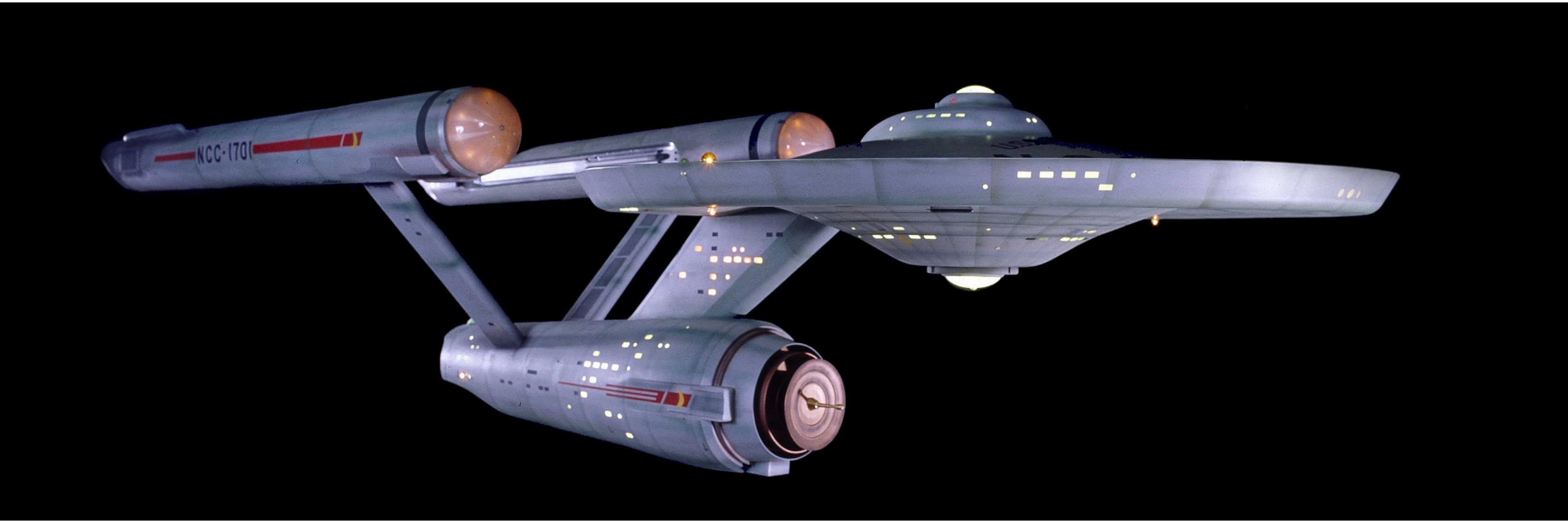


CSCI 241

Lecture 22
Dijkstra’s Algorithm:
Proof of Correctness; Practice

Announcements

Announcements

- Quiz today as usual.

Announcements

- Quiz today as usual.
- Midterm grades are out - see announcement

Announcements

- Quiz today as usual.
- Midterm grades are out - see announcement
- A2 grades are out - nice work!

Goals

- See a proof of correctness of Dijkstra's algorithm
- Answer any questions you have about implementation / A4.
- Get some practice running Dijkstra on paper

Dijkstra's Shortest Paths: Cartoon

settled　　frontier　　unexplored

Before:

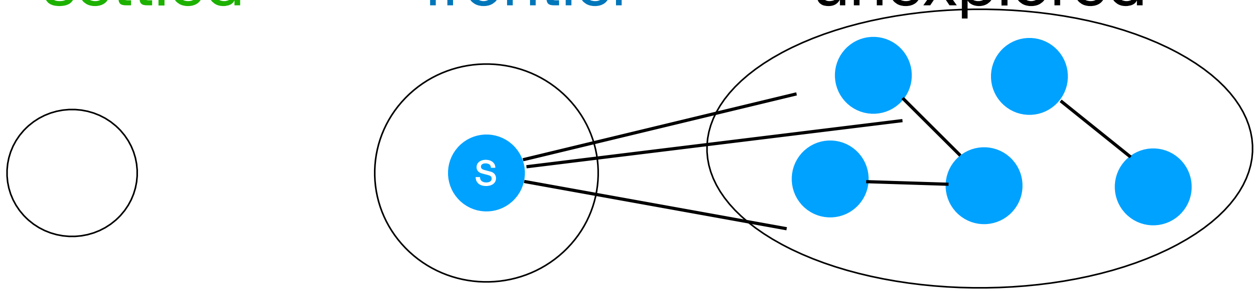
During:

After:

Dijkstra's Shortest Paths: Cartoon

settled frontier unexplored

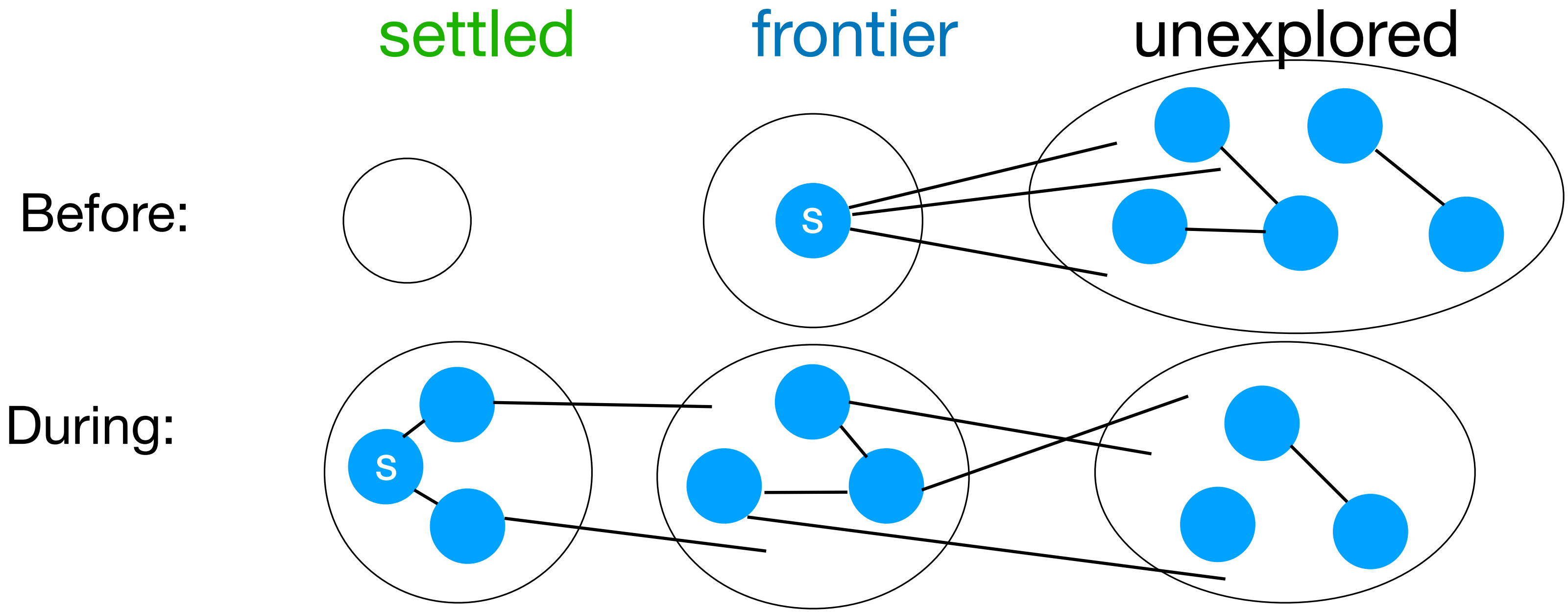
Before:



During:

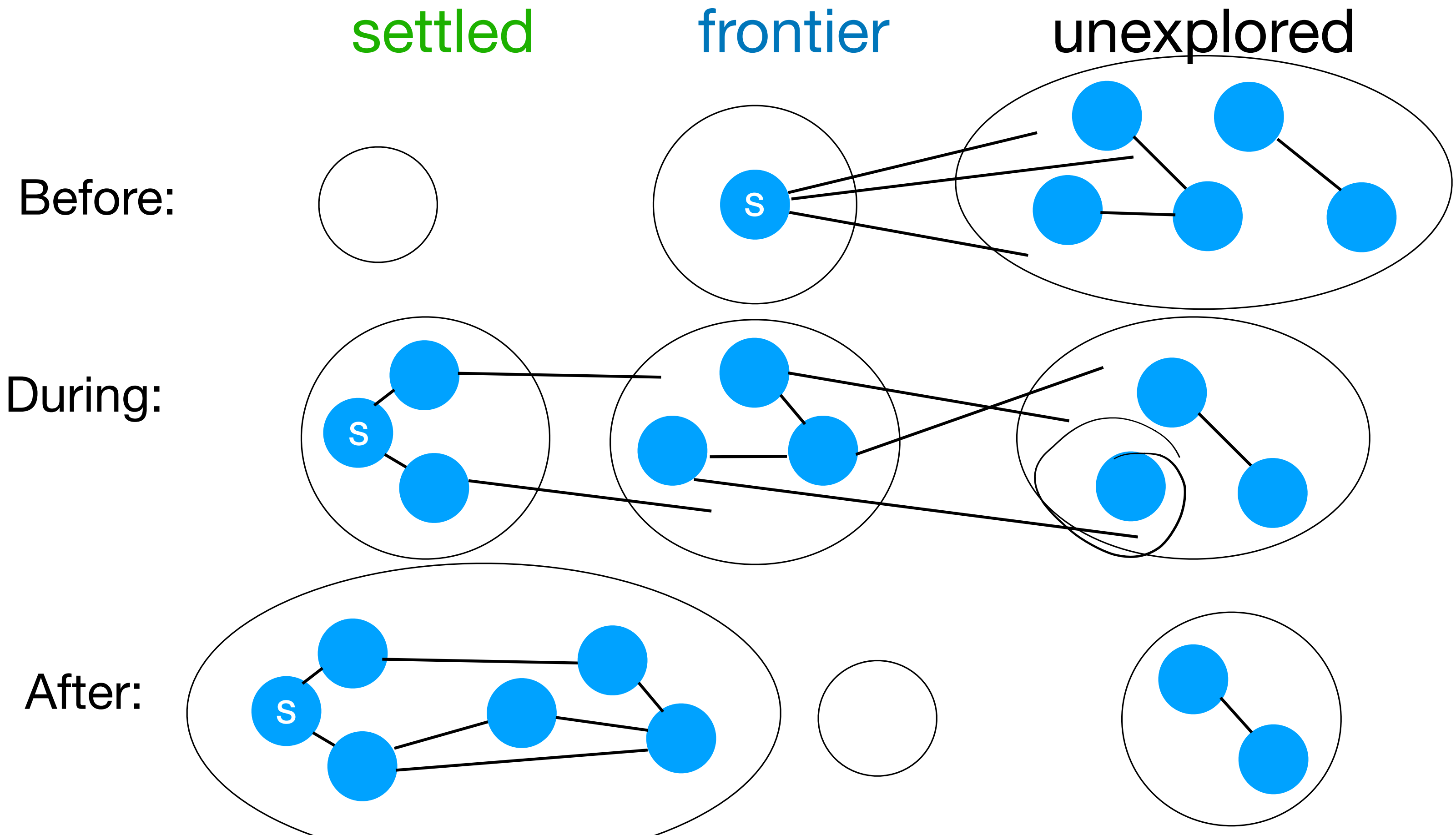
After:

Dijkstra's Shortest Paths: Cartoon

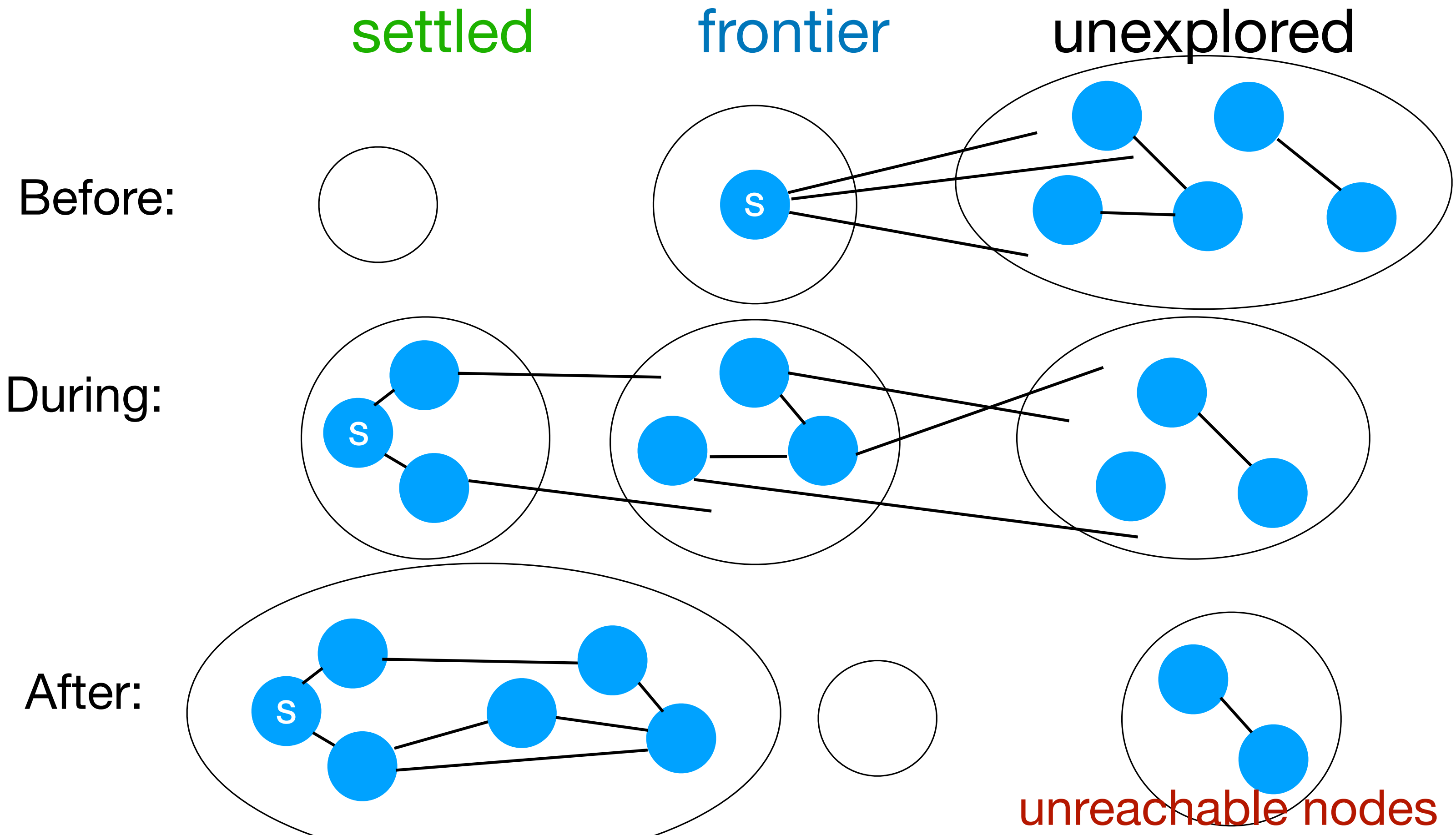


After:

Dijkstra's Shortest Paths: Cartoon



Dijkstra’s Shortest Paths: Cartoon

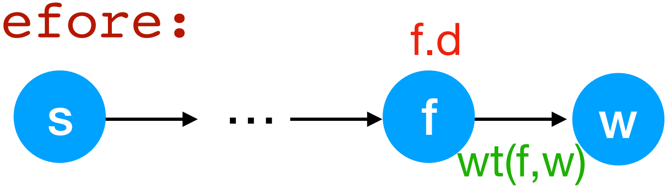


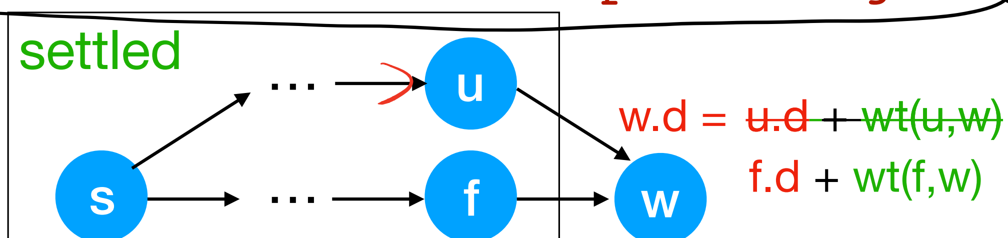
Dijkstra’s Shortest Paths: Intuition

- Intuition: explore nodes kinda like BFS.
- There are three kinds of nodes:
  - Settled - nodes for which we know the actual shortest path.
  - Frontier - nodes that have been visited but we don’t necessarily have their actual shortest path
  - Unexplored - all other nodes.
- Each node n keeps track of `n.d`, the length of the shortest known known path from start.
- We may discover a shorter path to a frontier node than the one we’ve found already - if so, update `n.d`.

Dijkstra's Shortest Paths: High-Level Algorithm

```
Initialize Settled to empty
Initialize Frontier to the start node
While the frontier isn't empty:
   move the node f with smallest d from F to S
   For each neighbor w of f:
     if we've never seen w before:
        set its path length
        add it to frontier
   else if the path to w via f is shorter:
        update w's shortest path length
```





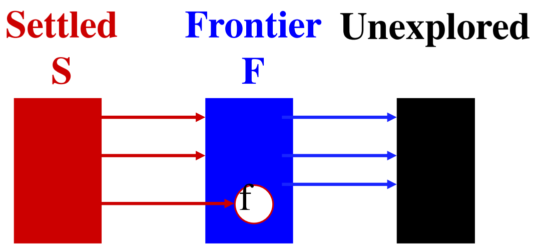
Proof of Correctness

- Dijkstra's algorithm is **greedy**: it makes a sequence of *locally* optimal moves, which results in the *globally* optimal solution.
    - Most algorithms don't work like this - need to prove that it results in the global optimum.
- Specifically: It is not obvious that there cannot still be a shorter path to the Frontier node with smallest d-value.

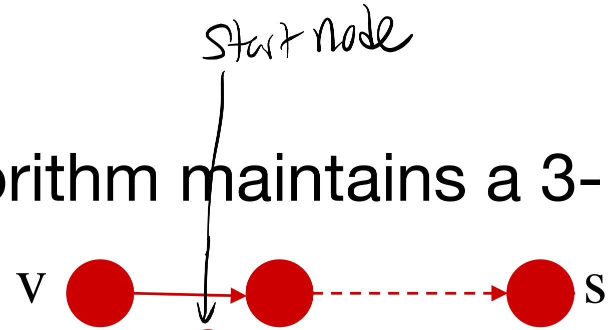
Proof Sketch

1. State a loop invariant.
2. Prove that **if** that invariant is maintained, **then** the algorithm is correct.
3. Prove that the algorithm maintains the invariant.

Proof of Correctness: Invariant

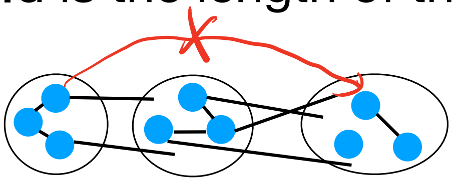


The while loop in Dijkstra's algorithm maintains a 3-part invariant:



1. For a Settled node s, a shortest path from v to s contains only settled nodes and s.d is length of shortest v -> s path.

2. For a Frontier node f, at least one v -> f path contains only settled nodes (except perhaps for f) and f.d is the length of the shortest such path



3. All edges leaving S go to F (or: no edges from S to Unexplored)

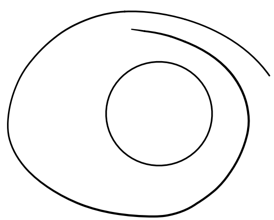
Proof of Correctness: Theorem

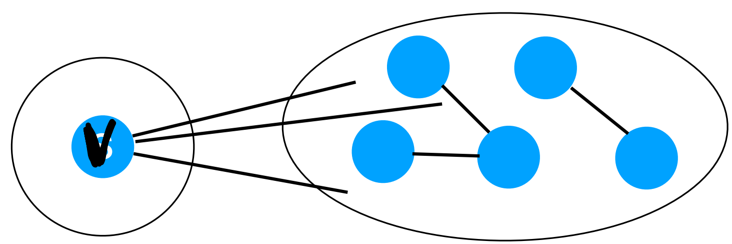
```
S = { }; F = {v};  v.d = 0;
while  (F ≠ {})  {
   f = node in F with min d value;
   Remove f from F, add it to S;
   for each neighbor w of f {
      if (w not in S or F) {
         w.d =  f.d + weight(f, w);
         add w to F;
      } else if (f.d+weight(f,w) < w.d) {
         w.d = f.d+weight(f,w);
      }
   }
}
```

Theorem: For a node f in the Frontier with minimum d value (over all nodes in the Frontier), f.d is the shortest-path distance from v to f.

Proof: Show that any other path from v to f has length >= f.d

Case 1: if v is in F, then S is empty and v.d = 0, which is trivially the shortest distance from v to v.





Proof of Correctness: Theorem

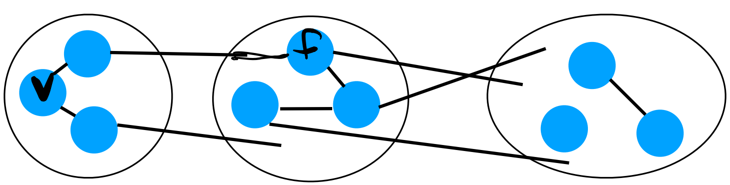
```
S = { }; F = {v};  v.d = 0;
while  (F ≠ {})  {
   f = node in F with min d value;
   Remove f from F, add it to S;
   for each neighbor w of f {
      if (w not in S or F) {
         w.d =  f.d + weight(f, w);
         add w to F;
      } else if (f.d+weight(f,w) < w.d) {
         w.d = f.d+weight(f,w);
      }
   }
}
```

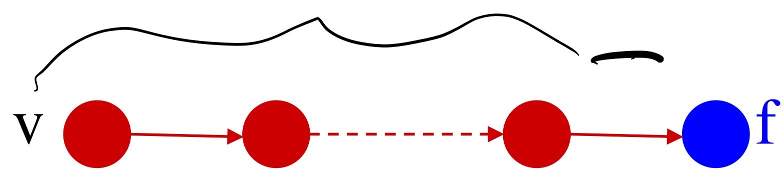
Theorem: For a node f in the Frontier with minimum d value (over all nodes in the Frontier), f.d is the shortest-path distance from v to f.

Proof: Show that any other path from v to if has length >= f.d

Case 2: v is in S. Part 2 of the invariant says:

- f.d is the length of the shortest path from v to f containing all settled nodes except f, and f.d is the length of such a path.





Proof of Correctness: Theorem

```
S = { }; F = {v};  v.d = 0;
while  (F ≠ {})  {
   f = node in F with min d value;
   Remove f from F, add it to S;
   for each neighbor w of f {
      if (w not in S or F) {
         w.d =  f.d + weight(f, w);
         add w to F;
      } else if (f.d+weight(f,w) < w.d) {
         w.d = f.d+weight(f,w);
      }
   }
}
```

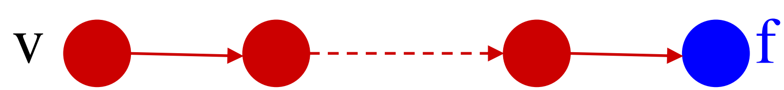
Theorem: For a node f in the Frontier with minimum d value (over all nodes in the Frontier), f.d is the shortest-path distance from v to f.

Proof: Show that any other path from v to if has length >= f.d

Case 2: v is in S. Part 2 of the invariant says:

- f.d is the length of the shortest path from v to f containing all settled nodes except f, and f.d is the length of such a path.

Any other v-f path must either be longer or go through another frontier node g then arrive at f:



Proof of Correctness: Theorem

```
S = { }; F = {v};  v.d = 0;
while  (F ≠ {})  {
   f = node in F with min d value;
   Remove f from F, add it to S;
   for each neighbor w of f {
      if (w not in S or F) {
         w.d =  f.d + weight(f, w);
         add w to F;
      } else if (f.d+weight(f,w) < w.d) {
         w.d = f.d+weight(f,w);
      }
   }
}
```

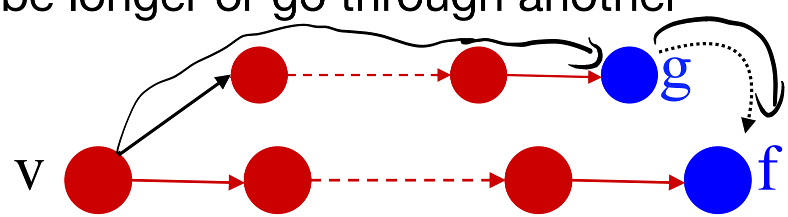
Theorem: For a node f in the Frontier with minimum d value (over all nodes in the Frontier), f.d is the shortest-path distance from v to f.

Proof: Show that any other path from v to if has length >= f.d

Case 2: v is in S. Part 2 of the invariant says:

- f.d is the length of the shortest path from v to f containing all settled nodes except f, and f.d is the length of such a path.

Any other v-f path must either be longer or go through another frontier node g then arrive at f:



Proof of Correctness: Theorem

```
S = { }; F = {v};  v.d = 0;
while  (F ≠ {})  {
   f = node in F with min d value;
   Remove f from F, add it to S;
   for each neighbor w of f {
      if (w not in S or F) {
         w.d =  f.d + weight(f, w);
         add w to F;
      } else if (f.d+weight(f,w) < w.d) {
         w.d = f.d+weight(f,w);
      }
   }
}
```

Theorem: For a node f in the Frontier with minimum d value (over all nodes in the Frontier), f.d is the shortest-path distance from v to f.

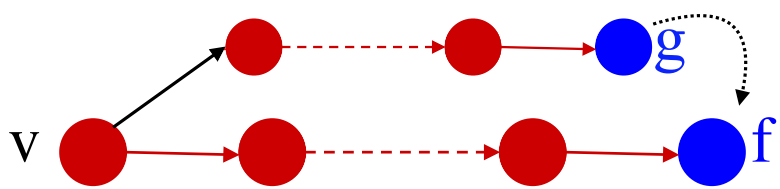
Proof: Show that any other path from v to if has length >= f.d

Case 2: v is in S. Part 2 of the invariant says:

- f.d is the length of the shortest path from v to f containing all settled nodes except f, and f.d is the length of such a path.

Any other v-f path must either be longer or go through another frontier node g then arrive at f:

d.f <= d.g,
so that path cannot be shorter



Proof of Correctness: Invariant Maintenance

```
S = { }; F = {v};  v.d = 0;
while  (F ≠ {})  {
   f = node in F with min d value;
   Remove f from F, add it to S;
   for each neighbor w of f {
      if (w not in S or F) {
          w.d =  f.d + weight(f, w);
          add w to F;
      } else if (f.d+weight(f,w) < w.d) {
          w.d = f.d+weight(f,w);
      }
   }
}
```

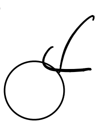
1. For a Settled node s, a shortest path from v to s contains only settled nodes and s.d is length of shortest v -> s path.

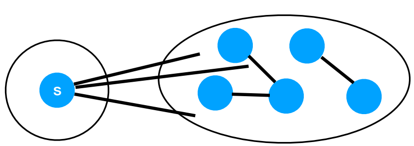
2. For a Frontier node f, at least one v -> f path contains only settled nodes (except perhaps for f) and f.d is the length of the shortest such path

3. All edges leaving S go to F (or: no edges from S to Unexplored)

Proof of Correctness: Invariant Maintenance

```
S = { }; F = {v};  v.d = 0;
while  (F ≠ {})  {
   f = node in F with min d value;
   Remove f from F, add it to S;
   for each neighbor w of f {
      if (w not in S or F) {
         w.d =  f.d + weight(f, w);
         add w to F;
      } else if (f.d+weight(f,w) < w.d) {
         w.d = f.d+weight(f,w);
      }
   }
}
```





1. For a Settled node s, a shortest path from v to s contains only settled nodes and s.d is length of shortest v -> s path.
2. For a Frontier node f, at least one v -> f path contains only settled nodes (except perhaps for f) and f.d is the length of the shortest such path
3. All edges leaving S go to F (or: no edges from S to Unexplored)

At initialization:

1. S is empty; trivially true.
2. v.d = 0, which is the shortest path.
3. S is empty, so no edges leave it.

Proof of Correctness: Invariant Maintenance

```
S = { }; F = {v};  v.d = 0;
while  (F ≠ {})  {
  f = node in F with min d value;
  Remove f from F, add it to S;
  for each neighbor w of f {
     if (w not in S or F) {
        w.d = f.d + weight(f, w);
        add w to F;
     } else if (f.d+weight(f,w) < w.d) {
        w.d = f.d+weight(f,w);
     }
  }
}
```

1. For a Settled node s, a shortest path from v to s contains only settled nodes and s.d is length of shortest v -> s path.
2. For a Frontier node f, at least one v -> f path contains only settled nodes (except perhaps for f) and f.d is the length of the shortest such path
3. All edges leaving S go to F (or: no edges from S to Unexplored)

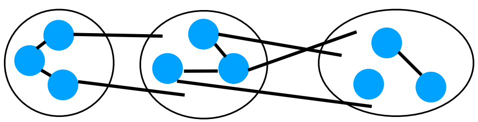
At each iteration:

1. Theorem says f.d is the shortest path, so it can safely move to S
2. Updating w.d maintains Part 2 of the invariant.
3. Each neighbor is either already in F or gets moved there.

Proof of Correctness: Invariant Maintenance

```
S = { }; F = {v};  v.d = 0;
while  (F ≠ {})  {
  f = node in F with min d value;
  Remove f from F, add it to S;
  for each neighbor w of f {
     if (w not in S or F) {
        w.d =  f.d + weight(f, w);
        add w to F;
    } else if (f.d+weight(f,w) < w.d) {
        w.d = f.d+weight(f,w);
    }
  }
}
```

1. For a Settled node s, a shortest path from v to s contains only settled nodes and s.d is length of shortest v -> s path.
2. For a Frontier node f, at least one v -> f path contains only settled nodes (except perhaps for f) and f.d is the length of the shortest such path
3. All edges leaving S go to F (or: no edges from S to Unexplored)



At each iteration:

1. Theorem says f.d is the shortest path, so it can safely move to S
2. Updating w.d maintains Part 2 of the invariant.
3. Each neighbor is either already in F or gets moved there.

Questions?

Dijkstra Practice

Draw the following directed, weighted graph:

Dijkstra(1)

V = {1, 2, 3, 4, 5, 6}
E = {
(1, 2): 7
(1, 3): 9
(1, 6): 14
(2, 3): 10
(2, 4): 15
(3, 4): 11
(3, 6): 2
(4, 5): 6
(6, 5): 9
}

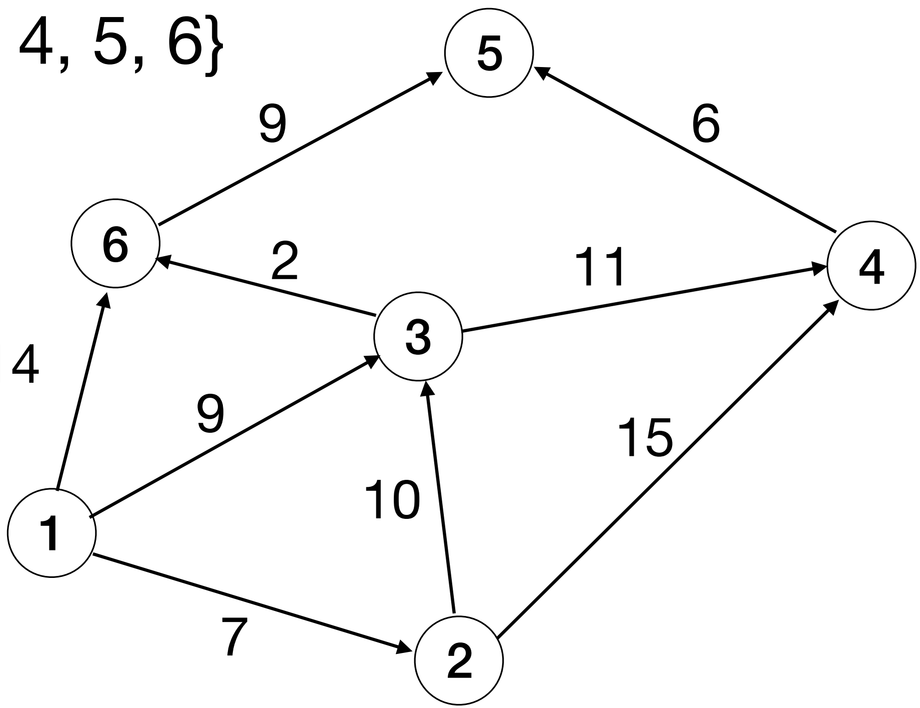
Dijkstra Practice

V = {1, 2, 3, 4, 5, 6}

E = {

(1, 2): 7
(1, 3): 9
(1, 6): 14
(2, 3): 10
(2, 4): 15
(3, 4): 11
(3, 6): 2
(4, 5): 6
(6, 5): 9

}



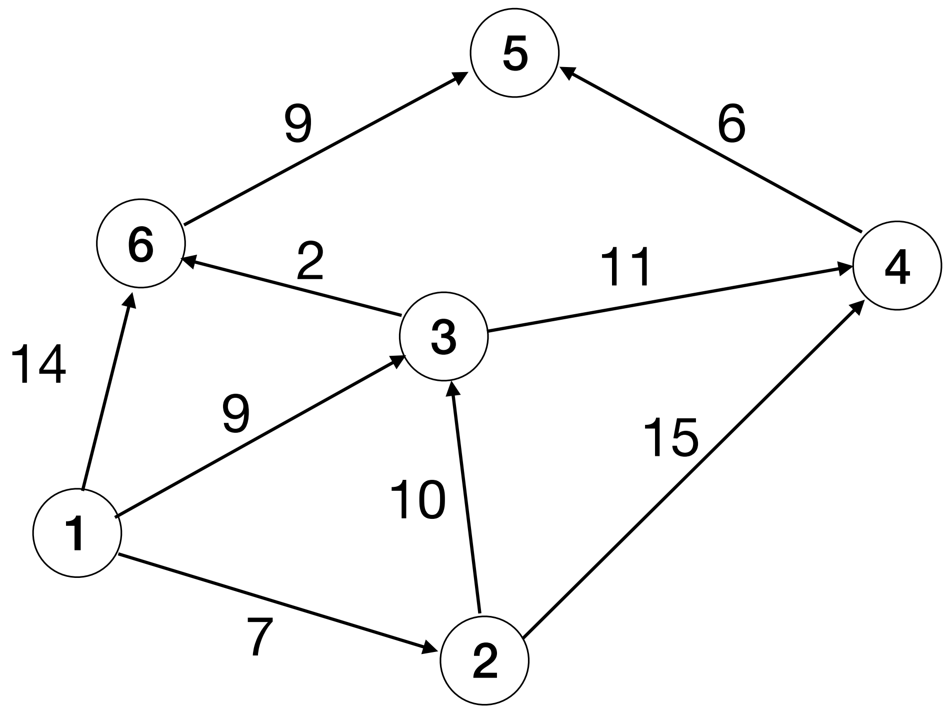
Dijkstra Practice

Run Dijkstra's algorithm on the graph starting at node 1.

F: ~~1~~ ~~2~~ ~~3~~ ~~6~~ ~~4~~ ~~5~~

S: 1 2 3 6 4 5

n	d	bp
1	0	null
2	7	1
3	9	1
4	~~22~~ 20	~~2~~ 3
5	20	6
6	~~14~~ 11	~~1~~ 3



Dijkstra Practice

Run Dijkstra's algorithm on the graph starting at node 1.

F: ~~1~~ ~~2~~ ~~3~~ ~~6~~ ~~4~~ ~~5~~

S: 1 2 3 6 4 5

n	d	bp
1	0	null
2	7	1
3	9	1
4	~~22~~ 20	~~2~~ 3
5	20	6
6	~~14~~ 11	~~1~~ 3