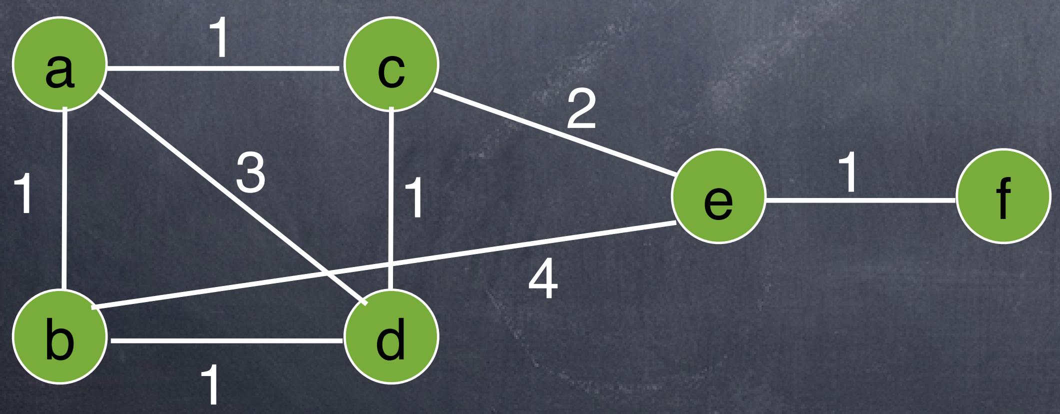
Dijkstra's Algorithm

Shortest Path Problem

- Directed graph G = (V, E)
- Source s
- l_e = length of edge e
- $l_e \geq 0$ for all edges e

Shortest path problem: (Google Maps!) find shortest path from s to all other nodes



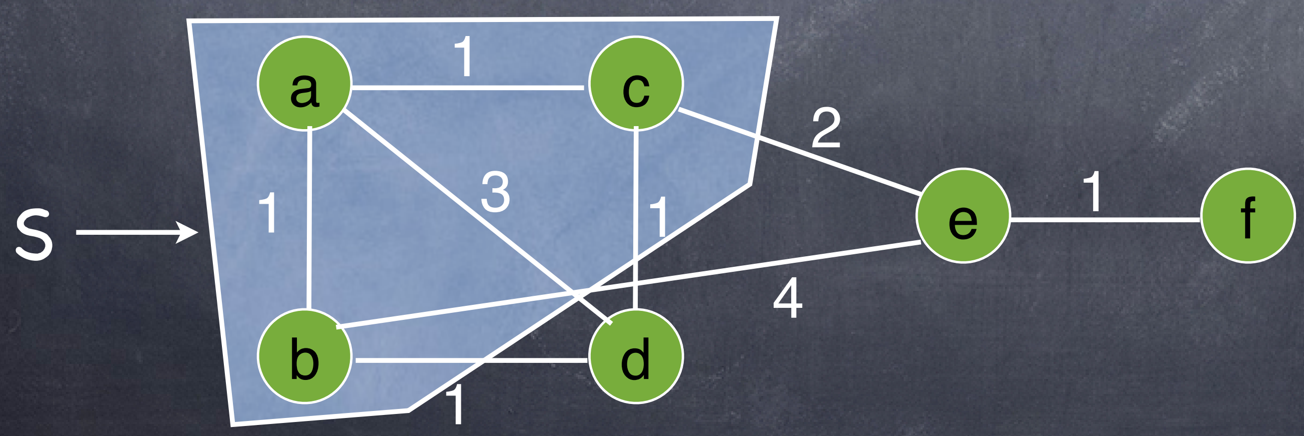
Simplification

- For now, let's just find the lengths of the shortest paths to every other node
- We'll show how to recover the paths later

Dijkstra's Algorithm

Idea: explore outward by distance

- Maintain set S of explored nodes: for $u \in S$, we know the length $d(u)$ of the shortest path from s to u.
- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly find shortest path to any node $v \notin S$ that remains in S until the final edge $e = (u, v)$
- $S = S \cup \{v\}$, $d(v) = d(u) + l_e$

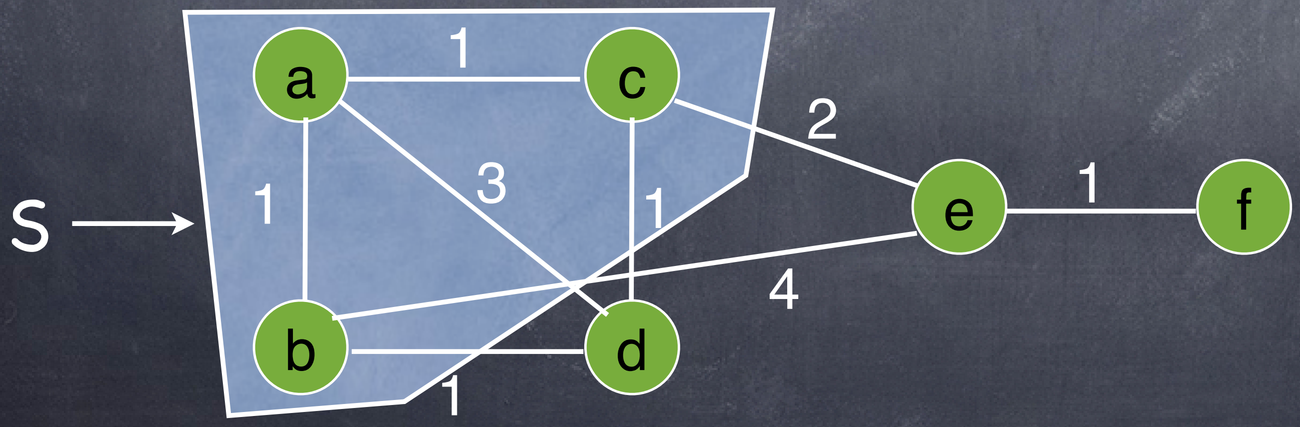


Dijkstra's Algorithm

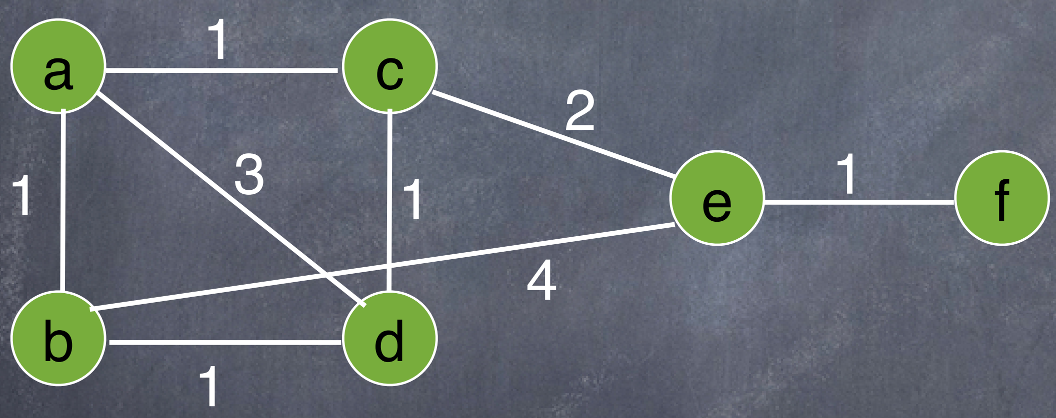
Repeatedly find shortest path to any node $v \notin S$ that remains in S until the final edge $e = (u, v)$

Minimize:

$$d'(v) = \min \{ d(u) + l_e : u \in S, e = (u, v) \in E \}$$

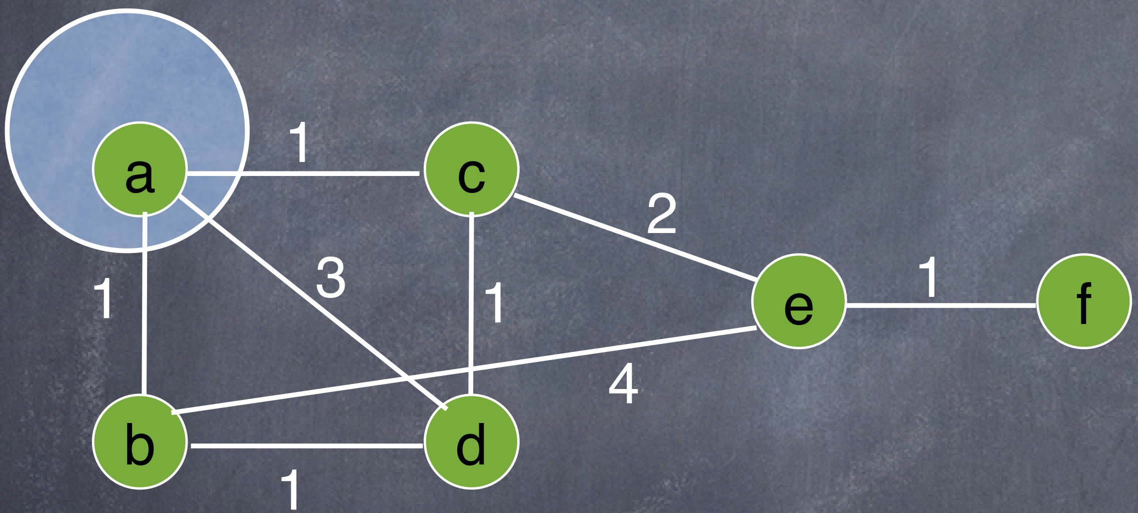


Dijkstra's Algorithm



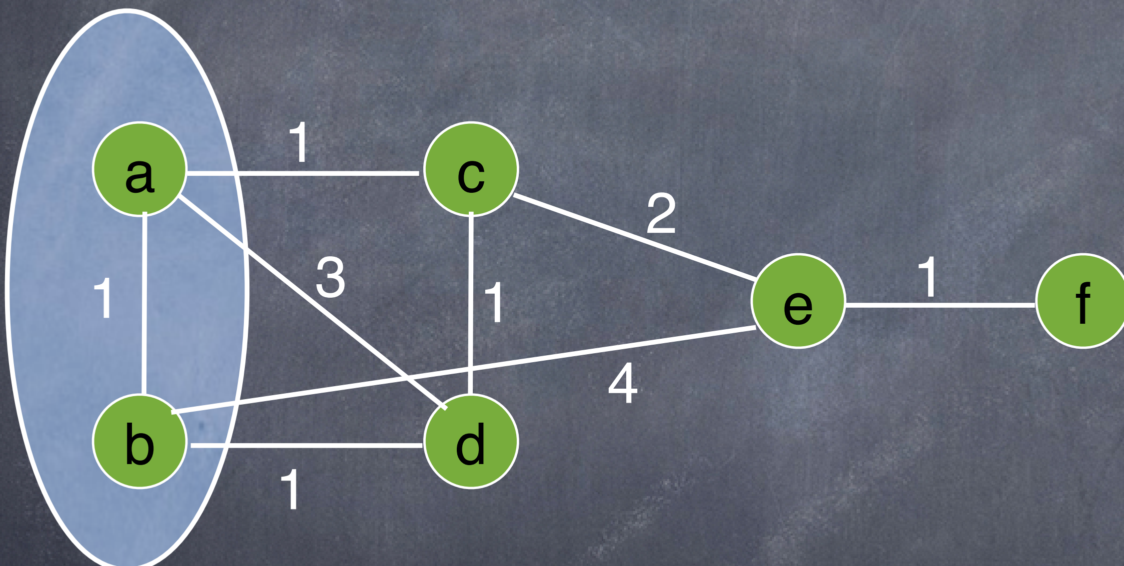
Node	d()

Dijkstra's Algorithm



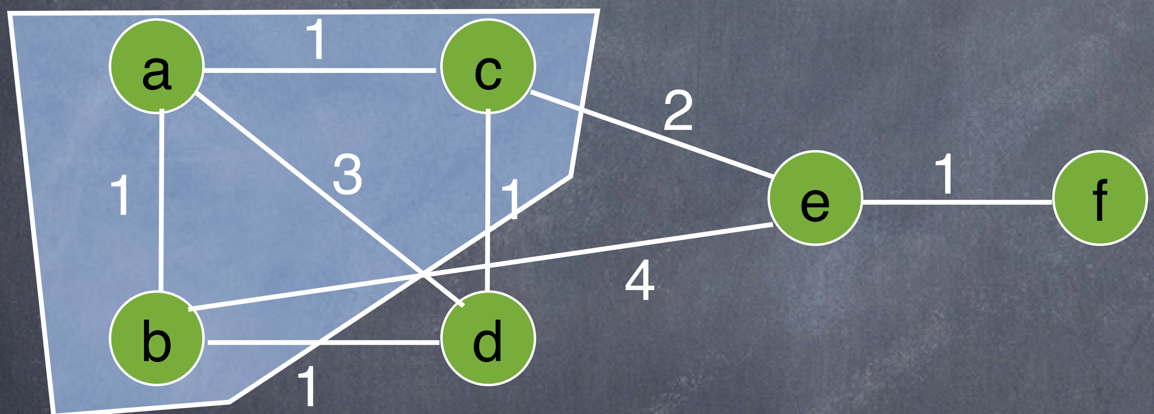
Node	d()
a	0

Dijkstra's Algorithm



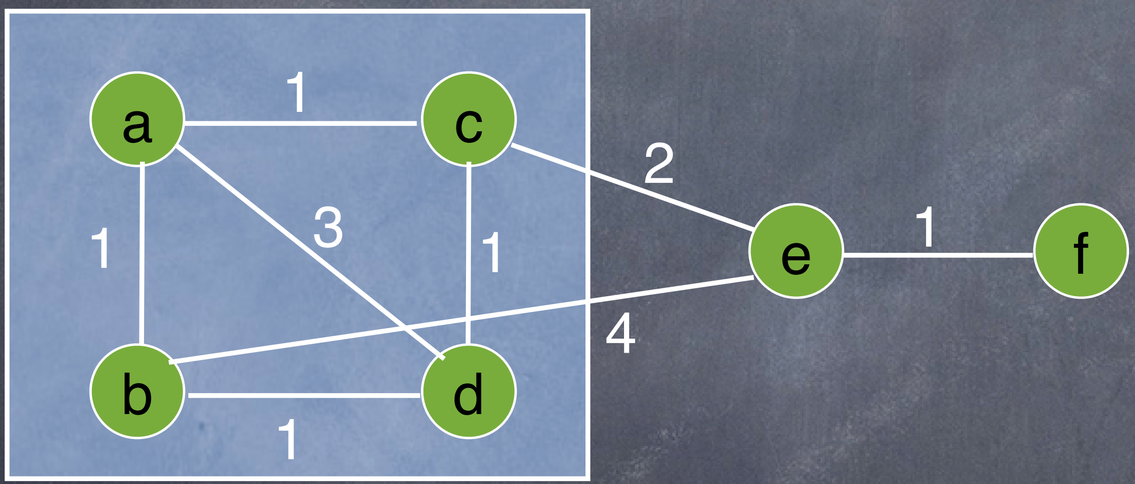
Node	d()
a	0
b	1

Dijkstra's Algorithm



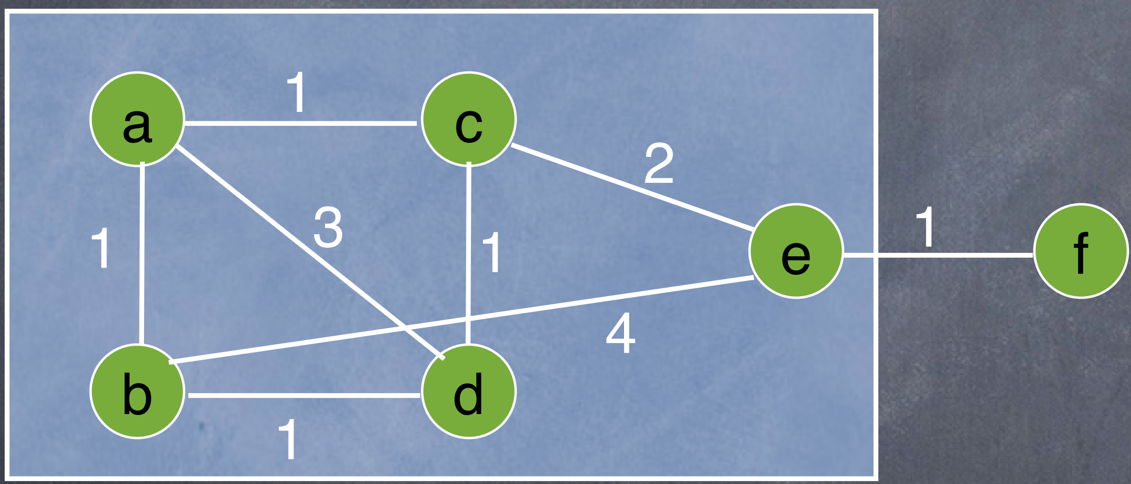
Node	d()
a	0
b	1
c	1

Dijkstra's Algorithm



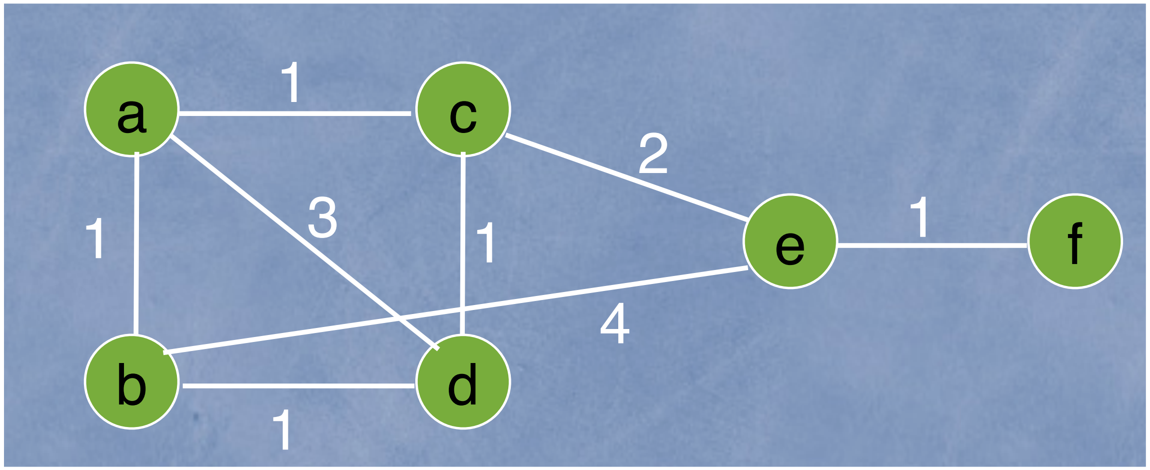
Node	d()
a	0
b	1
c	1
d	2

Dijkstra's Algorithm



Node	d()
a	0
b	1
c	1
d	2
e	3

Dijkstra's Algorithm



Node	d()
a	0
b	1
c	1
d	2
e	3
f	4

Example

- Second example on board

Dijkstra's Algorithm: Implementation

```
Dijkstra's Algorithm (G, s) {
  S = {s}      // S is the set of explored nodes
  d(s) = 0      // d is the distance to the node from s

  while S ≠ V {    // there are unexplored nodes
    select a node v ∉ S with at least one edge from S to
      minimize d'(v) = min { d(u) + l_e : u ∈ S, e = (u, v) ∈ E }
    add v to S
    d(v) = d'(v)
  }
```

How do we recover a path with length d(v)?

Dijkstra's Algorithm: Implementation

```
Dijkstra's Algorithm (G, s) {
  S = {s}      // S is the set of explored nodes
  d(s) = 0     // d is the distance to the node from s

  while S ≠ V {    // there are unexplored nodes
    select a node v ∉ S with at least one edge from S to
      minimize d'(v) = min { d(u) + l_e : u ∈ S, e = (u, v) ∈ E }
    add v to S
    d(v) = d'(v)
    prev(v) = argmin { d(u) + l_e : u ∈ S, e = (u, v) ∈ E }
}
```

Proof of correctness: start in small groups, complete on board

Dijkstra's Algorithm: Implementation

```
Dijkstra's Algorithm (G, s) {
  S = {s}      // S is the set of explored nodes
  d(s) = 0     // d is the distance to the node from s

  while S ≠ V {    // there are unexplored nodes
    select a node v ∉ S with at least one edge from S to
      minimize d'(v) = min { d(u) + l_e : u ∈ S, e = (u, v) ∈ E }
    add v to S
    d(v) = d'(v)
    prev(v) = argmin { d(u) + l_e : u ∈ S, e = (u, v) ∈ E }
  }
```

How do we implement this efficiently?

```
Dijkstra's Algorithm (G, s) {
  S = {s}        // S is the set of explored nodes
  d'(s) = 0   // explicitly maintain the d' values
  d'(v) = ∞ for all v ∉ S

  while S != V {    // there are unexplored nodes
    Let v be the node that minimizes d'(v)
    d(v) = d'(v)
    for each edge (v, w) where v ∈ S, w ∉ S {
        d'(v) = min( d'(v), d(v) + l(v, w) )
     }
  }
}
```

Data structure?

Running Time

- With heap-based priority queue, running time of Dijkstra's algorithm is:
  - O(m) - traverse edges
  - O(n log n) - n extractMin operations
  - O(m log n) - m changeKey operations
- Total: O(m log n)