Breadth First Search/Dijkstra.
Lecture in a minute.

Breadth First Search of graph:
Search with queue instead of stack.
Get “distances” from source.
Proof idea: queue has distance 0, then level 1, …

Djikstra: shortest paths in weighted graph:
Replace weights by paths + BFS
Implement using priority queue.
Idea: ignores “new” nodes.
Runtime: |V| extracts, |E| reduce-key for p-queue.

Priority Queue:
Implementation: degree $d$ tree.
Heap Property: children larger than parent.
Minimum at top.
Remove min: $O(d \log_d n)$ time: Replace min/percolate down.
Reduce Key: $O(\log_d n)$: percolate up.
Binary Heap.

Heap\textsuperscript{1}: bigger children.  
\[ \Rightarrow \]  smallest at root.

---

Insert(7): Bubble up: check parent.  
\[ \text{depth} \times \text{comp.} \]

DeleteMin: Replace.  
Bubble down: check both children.

\[ 2 \times \text{depth} \] – comparisons.

\textsuperscript{1}values only
**d-ary heap**

Degree – $d$, Depth – $\log_d n$.
Insert/DecreaseKey – $\log n / \log d$.
DeleteMin – $d \log n / \log d$. (Check all children.)

Dijkstra:
$O(|V|)$ deletemins. $O(d \log n / \log d)$ each.
$O(|E|)$ insert/decrease-keys. $O(\log n / \log d)$ each.

$O(|V|d \log n / \log d + |E| \log n / \log d)$.

Optimal Choice: Choose $d = |E| / |V|$ (average degree/2)
$O(|E| \log n / \log d)$

For dense graphs it approaches linear.

Fibonacci Heaps:
$O(\log n)$ per delete.
$O(1)$ average decrease-key.

$O(|V| \log |V| + |E|)$.
Linear for moderately dense graphs!
Breadth First Search of graph:
Search with queue instead of stack.
Get “distances” from source.
Proof idea: queue has distance 0, then level 1, …

Dijkstra: shortest paths in weighted graph:
Replace weights by paths + BFS
Implement using priority queue.
Idea: ignores “new” nodes.
Runtime: $|V|$ extracts, $|E|$ reduce-key for p-queue.

Priority Queue:
Implementation: degree $d$ tree.
Heap Property: children larger than parent.
Minimum at top.
Remove min: $O(d \log_d n)$ time: Replace min/percolate down.
Reduce Key: $O(\log_d n)$: percolate up.