CSCI 241

Scott Wehrwein

Dijkstra's Algorithm:
Detailed Pseudocode
The Path Itself

Goals

Know how to augment Dijkstra's algorithm to keep backpointers in order to reconstruct the sequence of nodes in a shortest path.

Dijkstra's Shortest Paths: Pseudocode

```
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);
        }
    }
}
```

Dijkstra's Shortest Paths: Pseudocode

```
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);
        }
    }
}
```

```
Initialize Settled to empty
Initialize Frontier to the start node
```

Dijkstra's Shortest Paths: Pseudocode

```
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);
      }
   }
}
```

```
Initialize Settled to empty
Initialize Frontier to the start node

 While the frontier isn't empty:
   move node f with smallest d
    from F to S
```

Dijkstra's Shortest Paths: Pseudocode

```
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);
      }
   }
}
```

```
Initialize Settled to empty
Initialize Frontier to the start node

While the frontier isn't empty:
   move 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 path to w via f is shorter:
          update w's shortest path length
```

Dijkstra's Shortest Paths: Pseudocode

```
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);
      }
   }
}
```

```
Initialize Settled to empty
Initialize Frontier to the start node

While the frontier isn't empty:
   move 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 path to w via f is shorter:
          update w's shortest path length
```

What if we want to know the shortest path?

```
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);
        }
    }
}
```

At termination: for each reachable node n, n.d stores the **length** of the shortest path from v to n.

We didn't keep track of **how** to get from v to n!

What if we want to know the shortest path?

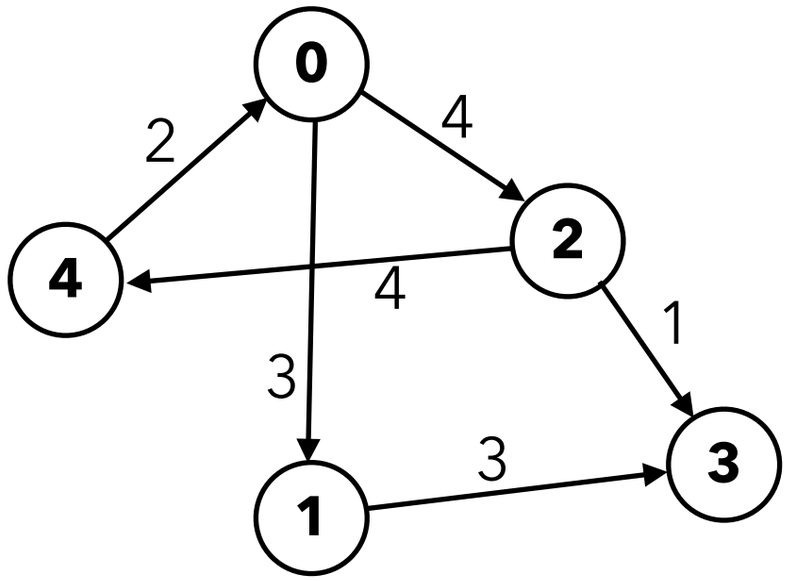
```
S = { }; F = {v};  v.d = 0; v.bp = null;
while  (F ≠ {})  {
    f = node in F with min d value;
    Remove f from F, add it to S;
    for each neighbor w of f {
        if (w not in S or F) {
            w.d =  f.d + weight(f, w);
            w.bp = f;
            add w to F;
        } else if (f.d+weight(f,w)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

Each node could store the full path, but that would be expensive to keep updated.

Strategy: maintain a **backpointer** at each node pointing to the previous node in the shortest path.

What if we want to know the shortest path? Example

```
S = { }; F = {v};  v.d = 0; v.bp = null;
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);
            w.bp = f;
            add w to F;
        } else if (f.d+weight(f,w) < w.d) {
            w.d = f.d+weight(f,w);
            w.bp = f
        }
    }
}
```

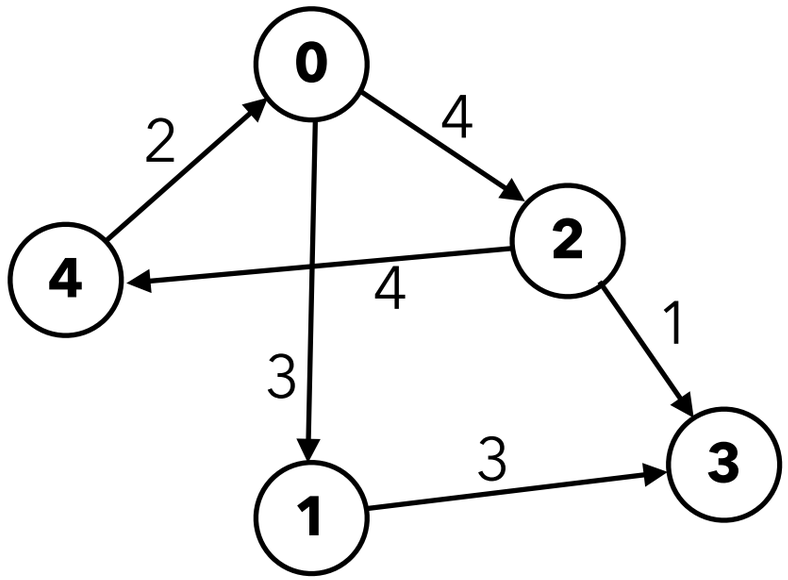


`shortest-paths(4)`

Node	d	bp
0	2	4
1	5	0
2	6	0
3	7	2
4	0	null

What if we want to know the shortest path? Example

```
S = { }; F = {v};  v.d = 0; v.bp = null;
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);
            w.bp = f;
            add w to F;
        } else if (f.d+weight(f,w) < w.d) {
            w.d = f.d+weight(f,w);
            w.bp = f
        }
    }
}
```



`shortest-paths(4)`

Shortest path to Node 3:

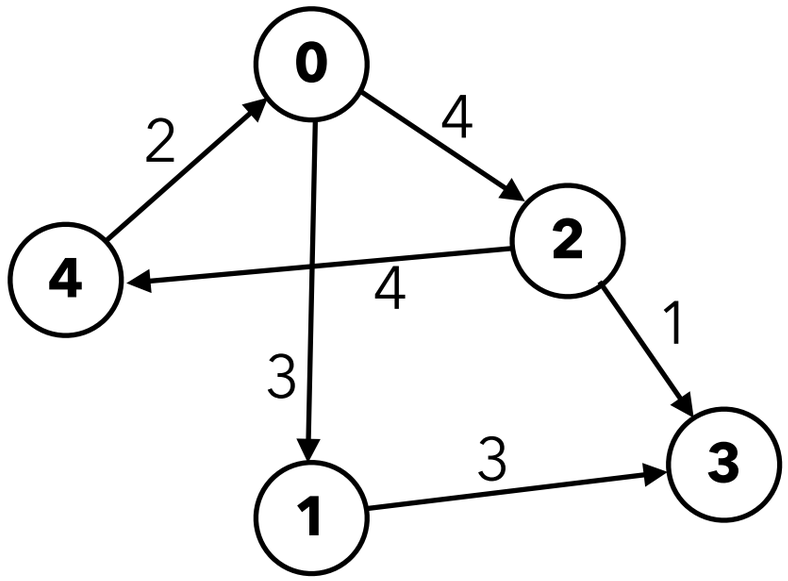
Node	d	bp
0	2	4
1	5	0
2	6	0
3	7	2
4	0	null

What if we want to know the shortest path? Example

```
S = { }; F = {v};  v.d = 0; v.bp = null;
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);
            w.bp = f;
            add w to F;
        } else if (f.d+weight(f,w) < w.d) {
            w.d = f.d+weight(f,w);
            w.bp = f
        }
    }
}
```

Shortest path to Node 3:

3



shortest-paths(4)

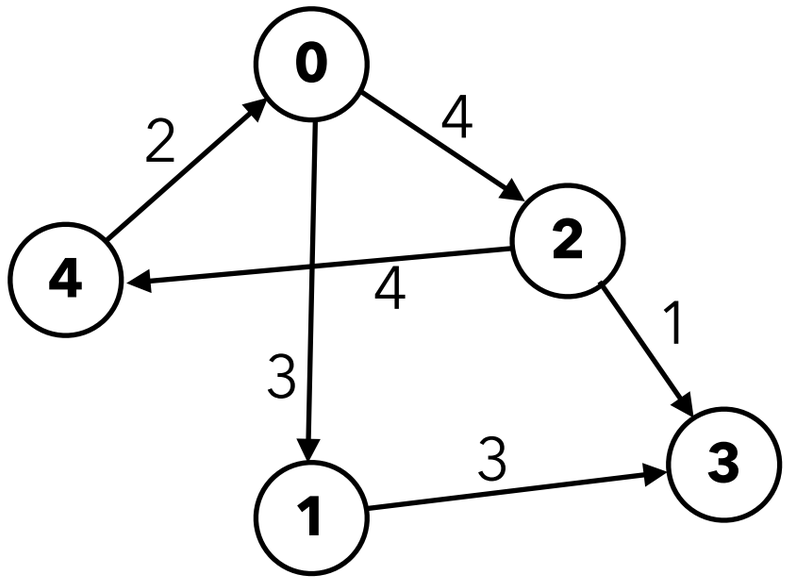
Node	d	bp
0	2	4
1	5	0
2	6	0
3	7	2
4	0	null

What if we want to know the shortest path? Example

```
S = { }; F = {v};  v.d = 0; v.bp = null;
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);
            w.bp = f;
            add w to F;
        } else if (f.d+weight(f,w) < w.d) {
            w.d = f.d+weight(f,w);
            w.bp = f
        }
    }
}
```

Shortest path to Node 3:

2 3



shortest-paths(4)

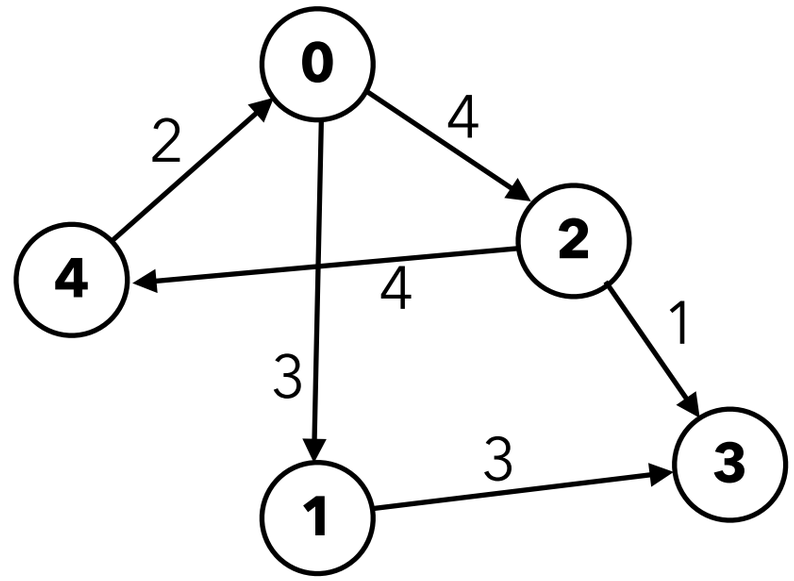
Node	d	bp
0	2	4
1	5	0
2	6	0
3	7	2
4	0	null

What if we want to know the shortest path? Example

```
S = { }; F = {v};  v.d = 0; v.bp = null;
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);
            w.bp = f;
            add w to F;
        } else if (f.d+weight(f,w) < w.d) {
            w.d = f.d+weight(f,w);
            w.bp = f
        }
    }
}
```

Shortest path to Node 3:

0 2 3



`shortest-paths(4)`

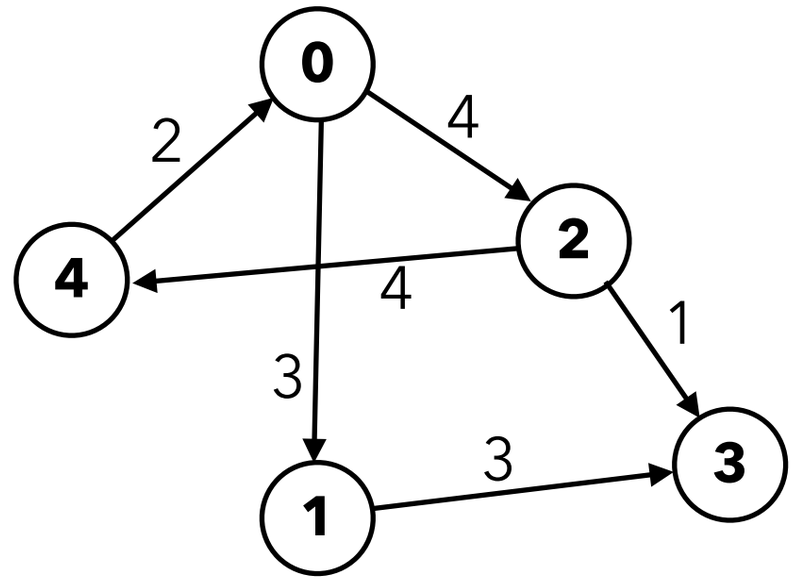
Node	d	bp
0	2	4
1	5	0
2	6	0
3	7	2
4	0	null

What if we want to know the shortest path? Example

```
S = { }; F = {v};  v.d = 0; v.bp = null;
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);
            w.bp = f;
            add w to F;
        } else if (f.d+weight(f,w) < w.d) {
            w.d = f.d+weight(f,w);
            w.bp = f
        }
    }
}
```

Shortest path to Node 3:

4 0 2 3



`shortest-paths(4)`

Node	d	bp
0	2	4
1	5	0
2	6	0
3	7	2
4	0	null