Breadth First Search of graph:
Search with queue instead of stack.
Get “distances” from source.

Dijkstra: shortest paths in weighted graph:
Replace weights by paths + BFS
Implement using priority queue.
Idea: ignores “new” nodes.

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.

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

Dijkstra:
$O(1)$ deletions, $O(d \log n / \log d)$ each.
$O(1)$ insert/decrease-keys. $O(\log n / \log d)$ each.
$O(1)$ distance $d = |E| / |V|$ (average degree $d$).
$O(1)$ average decrease key.

Optimal Choice: Choose $d = |E| / |V|$ (average degree $d$).

For dense graphs it approaches linear.

Fibonacci Heaps:
$O(\log n)$ per delete.
$O(1)$ average decrease key.
$O(1)$ decrease $|V| + |E|$.
Linear for moderately dense graphs!