CS 61B Fall 2025

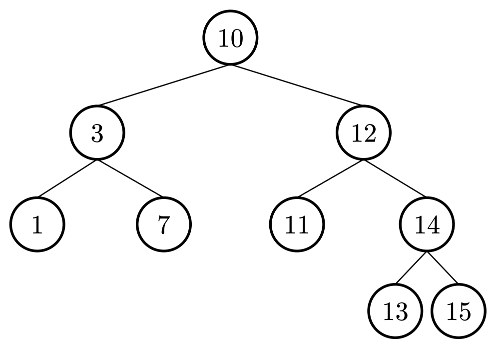
Tree Traversals, Graphs, and Shortest Paths

Discussion 08: October 20, 2025

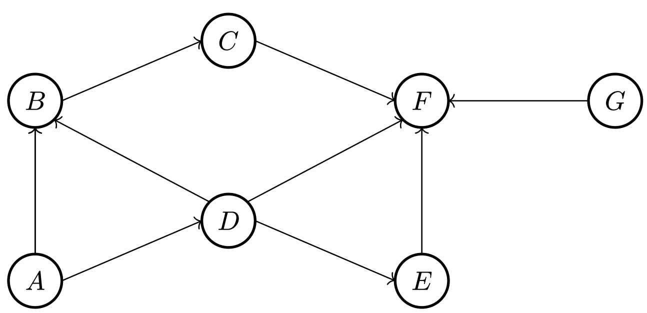
1 Trees, Graphs, and Traversals

(a) Write the following traversals of the BST below.

Pre-order:
In-order:
Post-order:
Level-order (BFS):



(b) Write the graph below as an adjacency matrix, then as an adjacency list. What would be different if the graph were undirected instead?



(c) Write the order in which (1) DFS pre-order, (2) DFS post-order, and (3) BFS would visit nodes in the same directed graph above, starting from vertex A. Break ties alphabetically.

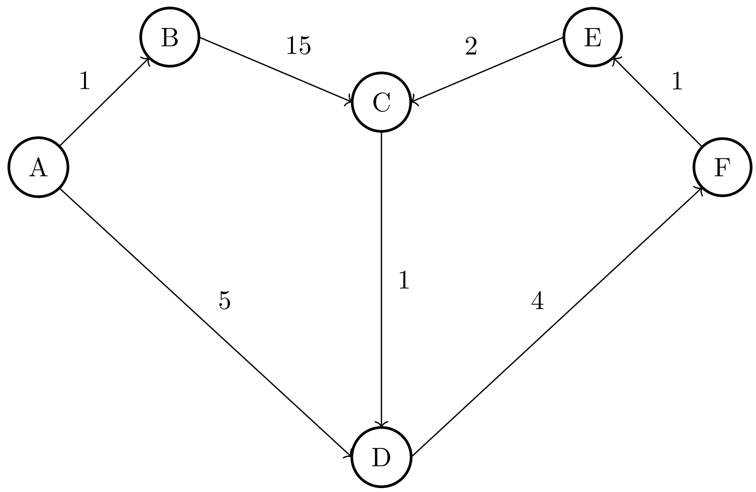
Pre-order:

Post-order:

BFS:

2 The Shortest Path to your Heart

For the graph below, let **g(u, v)** be the weight of the edge between any nodes **u** and **v**. Let **h(u, v)** be the value returned by the heuristic for any nodes u and v.



Below, the pseudocode for Dijkstra's and A∗ are both shown for your reference throughout the problem.

Dijkstra's Pseudocode

```
PQ = new PriorityQueue()
PQ.add(A, 0)
PQ.add(v, infinity) # (all nodes except A).

distTo = {} # map
edgeTo = {} # map
distTo[A] = 0
distTo[v] = infinity # (all nodes except A).

while (not PQ.isEmpty()):
  poppedNode, poppedPriority = PQ.pop()

  for child in poppedNode.children:
    potentialDist = distTo[poppedNode] +
      edgeWeight(poppedNode, child)

    if potentialDist < distTo[child]:
      distTo.put(child, potentialDist)
      PQ.changePriority(child,
potentialDist)
      edgeTo[child] = poppedNode
```

A∗ Pseudocode

```
PQ = new PriorityQueue()
PQ.add(A, h(A, goal))
PQ.add(v, infinity) # (all nodes except A).

distTo = {} # map
distTo[A] = 0
distTo[v] = infinity # (all nodes except A).

while (not PQ.isEmpty()):
  poppedNode, poppedPriority = PQ.pop()
  if (poppedNode == goal): terminate

  for child in poppedNode.children:
    potentialDist = distTo[poppedNode] +
      edgeWeight(poppedNode, child)

    if potentialDist < distTo[child]:
      distTo.put(child, potentialDist)
      PQ.changePriority(child, potentialDist
        + h(child, goal))
      edgeTo[child] = poppedNode
```

(a) Run Dijkstra's algorithm to find the shortest paths from A to every other vertex. You may find it helpful to keep track of the priority queue. We have provided a table to keep track of best distances, and the adjacent vertex that has an edge going to the target vertex in the current shortest paths tree so far.

	A	B	C	D	E	F
distTo						
edgeTo						

(b) Given the weights and heuristic values for the graph above, what path would A∗ search return, starting from A and with F as a goal?

Edge Weights	**Heuristics**
$g(A, B) = 1$	$h(A, F) = 8$
$g(A, D) = 5$	$h(B, F) = 16$
$g(B, C) = 15$	$h(C, F) = 4$
$g(C, D) = 1$	$h(D, F) = 4$
$g(D, F) = 4$	$h(E, F) = 5$
$g(F, E) = 1$	
$g(E, C) = 2$	

	A	B	C	D	E	F
distTo						
edgeTo						

(c) Based on the heuristics for part b, is the A∗ heuristic for this graph good? In other words, will it always give us the actual shortest path from A to F? If it is good, give an example of a change you would make to the heuristic so that it is no longer good. If it is not, correct it.

3 Extra: Graph Conceptuals

(a) Answer the following questions as either **True** or **False** and provide a brief explanation:

1. If a graph with n vertices has $n-1$ edges, it **must** be a tree.

2. Every edge is looked at exactly twice in each full run of DFS on a connected, undirected graph.

3. In BFS, let $d(v)$ be the minimum number of edges between a vertex v and the start vertex. For any two vertices u, v in the fringe (recall that the fringe in BFS is a queue), $|d(u) - d(v)|$ is **always less than** 2.

(b) Given an undirected graph, provide an algorithm that returns true if a cycle exists in the graph, and false otherwise. Also, provide a Θ bound for the worst case runtime of your algorithm.