Minimum spanning trees

What is the lowest weight set of edges that connects all vertices of an undirected graph with positive weights?

Input: An undirected, positive weight graph, $G=(V,E)$

Output: A tree $T=(V,E')$ where $E' \subseteq E$ that minimizes

$$\text{weight}(T) = \sum_{e \in E'} w_e$$
Minimum cut property

Given a partition $S$, let edge $e$ be the minimum cost edge that crosses the partition. Every minimum spanning tree contains edge $e$.

Prim’s algorithm

Start at some root node and build out the MST by adding the lowest weighted edge out of the MST constructed so far.

Correctness of Prim’s?

Can we use the min-cut property?

- Given a partition $S$, let edge $e$ be the minimum cost edge that crosses the partition. Every minimum spanning tree contains edge $e$.

Let $S$ be the set of vertices visited so far.

The only time we add a new edge is if it’s the lowest weight edge from $S$ to $V-S$. 

Prim’s
Running time of Prim’s

PRIM(G, r)
1 for all v ∈ V
2 key[v] ← ∞
3 prec[v] ← null
4 heap[r] ← 0
5 H ← MAKEHEAP(heap)
6 while |E| > 0
7 u ← EXTRACT-MIN(H)
8 visited[u] ← true
9 for each edge (u, v) ∈ E
10 if visited[v] and key[v] < key[u]
11 key[v] ← key[u]
12 prec[v] ← u

Running time of Prim’s

PRIM(G, r)
1 for all v ∈ V
2 key[v] ← ∞
3 prec[v] ← null
4 heap[r] ← 0
5 H ← MAKEHEAP(heap)
6 while |E| > 0
7 u ← EXTRACT-MIN(H)
8 visited[u] ← true
9 for each edge (u, v) ∈ E
10 if visited[v] and key[v] < key[u]
11 key[v] ← key[u]
12 prec[v] ← u

1 MakeHeap | V ExtractMin | E DecreaseKey | Total
Array    θ(V)   O(V)   O(1)   O(V)
Bin heap  θ(V)   O(V log V) O(E log V) O((V+E) log V) O(E log V)
Fib heap  θ(V)   O(V log V) O(E)   O(V log V + E)

Kruskal’s: O(E log V)

When should we use Kruskal’s or Prim’s?

1 MakeHeap | V ExtractMin | E DecreaseKey | Total
Array    θ(V)   O(V)   O(1)   O(V)
Bin heap  θ(V)   O(V log V) O(E log V) O((V+E) log V) O(E log V)
Fib heap  θ(V)   O(V log V) O(E)   O(V log V + E)

Kruskal’s: O(E log V)

11
What is the shortest path from a to d?

How can we find this?

BFS

What is the shortest path from a to d?
### Shortest paths

What is the shortest path from a to d?

![Graph](https://via.placeholder.com/150)

### Shortest path algorithms?

What is \( \text{dist} \)?

What is \( \text{prev} \)?

How does it work?

What is the run-time?

How do we get the shortest path?

---

**Dijkstra's algorithm**

```plaintext
Dijkstra(G, s)
1. for all \( v \in V \)
2. \( \text{dist}[v] \leftarrow \infty \)
3. \( \text{prev}[v] \leftarrow \text{null} \)
4. \( \text{dist}[s] \leftarrow 0 \)
5. \( Q \leftarrow \text{MaxHeap}(V) \)
6. while \( \text{Heapify}(Q) \)
7. \( u \leftarrow \text{ExtractMin}(Q) \)
8. for all edges \((u, v) \in E\)
9. \[ \text{if} \quad \text{dist}[u] > \text{dist}[u] + w(u, v) \]
10. \( \text{dist}[v] \leftarrow \text{dist}[u] + w(u, v) \)
11. \( \text{DCreaseKey}(Q, v, \text{dist}[v]) \)
12. \( \text{prev}(v) \leftarrow u \)
```

---

**BFS(G, s)**

```plaintext
BFS(G, s)
1. for each \( v \in V \)
2. \( \text{dist}[v] \leftarrow \infty \)
3. \( \text{dist}[s] \leftarrow 0 \)
4. \( \text{dist}(s) \leftarrow 0 \)
5. \( \text{Queue}(Q) \)
6. while \( \text{Queue}(Q) \)
7. \( u \leftarrow \text{Queue}(Q) \)
8. for each edge \((u, v) \in E\)
9. \[ \text{if} \quad \text{dist}[u] = \infty \]
10. \( \text{dist}[v] \leftarrow \text{dist}[u] + 1 \)
11. \( \text{Enqueue}(Q, v) \)
12. \( \text{dist}(v) \leftarrow \text{dist}(u) + 1 \)
```
Dijkstra's algorithm

prev keeps track of the shortest path

Dijkstra(G, s)
1. for all v ∈ V
2. \( \text{dist}(v) = \infty \)
3. \( \text{prev}(v) = \text{null} \)
4. \( \text{Q} = \text{EmptyTree}(\text{V}) \)
5. while (\( \text{Q} \) ≠ \text{EmptyTree}()):
6. \( s = \text{ExtractMin}(\text{Q}) \)
7. for all edges (s, v) ∈ E:
8. if \( \text{dist}(v) > \text{dist}(s) + w(s, v) \)
9. \( \text{dist}(v) = \text{dist}(s) + w(s, v) \)
10. \( \text{DecreaseKey}(\text{Q}, v, \text{dist}(v)) \)
11. \( \text{prev}(v) = s \)
12. \( \text{dist}(s) = 0 \)

BFS(G, s)
1. for all s ∈ V
2. \( \text{dist}(s) = 0 \)
3. \( \text{prev}(s) = \text{null} \)
4. \( \text{Q} = \text{EmptyTree}(\text{V}) \)
5. while (\( \text{Q} \) ≠ \text{EmptyTree}()):
6. \( s = \text{ExtractMin}(\text{Q}) \)
7. for all edges (s, v) ∈ E:
8. if \( \text{dist}(v) = \infty \)
9. \( \text{dist}(v) = \text{dist}(s) + 1 \)
10. \( \text{Enqueue}(\text{Q}, v) \)
11. \( \text{prev}(v) = s \)
Dijkstra(G, e)
1 for all v ∈ V
2 dist[v] ← ∞
3 prev[v] ← null
4 dist[ε] ← 0
5 Q ← MaxHeap(V)
6 while !Empty(Q)
7 u ← ExtractMin(Q)
8 for all edges (u, v) ∈ E
9 if dist[u] > dist[v] + w(u, v)
10 Then dist[u] ← dist[v] + w(u, v)
11 DecreaseKey(Q, u, dist[u])
12 prev[v] ← u

Heap
A 0
B
C
D
E

Dijkstra(G, e)
1 for all v ∈ V
2 dist[v] ← ∞
3 prev[v] ← null
4 dist[ε] ← 0
5 Q ← MaxHeap(V)
6 while !Empty(Q)
7 u ← ExtractMin(Q)
8 for all edges (u, v) ∈ E
9 if dist[u] > dist[v] + w(u, v)
10 Then dist[v] ← dist[u] + w(u, v)
11 DecreaseKey(Q, v, dist[v])
12 prev[v] ← u

Heap
A 0
B
C
D
E

Dijkstra(G, e)
1 for all v ∈ V
2 dist[v] ← ∞
3 prev[v] ← null
4 dist[ε] ← 0
5 Q ← MaxHeap(V)
6 while !Empty(Q)
7 u ← ExtractMin(Q)
8 for all edges (u, v) ∈ E
9 if dist[u] > dist[v] + w(u, v)
10 Then dist[v] ← dist[u] + w(u, v)
11 DecreaseKey(Q, v, dist[v])
12 prev[v] ← u

Heap
A 0
B
C
D
E

Dijkstra(G, e)
1 for all v ∈ V
2 dist[v] ← ∞
3 prev[v] ← null
4 dist[ε] ← 0
5 Q ← MaxHeap(V)
6 while !Empty(Q)
7 u ← ExtractMin(Q)
8 for all edges (u, v) ∈ E
9 if dist[u] > dist[v] + w(u, v)
10 Then dist[v] ← dist[u] + w(u, v)
11 DecreaseKey(Q, v, dist[v])
12 prev[v] ← u

Heap
A 0
B
C
D
E

4/5/23
Dijkstra\((G, e)\)
1. for all \(v \in V\)
2. \(dist[v] = \infty\)
3. \(prev[v] = \text{null}\)
4. \(Q := \text{MaxHeapsort}(V)\)
5. while \(\text{Empty}(Q)\)
6. \(u := \text{ExtractMin}(Q)\)
7. for all edges \((u, v) \in E\)
8. if \(dist[u] > dist[v] + w(u,v)\)
9. \(dist[u] := dist[u] + w(u,v)\)
10. \(prev[u] := u\)

Heap

\[\begin{array}{cccc}
\text{B} & \infty \\
\text{C} & \infty \\
\text{D} & \infty \\
\text{E} & \infty
\end{array}\]

### Heap 29

\[\begin{array}{cccc}
\text{B} & \infty \\
\text{C} & \infty \\
\text{D} & \infty \\
\text{E} & \infty
\end{array}\]

### Heap 30

\[\begin{array}{cccc}
\text{C} & 1 \\
\text{B} & \infty \\
\text{D} & \infty \\
\text{E} & \infty
\end{array}\]

### Heap 31

\[\begin{array}{cccc}
\text{B} & \infty \\
\text{C} & \infty \\
\text{D} & \infty \\
\text{E} & \infty
\end{array}\]

### Heap 32

\[\begin{array}{cccc}
\text{C} & 1 \\
\text{B} & 3 \\
\text{D} & \infty \\
\text{E} & \infty
\end{array}\]
**Dijkstra(G, e)**

1. for all $v \in V$
2. $dist[v] \leftarrow \infty$
3. $prev[v] \leftarrow \text{null}$
4. $dist[] \leftarrow 0$
5. $Q \leftarrow \text{MaxHeap}[V]$
6. while $\text{Empty}(Q)$
7. $u \leftarrow \text{ExtractMin}(Q)$
8. for all edges $(u, v) \in E$
9. \quad if $dist[u] > dist[v] + w(u, v)$
10. \quad $dist[v] \leftarrow dist[u] + w(u, v)$
11. \quad DecreaseKey($Q$, $v$, $dist[v]$)
12. \quad $prev[v] \leftarrow u$

**Heap**

- C 3
- B $\infty$
- D $\infty$
- E $\infty$

---

**Dijkstra(G, e)**

1. for all $v \in V$
2. $dist[v] \leftarrow \infty$
3. $prev[v] \leftarrow \text{null}$
4. $dist[] \leftarrow 0$
5. $Q \leftarrow \text{MaxHeap}[V]$
6. while $\text{Empty}(Q)$
7. $u \leftarrow \text{ExtractMin}(Q)$
8. for all edges $(u, v) \in E$
9. \quad if $dist[u] > dist[v] + w(u, v)$
10. \quad $dist[v] \leftarrow dist[u] + w(u, v)$
11. \quad DecreaseKey($Q$, $v$, $dist[v]$)
12. \quad $prev[v] \leftarrow u$

**Heap**

- B 3
- D $\infty$
- E $\infty$

---

**Dijkstra(G, e)**

1. for all $v \in V$
2. $dist[v] \leftarrow \infty$
3. $prev[v] \leftarrow \text{null}$
4. $dist[] \leftarrow 0$
5. $Q \leftarrow \text{MaxHeap}[V]$
6. while $\text{Empty}(Q)$
7. $u \leftarrow \text{ExtractMin}(Q)$
8. for all edges $(u, v) \in E$
9. \quad if $dist[u] > dist[v] + w(u, v)$
10. \quad $dist[v] \leftarrow dist[u] + w(u, v)$
11. \quad DecreaseKey($Q$, $v$, $dist[v]$)
12. \quad $prev[v] \leftarrow u$

**Heap**

- B 3
- D $\infty$
- E $\infty$
Dijkstra(G, s):
1 for all v ∈ V
2 nearest[0] = −∞
3 nearest[s] = null
4 dist[s] = 0
5 Q = MinHeap(V)
6 while (Empty(Q))
7 u = ExtractMin(Q)
8 for all edges (u, v) ∈ E
9 if dist[u] > dist[v] + w(u, v)
10 dist[v] = dist[u] + w(u, v)
11 DecreaseKey(Q, v, dist[v])
12 nearest[v] = u

Heap

B 2
D 5
E 3

Dijkstra(G, s):
1 for all v ∈ V
2 nearest[0] = −∞
3 nearest[s] = null
4 dist[s] = 0
5 Q = MinHeap(V)
6 while (Empty(Q))
7 u = ExtractMin(Q)
8 for all edges (u, v) ∈ E
9 if dist[u] > dist[v] + w(u, v)
10 dist[v] = dist[u] + w(u, v)
11 DecreaseKey(Q, v, dist[v])
12 nearest[v] = u

Heap

B 2
D 5
E 3

Heap

B 2
D 5
E 3

Heap

B 2
D 5
E 3
**Dijkstra(G, s)**
1. for all $v \in V$
2. $\text{dist}(v) = \infty$
3. $\text{prev}(v) = \text{null}$
4. $\text{dist}(s) = 0$
5. $Q = \text{MaxHeap}(V)$
6. while $\text{Empty}(Q)$
7. $u = \text{ExtractMin}(Q)$
8. for all edges $(u, v) \in E$
9. $\text{if dist}(u) + w(u, v) < \text{dist}(v)$
10. $\text{dist}(v) = \text{dist}(u) + w(u, v)$
11. $\text{DecreaseKey}(Q, v, \text{dist}(v))$
12. $\text{prev}(v) = u$

**Heap**

41. Heap
   - A
   - B
   - C
   - D
   - E

42. Heap
   - A
   - B
   - C
   - D
   - E

**Dijkstra(G, a)**
1. for all $v \in V$
2. $\text{dist}(v) = \infty$
3. $\text{prev}(v) = \text{null}$
4. $\text{dist}(a) = 0$
5. $Q = \text{MaxHeap}(V)$
6. while $\text{Empty}(Q)$
7. $u = \text{ExtractMin}(Q)$
8. for all edges $(u, v) \in E$
9. $\text{if dist}(u) + w(u, v) < \text{dist}(v)$
10. $\text{dist}(v) = \text{dist}(u) + w(u, v)$
11. $\text{DecreaseKey}(Q, v, \text{dist}(v))$
12. $\text{prev}(v) = u$

**Heap**

43. Heap
   - A
   - B
   - C
   - D
   - E

44. Heap
   - A
   - B
   - C
   - D
   - E

43. How do we get the actual paths?
Is Dijkstra’s algorithm correct?

Invariant: For every vertex removed from the heap, dist[v] is the actual shortest distance from s to v

```
Dijkstra(G, s)
1 for all v ∈ V
2 dist[s] ← ∞
3 prev[s] ← null
4 dist[s] ← 0
5 Q ← MakeHeap(\{s\})
6 while Q ≠ \emptyset
7 a ← ExtractMin(Q)
8 for all edges (a, v) ∈ E
9 if dist[v] > dist[a] + w(a, v)
10 dist[v] ← dist[a] + w(a, v)
11 DecreaseKey(Q, a, dist[v])
12 prev[v] ← a
```

proof?

The only time a vertex gets visited is when the distance from s to that vertex is smaller than the distance to any remaining vertex.

Therefore, there cannot be any other path that hasn’t been visited already that would result in a shorter path.

Running time?

```
Dijkstra(G, s)
1 for all v ∈ V
2 dist[s] ← ∞
3 prev[s] ← null
4 dist[s] ← 0
5 Q ← MakeHeap(\{s\})
6 while Q ≠ \emptyset
7 a ← ExtractMin(Q)
8 for all edges (a, v) ∈ E
9 if dist[v] > dist[a] + w(a, v)
10 dist[v] ← dist[a] + w(a, v)
11 DecreaseKey(Q, a, dist[v])
12 prev[v] ← a
```

1 call to MakeHeap
Running time?

**Dijkstra(G, s)**
1. for all \( v \in V \)
   2. \( d[v] \leftarrow \infty \)
   3. \( prev[v] \leftarrow \text{null} \)
   4. \( d[s] \leftarrow 0 \)
5. \( Q \leftarrow \text{MakeHeap}(V) \)
6. while \( \text{NotEmpty}(Q) \)
   7. \( u \leftarrow \text{ExtractMin}(Q) \)
   8. for all edges \((u, v) \in E\)
     9. if \( d[u] > d[v] + w(u, v) \)
       10. \( d[v] \leftarrow d[u] + w(u, v) \)
       11. \( \text{DecreaseKey}(Q, u, d[v]) \)
       12. \( prev[v] \leftarrow u \)

**Array**
- \( O(|V|) \) calls
- \( O(|V|^2) \) calls
- \( O(|E|) \) calls
- \( O(|V|^2) \) total

**Bin heap**
- \( O(|V|) \) calls
- \( O(|V| \log |V|) \) calls
- \( O(|E| \log |V|) \) calls
- \( O(|V| + |E| \log |V|) \) total

### Running time?

Depends on the heap implementation

|                | \( O(|V|) \) | \( O(|V|^2) \) | \( O(|E|) \) | \( O(|V|^2) \) |
|----------------|-------------|-------------|-------------|-------------|
| **Array**      |             |             |             |             |
| **Bin heap**   |             |             |             |             |

### Running time?

**Dijkstra(G, s)**
1. for all \( v \in V \)
   2. \( d[v] \leftarrow \infty \)
   3. \( prev[v] \leftarrow \text{null} \)
   4. \( d[s] \leftarrow 0 \)
5. \( Q \leftarrow \text{MakeHeap}(V) \)
6. while \( \text{NotEmpty}(Q) \)
   7. \( u \leftarrow \text{ExtractMin}(Q) \)
   8. for all edges \((u, v) \in E\)
     9. if \( d[u] > d[v] + w(u, v) \)
       10. \( d[v] \leftarrow d[u] + w(u, v) \)
       11. \( \text{DecreaseKey}(Q, u, d[v]) \)
       12. \( prev[v] \leftarrow u \)
Running time?

Depends on the heap implementation

<table>
<thead>
<tr>
<th>Operation</th>
<th>Array</th>
<th>Bin heap</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>MakeHeap</td>
<td>$O(</td>
<td>V</td>
<td>)$</td>
</tr>
<tr>
<td>ExtractMin</td>
<td>$O(</td>
<td>V</td>
<td>^2)$</td>
</tr>
<tr>
<td>DecreaseKey</td>
<td>$O(</td>
<td>V</td>
<td>^2)$</td>
</tr>
<tr>
<td>Total</td>
<td>$O(</td>
<td>V</td>
<td>^2)$</td>
</tr>
</tbody>
</table>

Is this an improvement? If $|E| < |V|^2 / \log |V|$
What about Dijkstra’s on…?

Dijkstra’s algorithm only works for positive edge weights

Is Dijkstra’s algorithm correct?

Invariant: For every vertex removed from the heap, dist[v] is the actual shortest distance from s to v

- The only time a vertex gets visited is when the distance from s to that vertex is smaller than the distance to any remaining vertex
- Therefore, there cannot be any other path that hasn’t been visited already that would result in a shorter path

We relied on having positive edge weights for correctness!

Bounding the distance

Another invariant: For each vertex v, dist[v] is an upper bound on the actual shortest distance

```
for all v ∈ V:
    dist[v] = ∞

pre(v) = ∞

Q = MAKEHEAP(V)

while NOT EMPTY(Q):
    u = EXTRACTMIN(Q)
    for all edges (u, v) ∈ E:
        if dist[u] + w(u, v) < dist[v]:
            dist[v] = dist[u] + w(u, v)
            pre[v] = u
```

Is this a valid invariant?

Bounding the distance

Another invariant: For each vertex v, dist[v] is an upper bound on the actual shortest distance

- start off at ∞
- only update the value if we find a shorter distance

An update procedure

\[ \text{dist}[v] = \min \{ \text{dist}[v], \text{dist}[u] + w(u, v) \} \]
Can we ever go wrong applying this update rule?
- We can apply this rule as many times as we want and will never underestimate \( \text{dist}[v] \).

When will \( \text{dist}[v] \) be right?
- If \( u \) is along the shortest path to \( v \) and \( \text{dist}[u] \) is correct.

\[
\text{dist}[v] = \min \{ \text{dist}[v], \text{dist}[u] + w(u,v) \}
\]

What happens if we update all of the vertices with the above update?

Consider the shortest path from \( s \) to \( v \):

\[
\text{dist}[v] \text{ will be right if } u \text{ is along the shortest path to } v \text{ and } \text{dist}[u] \text{ is correct}
\]
$dist[v] = \min\{dist[v], dist[u] + w(u,v)\}$

$dist[v]$ will be right if $u$ is along the shortest path to $v$ and $dist[u]$ is correct

What happens if we update all of the vertices with the above update?

$\min\{v, u, w\} + dist[v] = s$

How many times do we have to do this for vertex $p_i$ to have the correct shortest path from $s$?

- 1 times

$dist[v] = \min\{dist[v], dist[u] + w(u,v)\}$

$dist[v]$ will be right if $u$ is along the shortest path to $v$ and $dist[u]$ is correct

Does the order that we update the vertices matter?

$\min\{v, u, w\} + dist[v] = s$
dist[v] will be right if u is along the shortest path to v and dist[u] is correct.

How many times do we have to do this for vertex p_i to have the correct shortest path from s?

- i times

What is the longest (vertex-wise) the path from s to any node v can be?

- \(|V| - 1\) edges/vertices
Bellman-Ford algorithm

Bellman-Ford(G, s)
1 for all v ∈ V
2 dist[v] ← ∞
3 prev[v] ← null
4 dist[s] ← 0
5 for i = 1 to |V| − 1
6 for all edges (u, v) ∈ E
7 if dist[u] > dist[v] + w(u, v)
8 dist[v] ← dist[u] + w(u, v)
9 prev[v] ← u
10 for all edges (u, v) ∈ E
11 if dist[u] > dist[v] + w(u, v)
12 return false

Bellman-Ford algorithm

Bellman-Ford(G, s)
1 for all v ∈ V
2 dist[v] ← ∞
3 prev[v] ← null
4 dist[s] ← 0
5 for i = 1 to |V| − 1
6 for all edges (u, v) ∈ E
7 if dist[u] > dist[v] + w(u, v)
8 dist[v] ← dist[u] + w(u, v)
9 prev[v] ← u
10 for all edges (u, v) ∈ E
11 if dist[u] > dist[v] + w(u, v)
12 return false

Negative cycles

What is the shortest path from a to e?
Bellman-Ford algorithm

Bellman-Ford(G, s)
1 for all v ∈ V
2 dist[v] ← ∞
3 pre[v] ← null
4 dist[s] ← 0
5 for i = 1 to |V| - 1
6 for all edges (u, v) ∈ E
7 if dist[u] > dist[v] + w(u, v)
8 dist[v] ← dist[u] + w(u, v)
9 pre[v] ← u
10 return false

How many edges is the shortest path from s to:

A: 3
B: 8
C: 7
D: 6
How many edges is the shortest path from s to:

A: 3
B: 5
D: 7
Bellman-Ford algorithm

Iteration: 1

Iteration: 2

A has the correct distance and path

Iteration: 3

Iteration: 4
Bellman-Ford algorithm

Iteration: 5

B has the correct distance and path

Iteration: 6

D (and all other nodes) have the correct distance and path

Correctness of Bellman-Ford

Loop invariant: After iteration $i$, all vertices with shortest paths from $s$ of length $i$ edges or less have correct distances

```
Bellman-Ford(G, s)
1   for all v ∈ V
2      dist[v] ← ∞
3      prev[v] ← null
4      dist[s] ← 0
5   for i ← 1 to |V| − 1
6       for all edges $(u,v) ∈ E$
7           if dist[u] > dist[v] + w(u,v)
8               dist[u] ← dist[v] + w(u,v)
9               prev[u] ← v
10    for all edges $(u,v) ∈ E$
11       if dist[u] > dist[v] + w(u,v)
12       return false
```
Runtime of Bellman-Ford

```plaintext
BELLMAN-FORD(G, s)
1 for all v ∈ V
2 dist[v] ← ∞
3 prev[v] ← null
4 dist[s] ← 0
5 for i ← 1 to |V| − 1
6 for all edges (u, v) ∈ E
7 if dist[i] > dist[u] + w(u, v)
8 dist[v] ← dist[u] + w(u, v)
9 prev[v] ← u
10 for all edges (u, v) ∈ E
11 if dist[v] > dist[u] + w(u, v)
12 return false

O(|V| |E|)
```

Can you modify the algorithm to run faster (in some circumstances)?

Shortest Paths

What is the shortest path from A to E?

Shortest Paths

What is the shortest path from A to E?
What algorithm would we use to calculate this?

- Bellman-Ford (since the graph has negative edges)
- O(VE)

Called a single-source shortest path algorithm. Why?
What is the shortest path from A to C? If we already calculated A to E using Bellman-Ford do we need to do any work?

No new calculations! Bellman-Ford calculates all shortest paths starting at A.

What is the shortest path from D to C? If we already calculated A to E using Bellman-Ford do we need to do any work?

Different source. Have to run Bellman-Ford again!
All pairs shortest paths:
calculate the shortest paths between all vertices

Running time (in terms of E and V)?
- Bellman-Ford: $O(VE)$
- $V$ calls, one for each vertex
Floyd-Warshall: key idea

Label all vertices with a number from 1 to V

\[ d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \]
using only vertices \{1, 2, ..., k\}

What is \(d_{15}^2\)?

What is \(d_{41}^4\)?

\[ d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \]
using only vertices \{1, 2, ..., k\}

\(d_{15}^2 = 9\). Can only use 2.
Floyd-Warshall: key idea

\[ d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \]
using only vertices \{1, 2, ..., k\}

What is \(d_{15}^2\)?

Floyd-Warshall: key idea

\[ d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \]
using only vertices \{1, 2, ..., k\}

\(d_{13}^3 = 1\). Can't use vertex 4.

Floyd-Warshall: key idea

\[ d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \]
using only vertices \{1, 2, ..., k\}

What is \(d_{41}^4\)?

Floyd-Warshall: key idea

\[ d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \]
using only vertices \{1, 2, ..., k\}

\(d_{41}^4 = \infty\). No possible path.
What is $d_{33}^5$?

$d_{i,j}^k = \text{shortest path from vertex } i \text{ to vertex } j \text{ using only vertices } \{1, 2, \ldots, k\}$

If we want all possibilities, how many values are there (i.e. what is the size of $d_{i,j}^k$)?

$V^3$

• $i$: all vertices
• $j$: all vertices
• $k$: all vertices
Floyd-Warshall: key idea

Label all vertices with a number from 1 to V

\[ d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \]
using only vertices \{1, 2, ..., k\}

What is \( d_{ij}^r \)?

- Distance of the shortest path from \( i \) to \( j \)
- If we can calculate this, for all \((i, j)\), we’re done!

Recursive relationship

\[ d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \]
using only vertices \{1, 2, ..., k\}

Assume we know \( d_{ij}^k \)

How can we calculate \( d_{ij}^{k+1} \), i.e. shortest path now including vertex \( k+1 \)? (Hint: in terms of \( d_{ij}^k \))

Two options:
1) Vertex \( k+1 \) doesn’t give us a shorter path
2) Vertex \( k+1 \) does give us a shorter path

\[ d_{ij}^{k+1} = \]
Recursive relationship

\[ d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \]

using only vertices \( \{1, 2, \ldots, k\} \)

Two options:
1) Vertex \( k+1 \) doesn't give us a shorter path
2) Vertex \( k+1 \) does give us a shorter path

\[ d_{ij}^{k+1} = ? \]

125

Recursive relationship

\[ d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \]

using only vertices \( \{1, 2, \ldots, k\} \)

Two options:
1) Vertex \( k+1 \) doesn't give us a shorter path
2) Vertex \( k+1 \) does give us a shorter path

\[ d_{ij}^{k+1} = ? \]

126

Recursive relationship

\[ d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \]

using only vertices \( \{1, 2, \ldots, k\} \)

Two options:
1) Vertex \( k+1 \) doesn't give us a shorter path
2) Vertex \( k+1 \) does give us a shorter path

\[ d_{ij}^{k+1} = ? \]

127

Recursive relationship

\[ d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \]

using only vertices \( \{1, 2, \ldots, k\} \)

Two options:
1) Vertex \( k+1 \) doesn't give us a shorter path
2) Vertex \( k+1 \) does give us a shorter path

\[ d_{ij}^{k+1} = ? \]

128
Recursive relationship

\[ d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \]
using only vertices \( \{1, 2, \ldots, k\} \)

Two options:
1) Vertex \( k+1 \) doesn’t give us a shorter path
2) Vertex \( k+1 \) does give us a shorter path

\[ d_{ij}^{k+1} = \min(d_{ij}^k, d_{i((k+1)j)}^k + d_{(k+1)j}^k) \]

Pick whichever is shorter

Floyd-Warshall

Calculate \( d_{ij}^k \) for increasing \( k \), i.e. \( k = 1 \) to \( V \)

Floyd-Warshall \((G = (V,E,W))\):

\[ d^0 = W \quad \text{// initialize with edge weights} \]

for \( k = 1 \) to \( V \)

for \( i = 1 \) to \( V \)

for \( j = 1 \) to \( V \)

\[ d_{ij}^k = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]

return \( d^V \)

<table>
<thead>
<tr>
<th>( k = 0 )</th>
<th>( k = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( \begin{bmatrix} 0 &amp; 4 &amp; -1 &amp; \infty &amp; \infty \ 2 &amp; \infty &amp; \infty &amp; \infty &amp; 5 \ 3 &amp; \infty &amp; 3 &amp; 0 &amp; 2 \ 4 &amp; \infty &amp; \infty &amp; 0 &amp; -3 \ 5 &amp; \infty &amp; 1 &amp; \infty &amp; 0 \end{bmatrix} )</td>
<td>1 ( \begin{bmatrix} 0 &amp; 4 &amp; -1 &amp; \infty &amp; \infty \ 2 &amp; \infty &amp; \infty &amp; \infty &amp; 5 \ 3 &amp; \infty &amp; 3 &amp; 0 &amp; 2 \ 4 &amp; \infty &amp; \infty &amp; 0 &amp; -3 \ 5 &amp; \infty &amp; 1 &amp; \infty &amp; 0 \end{bmatrix} )</td>
</tr>
</tbody>
</table>

adjacency matrix | no change
### Floyd-Warshall \((G = (V,E,W))\)

1. **Dijkstra's Algorithm**
   - Initialize with edge weights:
   
   \[
   d^0 = W \quad // \text{initialize with edge weights}
   \]
   - For \(k = 1\) to \(V\):
   - \(i = 1\) to \(V\):
   - \(j = 1\) to \(V\):
   - \(d_{ij} = \min(d_{ij} + d_{ik} + d_{kj})\)

   
   \[
   k = 1 \\
   \begin{array}{cccc|c}
   i & 1 & 2 & 3 & 4 \\
   \hline
   1 & 0 & 4 & \infty & \infty \\
   2 & \infty & 0 & \infty & \infty \\
   3 & \infty & 3 & 0 & 2 \\
   4 & \infty & \infty & 0 & -3 \\
   5 & \infty & 1 & \infty & 0 \\
   \end{array}
   \]

   \[
   k = 2 \\
   \begin{array}{cccc|c}
   i & 1 & 2 & 3 & 4 \\
   \hline
   1 & 0 & 4 & -1 & \infty \\
   2 & \infty & 0 & \infty & 9 \\
   3 & \infty & 3 & 0 & 2 \\
   4 & \infty & \infty & 0 & 4 \\
   5 & \infty & 1 & \infty & 5 \\
   \end{array}
   \]

2. **Found a shorter path!**

3. **Minimum Path**

4. **Return \(d^f\)**

\[
\text{return } d^f
\]

\[
\begin{array}{cccc|c}
k = 2 & 1 & 2 & 3 & 4 \\
\hline
1 & 0 & 4 & -1 & \infty \\
2 & \infty & 0 & \infty & 9 \\
3 & \infty & 3 & 0 & 2 \\
4 & \infty & \infty & 0 & 4 \\
5 & \infty & 1 & \infty & 5 \\
\end{array}
\]

\[
\begin{array}{cccc|c}
k = 3 & 1 & 2 & 3 & 4 \\
\hline
1 & 0 & 4 & -1 & \infty \\
2 & \infty & 0 & \infty & 9 \\
3 & \infty & 3 & 0 & 2 \\
4 & \infty & \infty & 0 & 4 \\
5 & \infty & 1 & \infty & 5 \\
\end{array}
\]
Floyd-Warshall (G = (V, E, W)):
\[ d^k = W \] // initialize with edge weights
for \( k = 1 \) to \( V \)
for \( i = 1 \) to \( V \)
for \( j = 1 \) to \( V \)
    \[ d_{ij} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]
return \( d^k \)

Floyd-Warshall (G = (V, E, W)):
\[ d^k = W \] // initialize with edge weights
for \( k = 1 \) to \( V \)
for \( i = 1 \) to \( V \)
for \( j = 1 \) to \( V \)
    \[ d_{ij} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]
return \( d^k \)
Floyd–Warshall (G = (V, E, W)):
\( d^0 \equiv W \) // initialize with edge weights
for \( k = 1 \) to V
for \( i = 1 \) to V
for \( j = 1 \) to V
\[ d_{ij} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]
return \( d^k \)

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 4 & -1 & \infty & 9 \\
2 & \infty & 0 & \infty & \infty & 5 \\
3 & \infty & 3 & 0 & 2 & 2 \\
4 & \infty & \infty & 0 & -3 & 4 \\
5 & \infty & \infty & 1 & \infty & 0 \\
\end{array}
\]

minimum

Found a shorter path!

Floyd–Warshall (G = (V, E, W)):
\( d^0 \equiv W \) // initialize with edge weights
for \( k = 1 \) to V
for \( i = 1 \) to V
for \( j = 1 \) to V
\[ d_{ij} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]
return \( d^k \)

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 4 & -1 & \infty & 9 \\
2 & \infty & 0 & \infty & \infty & 5 \\
3 & \infty & 3 & 0 & 2 & 2 \\
4 & \infty & \infty & 0 & -3 & 4 \\
5 & \infty & \infty & 1 & \infty & 0 \\
\end{array}
\]

minimum

Found a shorter path!
Floyd-Warshall(G = (V,E,W)):
\[ d^0 \equiv W \] // initialize with edge weights
for \( k = 1 \) to V
for \( i = 1 \) to V
for \( j = 1 \) to V
\[ d_{ij} = \min(d_{ij}, d_{ik} + d_{kj}) \]
return \( d^k \)

Floyd-Warshall(G = (V,E,W)):
\[ d^0 \equiv W \] // initialize with edge weights
for \( k = 1 \) to V
for \( i = 1 \) to V
for \( j = 1 \) to V
\[ d_{ij} = \min(d_{ij}, d_{ik} + d_{kj}) \]
return \( d^k \)

minimum

Found a shorter path!
Floyd-Warshall analysis

Is it correct?

Any assumptions?

Assuming the graph has no negative cycles!

What happens if there is a negative cycle?
If the graph has a negative weight cycle, at the end, at least one of the diagonal entries will be a negative number, i.e., we there’s a way to get back to a vertex using all of the vertices that results in a negative weight.

\[
\begin{pmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 2 & -1 & 1 \\
2 & 0 & 2 & 7 & 9 \\
3 & \infty & 2 & 0 & 2 \\
4 & \infty & -2 & \infty & -3 \\
5 & \infty & \infty & \infty & 0
\end{pmatrix}
\]

Run-time:

\[
\text{Floyd-Warshall}(G = (V,E,W)) \]

\[
d^0 = W \quad \text{// initialize with edge weights}
\]

\[
\text{for } k = 1 \text{ to } V \\
\text{for } i = 1 \text{ to } V \\
\text{for } j = 1 \text{ to } V \\
d_{i,j}^k = \min(d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1})
\]

return \(d^V\)

What type of algorithm is Floyd-Warshall?

\[
\text{Floyd-Warshall}(G = (V,E,W)) \]

\[
d^0 = W \quad \text{// initialize with edge weights}
\]

\[
\text{for } k = 1 \text{ to } V \\
\text{for } i = 1 \text{ to } V \\
\text{for } j = 1 \text{ to } V \\
d_{i,j}^k = \min(d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1})
\]

return \(d^V\)
Floyd-Warshall analysis

Dynamic programming!!
Build up solutions to larger problems using solutions to smaller problems. Use a table to store the values.

Floyd-Warshall\(G = (V,E,W)\):
\[d^0 = W\] // initialize with edge weights
for \(k = 1\) to \(V\)
  for \(i = 1\) to \(V\)
    for \(j = 1\) to \(V\)
      \(d_{ijk} = \min(d_{ij}^k, d_{ik}^{k-1} + d_{kj}^{k-1})\)
    return \(d^V\)

Floyd-Warshall: key idea

Label all vertices with a number from 1 to \(V\)

\[d_{ijk} = \text{shortest path from vertex } i \text{ to vertex } j \]
using only vertices \(\{1, 2, ..., k\}\)

If we want all possibilities, how many values are there (i.e. what is the size of \(d_{i/k}\)?)

Space usage?

Floyd-Warshall\(G = (V,E,W)\):
\[d^0 = W\] // initialize with edge weights
for \(k = 1\) to \(V\)
  for \(i = 1\) to \(V\)
    for \(j = 1\) to \(V\)
      \(d_{ijk} = \min(d_{ij}^k, d_{ik}^{k-1} + d_{kj}^{k-1})\)
    return \(d^V\)

Can we do better?

\(V^3\)
- \(i\): all vertices
- \(j\): all vertices
- \(k\): all vertices
Floyd-Warshall analysis

Space usage: $\Theta(V^2)$

Only need the current value and the previous

Floyd-Warshall($G = (V, E, W)$):

\[ d^0 = W \quad // \text{initialize with edge weights} \]
\[ \text{for } k = 1 \text{ to } V \]
\[ \quad \text{for } i = 1 \text{ to } V \]
\[ \quad \text{for } j = 1 \text{ to } V \]
\[ \quad d_{ij} = \min(d_{ij}, d_{ik} + d_{kj}) \]
\[ \text{return } d^{V} \]

All pairs shortest paths

V * Bellman-Ford: $O(V^2E)$

Floyd-Warshall: $\Theta(V^3)$

All pairs shortest paths

All pairs shortest paths for positive weight graphs:
calculate the shortest paths between all points

Easy solution?

All pairs shortest paths

All pairs shortest paths for positive weight graphs:
calculate the shortest paths between all points

Run Dijkstra’s from each vertex!

Running time (in terms of $E$ and $V$)?
All pairs shortest paths

All pairs shortest paths for positive weight graphs:
calculate the shortest paths between all points

Run Dijkstra's from each vertex!

$O(V^2 \log V + V E)$

- $V$ calls do Dijkstra's
- Dijkstra's: $O(V \log V + E)$

165

All pairs shortest paths

V * Bellman-Ford: $O(V^2 E)$

Floyd-Warshall: $\Theta(V^3)$

V * Dijkstra: $O(V^2 \log V + V E)$

Is this any better?

166

All pairs shortest paths

V * Bellman-Ford: $O(V^2 E)$

Floyd-Warshall: $\Theta(V^3)$

V * Dijkstra: $O(V^2 \log V + V E)$

If the graph is sparse!

167

All pairs shortest paths

All pairs shortest paths for positive weight graphs:
calculate the shortest paths between all points

Run Dijkstra's from each vertex!

Challenge: Dijkstra's assumes positive weights

168
Johnson's: key idea

Reweight the graph to make all edges positive such that shortest paths are preserved

A → B → C → D → E

-1 2 3 1
1 0 4 3 2
4

Lemma

Let $h$ be any function mapping a vertex to a real value

If we change the graph weights as:

$$ \hat{w}(u, v) = w(u, v) + h(u) - h(v) $$

The shortest paths are preserved

Lemma: proof

Let $s, v_1, v_2, ..., v_k, t$ be a path from $s$ to $t$

The weight in the reweighted graph is:

$$ \hat{w}(s, v_1, ..., v_k, t) = w(s, v_1, ..., v_k, t) + h(s) - h(t) $$

Claim: the weight change preserves shortest paths, i.e. if a path was the shortest from $s$ to $t$ in the original graph it will still be the shortest path from $s$ to $t$ in the new graph.

Lemma: proof

$$ \hat{w}(s, v_1, ..., v_k, t) = w(s, v_1, ..., v_k, t) + h(s) - h(t) $$

Claims the weight change preserves shortest paths, i.e. if a path was the shortest from $s$ to $t$ in the original graph it will still be the shortest path from $s$ to $t$ in the new graph.

Justification?
Lemma: proof

\[ \hat{w}(s, v_1, \ldots, v_k, t) = w(s, v_1, \ldots, v_k, t) + h(s) - h(t) \]

Claim: the weight change preserves shortest paths, i.e. if a path was the shortest from \( s \) to \( t \) in the original graph it will still be the shortest path from \( s \) to \( t \) in the new graph.

\( h(s) - h(t) \) is a constant and will be the same for all paths from \( s \) to \( t \), so the absolute ordering of all paths from \( s \) to \( t \) will not change.

Selecting \( h \)

Need to pick \( h \) such that the resulting graph has all weights as positive

\[ \hat{w}(u, v) = w(u, v) + h(u) - h(v) \]

Johnson’s algorithm

Create \( G' \) with one extra node \( s \) with 0 weight edges to all nodes
run Bellman-Ford\( (G', s) \)
if no negative-weight cycle
reweight edges in \( G \) with \( h(v) \) = shortest path from \( s \) to \( v \)
run Dijkstra's from every vertex
reweight shortest paths based on \( G \)
Create $G'$
run Bellman-Ford($G',s$)
if no negative-weight cycle
    reweight edges in $G$ with $h(v)=$shortest path from $s$ to $v$
    run Dijkstra's from every vertex
    reweight shortest paths based on $G$

$A$ $B$ $C$ $D$ $E$
-1 2 1 -3
3 2 1 5
4 5 4 3
5 1 3 2

Create $G'$
run Bellman-Ford($G',s$)
if no negative-weight cycle
    reweight edges in $G$ with $h(v)=$shortest path from $s$ to $v$
    run Dijkstra's from every vertex
    reweight shortest paths based on $G$

$S$ $A$ $B$ $C$ $D$ $E$
0 0 0 0 2
0 0 0 0 0
0 -1 0 0 0
0 0 -1 0 0
0 0 0 0 -1

Create $G'$
run Bellman-Ford($G',s$)
if no negative-weight cycle
    reweight edges in $G$ with $h(v)=$shortest path from $s$ to $v$
    run Dijkstra's from every vertex
    reweight shortest paths based on $G$

$S$ $A$ $B$ $C$ $D$ $E$
0 0 0 0 2
0 0 0 0 0
0 -1 0 0 0
0 0 -1 0 0
0 0 0 0 -1

Create $G'$
run Bellman-Ford($G',s$)
if no negative-weight cycle
    reweight edges in $G$ with $h(v)=$shortest path from $s$ to $v$
    run Dijkstra's from every vertex
    reweight shortest paths based on $G$

$S$ $A$ $B$ $C$ $D$ $E$
0 0 0 0 2
0 0 0 0 0
0 -1 0 0 0
0 0 -1 0 0
0 0 0 0 -1
Create $G'$
run Bellman-Ford($G',s$)
if no negative-weight cycle
    reweight edges in $G$ with $h(v)=$shortest path from $s$ to $v$
    run Dijkstra's from every vertex
    reweight shortest paths based on $G$

181

Create $G'$
run Bellman-Ford($G',s$)
if no negative-weight cycle
    reweight edges in $G$ with $h(v)=$shortest path from $s$ to $v$
    run Dijkstra's from every vertex
    reweight shortest paths based on $G$

182

$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$

183

184
Create $G'$
run Bellman-Ford($G',s$)
if no negative-weight cycle
  reweight edges in $G$ with $h(v)$=shortest path from $s$ to $v$
run Dijkstra's from every vertex
  reweight shortest paths based on $G$

$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$

Create $G'$
run Bellman-Ford($G',s$)
if no negative-weight cycle
  reweight edges in $G$ with $h(v)$=shortest path from $s$ to $v$
run Dijkstra's from every vertex
  reweight shortest paths based on $G$

$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$
Create $G'$
run Bellman-Ford($G'$, $s$)
if no negative-weight cycle
reweight edges in $G$ with $h(v)$ = shortest path from $s$ to $v$
run Dijkstra's from every vertex
reweight shortest paths based on $G$

$\hat{w}(u, v) = w(u, v) + h(u) - h(v)$

$\hat{w}(u, v) = w(u, v) + h(u) - h(v)$
Create $G'$
run Bellman-Ford($G', s$)
if no negative-weight cycle
    reweight edges in $G$ with $h(v)$ = shortest path from $s$ to $v$
run Dijkstra's from every vertex
    reweight shortest paths based on $G$

Create $G'$
run Bellman-Ford($G', s$)
if no negative-weight cycle
    reweight edges in $G$ with $h(v)$ = shortest path from $s$ to $v$
run Dijkstra's from every vertex
    reweight shortest paths based on $G$
Selecting h

Need to pick $h$ such that the resulting graph has all weights as positive

Create $G'$ with one extra node $s$ with 0 weight edges to all nodes
run Bellman-Ford($G', s$)
if no negative-weight cycle
  reweight edges in $G$ with $h(v)$ = shortest path from $s$ to $v$
  run Dijkstra's from every vertex
reweight shortest paths based on $G$

Why does this work (i.e. how do we guarantee that reweighted graph has only positive edges)?

Reweighted graph is positive

Take two nodes $u$ and $v$

$h(u)$ shortest distance from $s$ to $u$
$h(v)$ shortest distance from $s$ to $v$

Claim: $h(v) \leq h(u) + w(u, v)$

Why?

Reweighted graph is positive

Take two nodes $u$ and $v$

$h(u)$ shortest distance from $s$ to $u$
$h(v)$ shortest distance from $s$ to $v$

Claim: $h(v) \leq h(u) + w(u, v)$

If this weren't true, we could have made a shorter path $s$ to $v$
using $u$

... but this is in contradiction with how we defined $h(v)$
Reweighted graph is positive

Take two nodes $u$ and $v$

- $h(u)$ shortest distance from $s$ to $u$
- $h(v)$ shortest distance from $s$ to $v$

\[
\hat{w}(u, v) = w(u, v) + h(u) - h(v) \geq 0
\]

All edge weights in reweighted graph are non-negative

Johnson's algorithm

Create $G'$

run Bellman-Ford($G', s$)

if no negative-weight cycle

reweight edges in $G$

run Dijkstra's from every vertex

reweight shortest paths based on $G$

Run-time?

All pairs shortest paths

- \[ V \times \text{Bellman-Ford}: O(V^2E) \]
- \[ \text{Floyd-Warshall}: \theta(V^3) \]
- Johnson's: \[ O(V^2 \log V + V E) \]