CS 170 Section 3
Shortest Paths

Owen Jow | owenjow@berkeley.edu

Agenda

- Breadth-first search
- Dijkstra’s algorithm
- Bellman-Ford algorithm

Breadth-First Search

Breadth-First Search

- Traverses a tree in order of increasing distance from a source node
- **The distance from A to B** is defined as the number of edges in the path from A to B
- Like DFS, except with a queue instead of a stack

Figure 4.3 Breadth-first search.

```
procedure bfs(G, s)
Input:    Graph G = (V, E), directed or undirected; vertex s ∈ V
Output:   For all vertices u reachable from s, dist(u) is set
          to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞

dist(s) = 0
Q = [s]  (queue containing just s)
while Q is not empty:
   u = eject(Q)
   for all edges (u, v) ∈ E:
      if dist(v) = ∞:
         inject(Q, v)
         dist(v) = dist(u) + 1
```

Dijkstra’s Algorithm

Dijkstra's Algorithm

- An algorithm for finding shortest paths in the presence of edge weights
- Processes nodes in order of increasing distance from the source
- If the graph contains negative edges, Dijkstra's may or may not work

Figure 4.8 Dijkstra's shortest-path algorithm.

```
procedure dijkstra(G, l, s)
Input:    Graph G = (V, E), directed or undirected;
          positive edge lengths {l_e : e ∈ E}; vertex s ∈ V
Output:   For all vertices u reachable from s, dist(u) is set
          to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞
   prev(u) = nil
dist(s) = 0

H = makequeue(V)   (using dist-values as keys)
while H is not empty:
   u = deletemin(H)
   for all edges (u, v) ∈ E:
      if dist(v) > dist(u) + l(u, v):
         dist(v) = dist(u) + l(u, v)
         prev(v) = u
         decreasekey(H, v)
```

Bellman-Ford Algorithm

Bellman-Ford Algorithm

- An algorithm for finding shortest paths from a source, regardless of negative edge weights
- Perform **every** edge update; repeat |V| - 1 times
- Guaranteed to update each shortest path's edges in the correct order

Figure 4.13 The Bellman-Ford algorithm for single-source shortest paths in general graphs.

```
procedure shortest-paths(G, l, s)
Input:   Directed graph G = (V, E);
         edge lengths {l_e : e ∈ E} with no negative cycles;
         vertex s ∈ V
Output:  For all vertices u reachable from s, dist(u) is set
         to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞
   prev(u) = nil

dist(s) = 0
repeat |V| − 1 times:
   for all e ∈ E:
      update(e)
```

```
procedure update((u, v) ∈ E)
dist(v) = min{dist(v), dist(u) + l(u, v)}
```