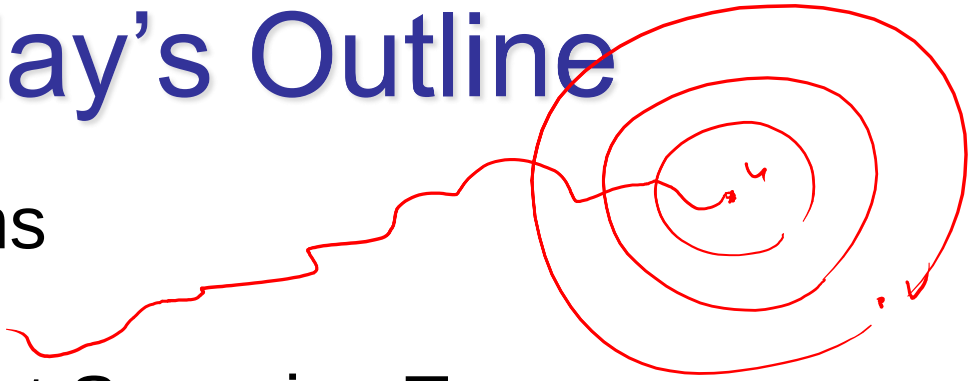
Today's Outline

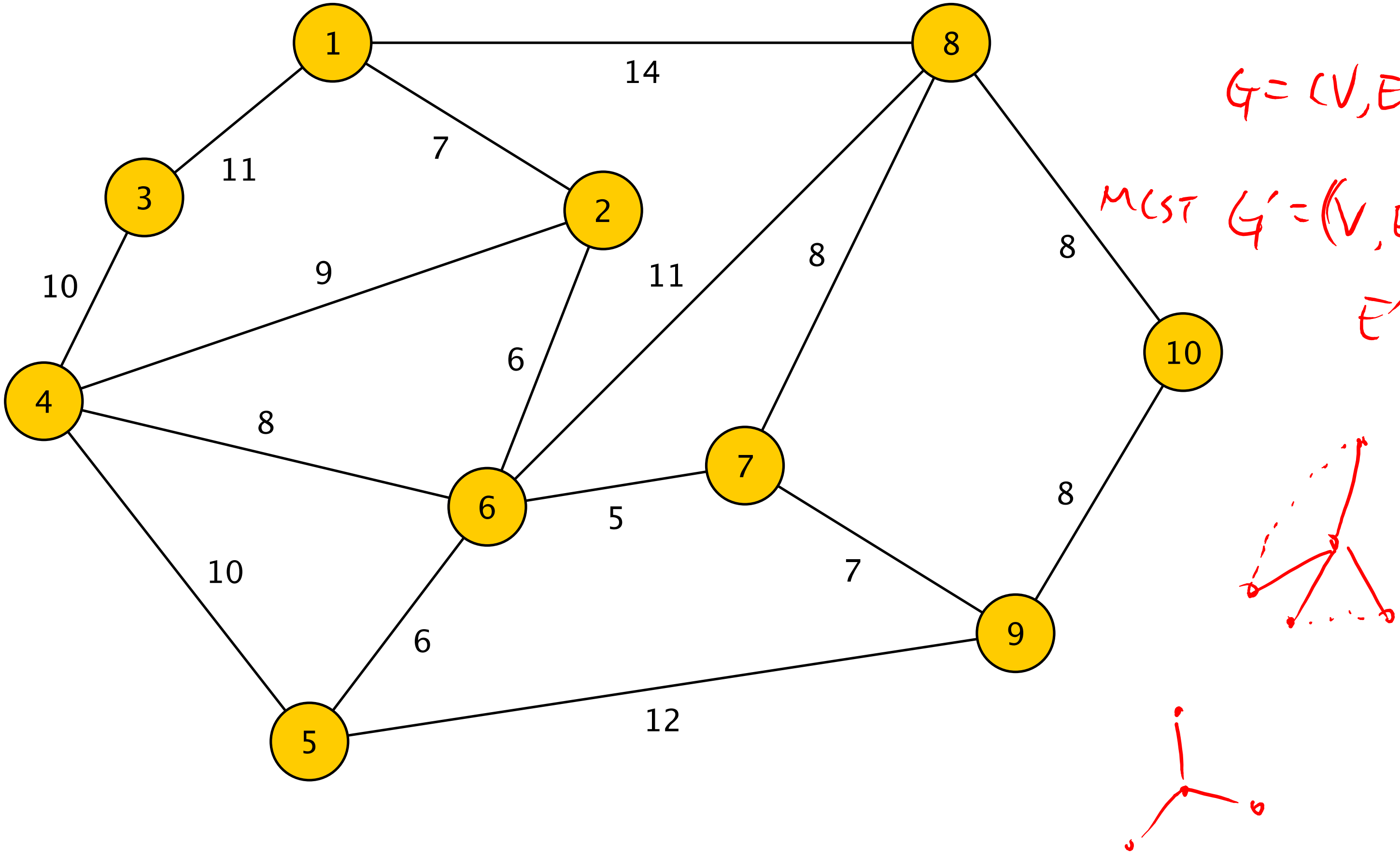


- Graph Algorithms
  - Reachability
  - Minimum-Cost Spanning Tree
  - Single Source Shortest Path

Minimum-Cost Spanning Trees

cover all v in V

no cycle



G = (V,E)

MCST G' = (V, E')

E' ⊂ E

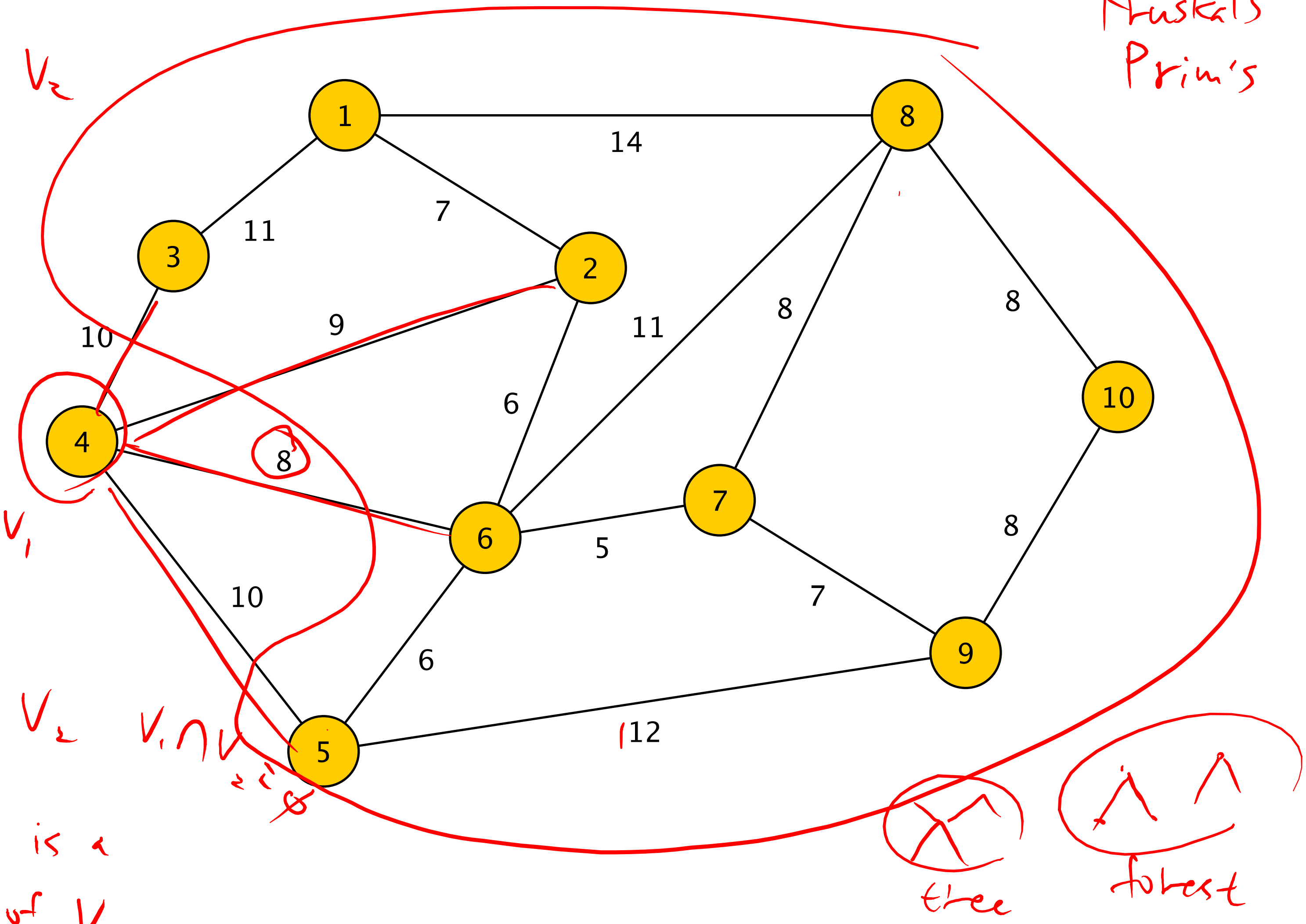
Minimum-Cost Spanning Trees

Given a connected, undirected graph G=(V,E) with non-negative edge weights, find a minimum-weight, connected, spanning subgraph of G.

↑
will be a tree
since we want to
minimize the cost
$\shortparallel$
$\sum$ edge weights

Minimum-Cost Spanning Trees

Kruskal's
Prim's



$V = V_1 \cup V_2$

$V_1 \cap V_2 = \emptyset$

V_1 and V_2 is a partition of V

Kruskal’s Algorithm

```
kruskal(G) //G=(V,E)
    sort E by edge weight, the smallest to the largest
    E’←{}
    V’←{}
    for each (u,v) in E:
        if u not in V’ or v not in V’: //if no cycle will be made
            E’← E’ ∪ {(u,v)} // add (u,v) to E’
            V’← V’ ∪ {u,v}  // add u and v to V’
return (V,E’) // V=V’ since G is connected
```

Prim’s Algorithm

```
prim(G) //G=(V,E)
    v ← a randomly chosen vertex in V
    $V_1$ ← {v}
    $V_2$ ← V - {v}
    E’ ← {}
    while(|$V_1$|<|V|)
        (u,v) ← cheapest edge between $V_1$ and $V_2$ (u in $V_1$ & v in $V_2$)
        E’ ← E’ ∪ {(u,v)} // add (u,v) to E’
        $V_1$ ← $V_1$ ∪ {v}
        $V_2$ ← $V_2$ - {v}
return (V,E’)
```

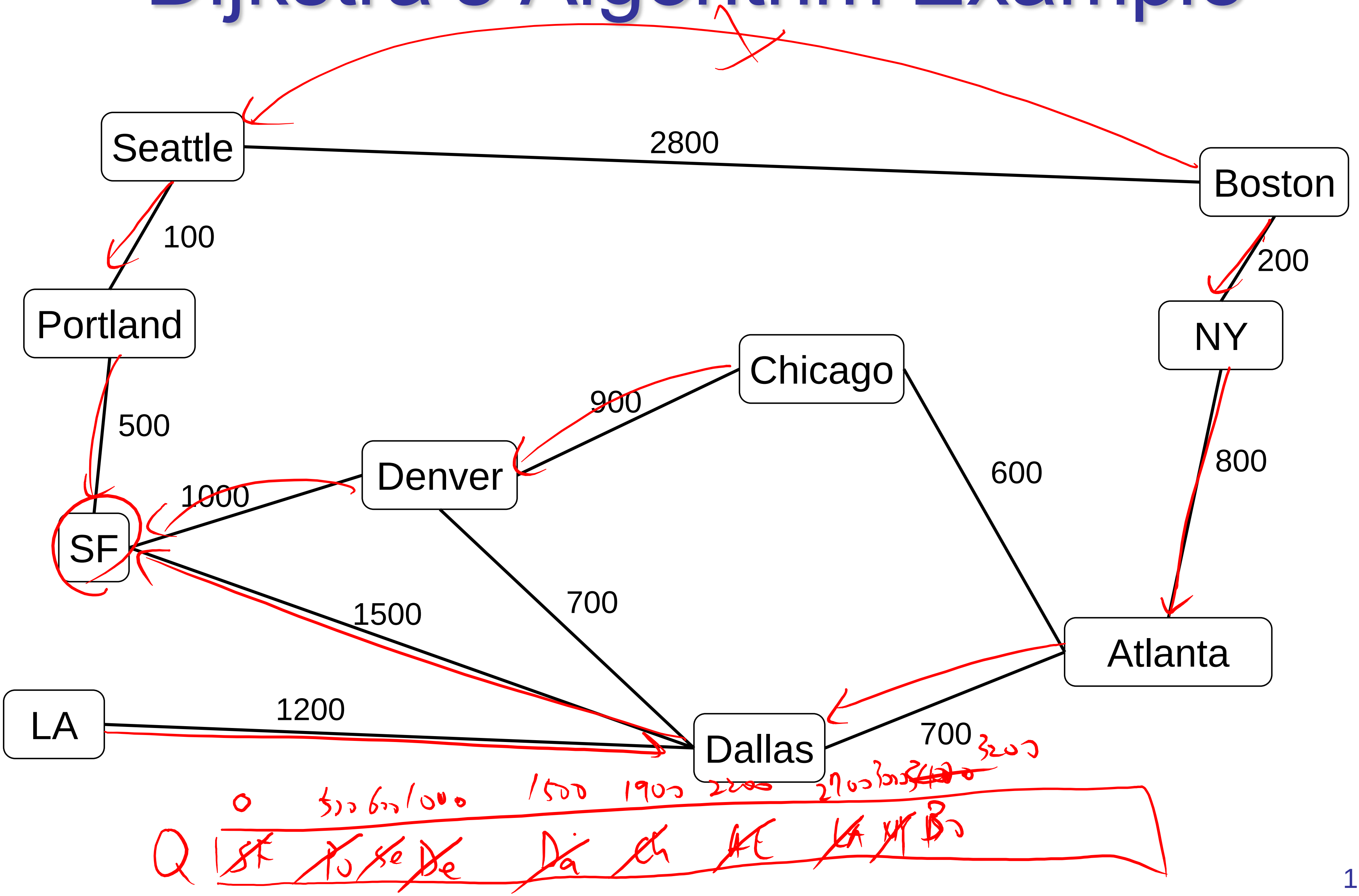
Today’s Outline

- Graph Algorithms
  - Reachability
  - Minimum-Cost Spanning Tree
  - ➤ Single Source Shortest Path

Single Source Shortest Paths

The Problem: Given a graph G and a starting vertex v, find, for *each* vertex u≠v reachable from v, a shortest path from v to u.

Dijkstra's Algorithm Example



Dijkstra’s Algorithm

```
Dijkstra(G, s):
   Q ← an empty priority queue
   Q.insert_with_priority(s, 0) // the key will the distance from s
                                 // (the shortest found so far)
   while Q is not empty:
     u ← Q.remove_highest_priority() // u has the smallest key
     for each neighbor v of u:
        new_dist ← u.key() + edge_length(u, v) //u.key()=dist(s,u)
        if v in Q:
           if new_dist < v.key():
              Q.update(v, new_dist) // v.prev now points to u
              // or remove(v) and insert_with_priority(v, new_dist)
        else:
           Q.insert_with_priority(v, new_dist)
```

// now, for each u reachable from s, the shortest path can be constructed by following *prev* starting from u until s is reached.