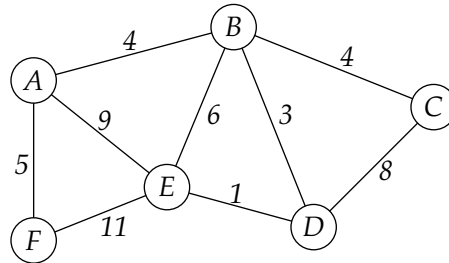


Paths & Searches

22.1 Distance

The *distance* between two vertices is the minimum number of arcs/edges on path between them. In a weighted graph, the *weight* of a path is the sum of weights of arcs/edges. The distance between two vertices is the minimum weight of a path between them.



In the example, distance A–E is 8 (via B and D)

22.2 Breadth-first Search

The idea is to *Visit source; then all its neighbors; then all their neighbors; and so on.* Thus, vertices are visited in order of their *distance* from the start. If the graph is a tree, this is *level ordering*.

```

Algorithm: BFS (start):
  enqueue start
  while queue not empty {
    v = dequeue
    for all out-neighbors w of v
      if ( w not visited ) {
        visit w
        enqueue w
      }
  }

```

22.3 Dijkstra's Algorithm

Dijkstra's algorithm determines the distance from the start vertex to all other vertices. The idea is to *Determine distances in increasing distance from the start*. For each vertex, maintain *dist* giving minimum weight of path to it found so far. Each iteration, choose vertex of minimum *dist*, finalize it and update *dist*.

```

Algorithm: dijkstraOutline (start):
  initialise dist for each vertex
  while some vertex un-finalized {
    v = un-finalized with minimum dist
    finalize v
    for all out-neighbors w of v
      dist(w) = min(dist(w), dist(v) + cost v-w)
  }
  
```

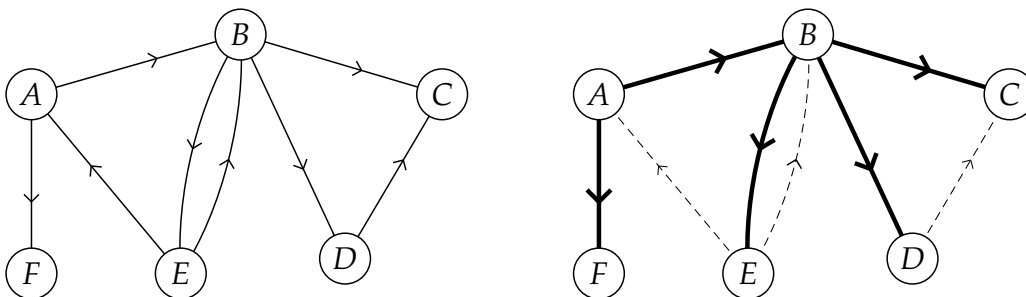
Comments:

- why works? Exercise.
- implementation, store boolean known
- to get path, store Vertex prev
- running time: naïve is $O(n^2)$
- to speed up: use priority queue that supports decreaseKey

22.4 Depth-First Search

The idea is *labyrinth wandering*: keep exploring new vertex from current vertex; when get stuck, backtrack to most recent vertex with unexplored neighbors. The edges/arcs used to discover new vertices are called *tree edges*.

EXAMPLE



Digraph and DFS tree from A

```
Algorithm: DFS(v):
  for all edges e outgoing from v
    w = other end of e
    if w unvisited then {
      label e as tree-edge
      recursively call DFS(w)
    }
```

Notes on DFS

- the search visits all vertices that are reachable
- fastest if uses adjacency list
- to keep track of whether visited a vertex, must add field to vertex (the decorator pattern)

22.5 Test for Strong Connectivity

Algorithm: 1. Do a DFS from arbitrary vertex v & check if all vertices
2. Reverse all arcs and repeat

Why does this work? Think of vertex v as the hub.