Depth-first Search

- Depth-first search treats the frontier as a stack
- It always selects one of the last elements added to the frontier.
- If the frontier is \([p_1, p_2, \ldots]\)
  - \(p_1\) is selected. Paths that extend \(p_1\) are added to the front of the stack (in front of \(p_2\).
  - \(p_2\) is only selected when all paths from \(p_1\) have been explored.
Illustrative Graph — Depth-first Search
Complexity of Depth-first Search

- Depth-first search isn’t guaranteed to halt on infinite graphs or on graphs with cycles.
- The space complexity is linear in the size of the path being explored.
- Search is unconstrained by the goal until it happens to stumble on the goal.
Breadth-first Search

- **Breadth-first search** treats the frontier as a queue.

- It always selects one of the earliest elements added to the frontier.

- If the frontier is \([p_1, p_2, \ldots, p_r]\):
  - \(p_1\) is selected. Its neighbors are added to the end of the queue, after \(p_r\).
  - \(p_2\) is selected next.
Illustrative Graph — Breadth-first Search
Complexity of Breadth-first Search

The **branching factor** of a node is the number of its neighbors.

If the branching factor for all nodes is finite, breadth-first search is guaranteed to find a solution if one exists. It is guaranteed to find the path with fewest arcs.

Time complexity is exponential in the path length: \( b^n \), where \( b \) is branching factor, \( n \) is path length.

The space complexity is exponential in path length: \( b^n \).

Search is unconstrained by the goal.
Lowest-cost-first Search

- Sometimes there are costs associated with arcs. The cost of a path is the sum of the costs of its arcs.

\[
\text{cost}(\langle n_0, \ldots, n_k \rangle) = \sum_{i=1}^{k} |\langle n_{i-1}, n_i \rangle|\]

- At each stage, lowest-cost-first search selects a path on the frontier with lowest cost.
- The frontier is a priority queue ordered by path cost.
- It finds a least-cost path to a goal node.
- When arc costs are equal \(\implies\) breadth-first search.