Today: Dijkstra's Algorithm
**Generic Search At (G, v)**

```
Make(B) // B is the Box (Data Structure)

Start(v, nil)
    color[v] ← grey
    Insert(v, B)

Update(v, nil, B)
    repeat
        u ← Delete(B)
        Finish(u)
            color[u] ← black
        for each w adjacent to u do
            if color[w] ← white then
                Start(w, u)
                color[w] ← grey
                Insert(w, B)
            if color[w] ← grey then
                Update(w, u, B)
        until Empty(B)
```

**Depth First Search**: Use a stack (LIFO)

- The vertices tend to be visited first when they are further away (visiting descendants before parents).

**Breadth First Search**: Use a queue (FIFO)

- Visit the parents before the descendants (the closer vertices are visited first).

(I'm thinking we'll use a BFS since if there are multiple paths but the vertices reach the destination, we'll have followed the shortest path.)
BFS returns PATH A (5)
while PATH B (3), which is shorter!!

The "queue" is a priority queue (which can be implemented as a binary min-heap)

1. We need to add attributes to the vertices:
   - $d(v)$: The weighted distance of the shortest path from $S$
   - $b(v)$: The "parent node" of $v$ along the shortest path.

When we start the algorithm, we initialize

- $d(v) = \infty$ for all $v \in V$ except $v = S$
- $b(v) = \text{nil}$ for all $v \in V$

Actually set $d(S) = 0$

We need to define the Update routine so that it always picks the shortest path.
Update \((v, u, B)\)

// if (Path A) > (Path B)

if \(d(v) > d(u) + c[u, v]\)

\[b(v) \leftarrow u\]

Decrease Key \((B, v, v)\)

To create Dijkstra's Algorithm
from the generic search:

1. Add attributes to the vertices
2. Implement a min-priority queue
3. Change the Update method

Worst Case Run-time: \(\Theta((V^2+E)\log(V))\)

Semmy@engr.sc.edu