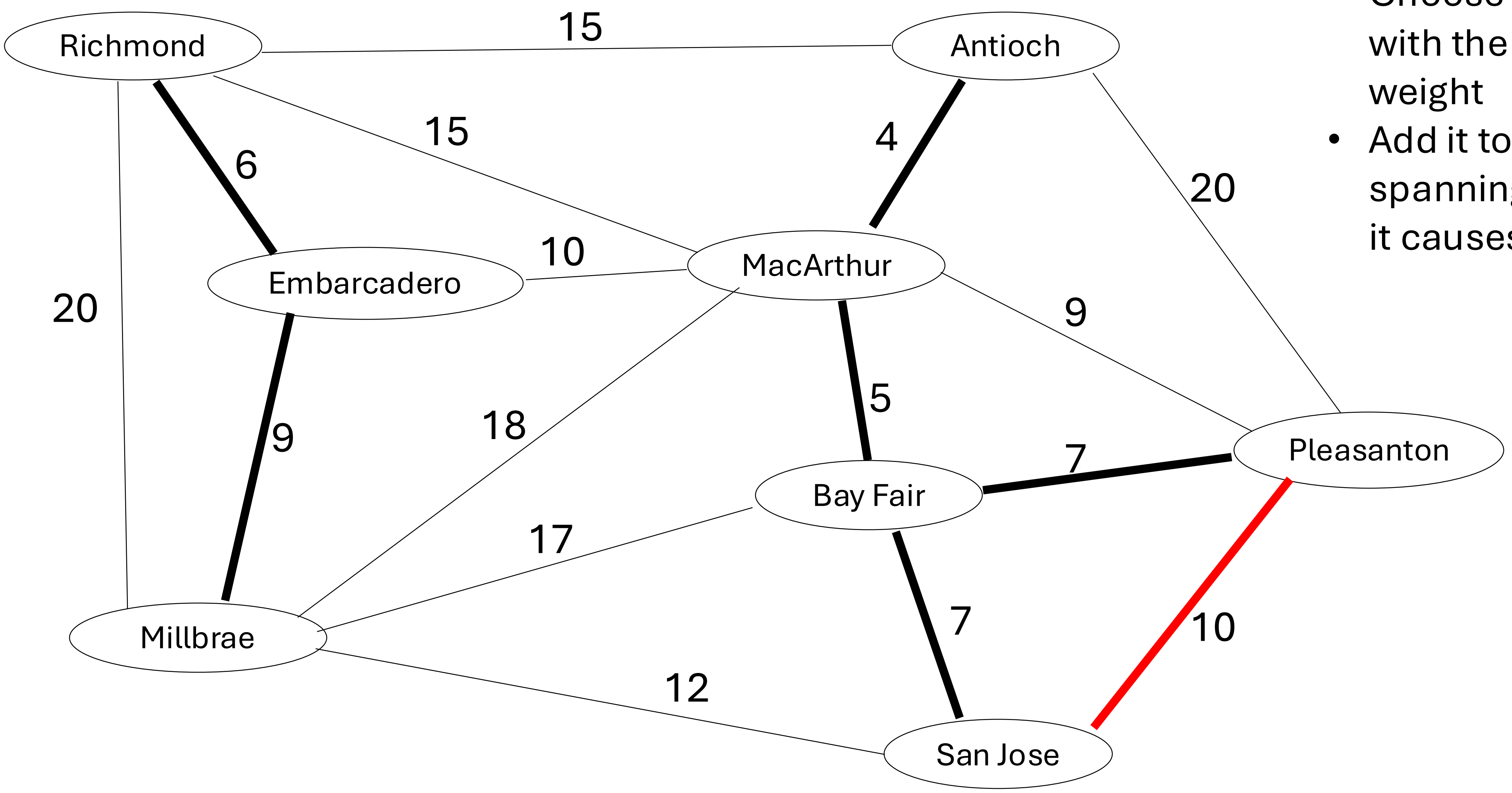
Until all the vertices are connected together:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle



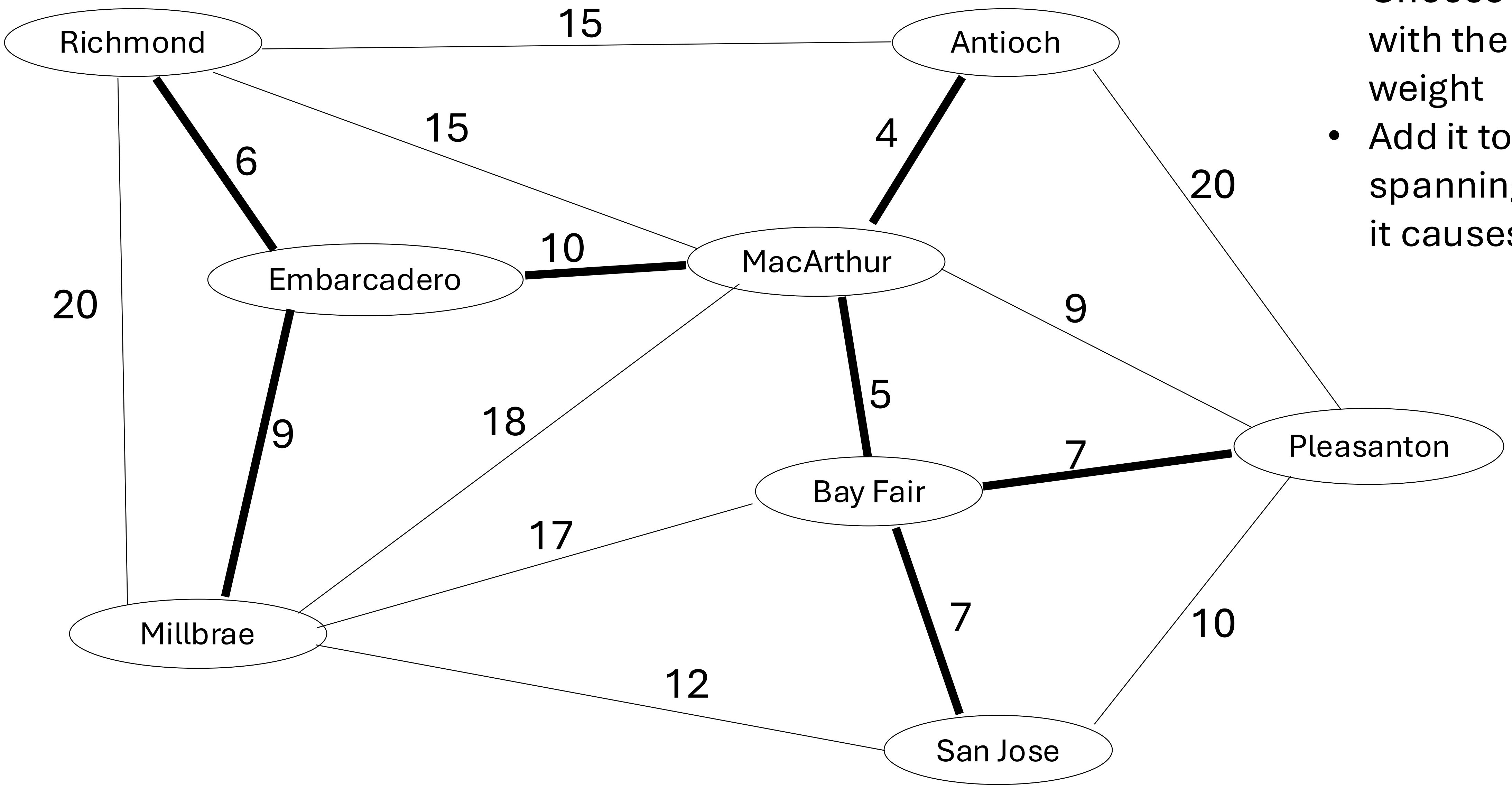
Until all the vertices are connected together:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle



Until all the vertices are connected together:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle



Until all the vertices are connected together:

- Choose the edge with the minimum weight
- Add it to the spanning tree unless it causes a cycle

Poll: Which of the following are true?

A. To iterate through the edges, smallest to largest, we can store them in a Priority Queue
B. To determine whether adding an edge would cause a cycle, we can use union-find to see if they are already in the same tree
C. We complete Kruskal's Algorithm when each node has at least one edge connected to it in the tree
D. The weighted graph edges chosen for the MST are different from the directed parent edges used for union-find