Blind (or Uninformed) Search Methodologies

Lecture 7: Artificial Intelligence

Dr. Bonaventure Chidube Molokwu

bonaventure.molokwu[at]csus.edu

College of Engineering and Computer Science
California State University
Sacramento
California - United States of America

Non-Heuristic or Uninformed Search

Lecture/Week Outline & Learning Outcomes

1. **Lesson/Week Outline:**

   1.1 Non-Heuristic (or Uninformed or Blind) Search

2. **Learning Outcomes:**

   2.1 Introduction to Search strategies/algorithms and their respective evaluation/comparison criteria.

   2.2 Conceptual details of Non-Heuristic/Uninformed Search algorithms: Breadth-First Search, Uniform -Cost Search, Depth-First Search, Depth-Limited Search, Iterative-Deepening Search, etc.

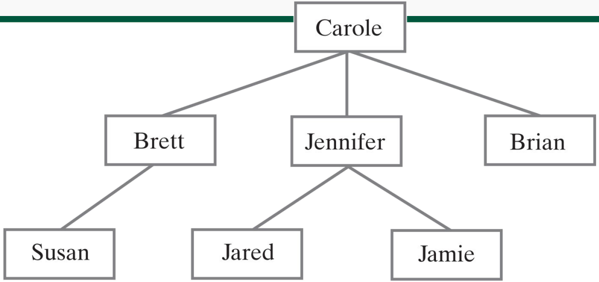
   2.3 Real-world usage scenarios such as pathfinding, decision-making process, planning, etc.

Non-Heuristic or Uninformed Search

Prelude

Non-Heuristic or Uninformed Search

Introduction



© 2019 Pearson Education, Inc.

Figure: **Tree:** arrange(Nonlinear) $\xrightarrow{\textit{hierarchy}}$ unordered(Items)

- **Tree:** edges(Relationship) $\xrightarrow{\textit{hierarchy}}$ set(Nodes)
- **Root:** node(Topmost Level) ⟶ node(Tree)

▶ **Major traversal(Tree) Techniques, viz:**

- Breadth-First Search (BFS).
- Depth-First Search (DFS).

Non-Heuristic or Uninformed Search

Breadth-First Search (BFS): Level-order Traversal of Binary Tree

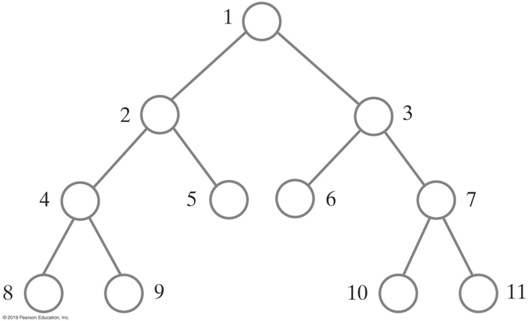


Figure: Visitation order of a `level-order` traversal

1. Visit the root.
2. Visit each node, from left-to-right, on level-by-level basis with respect to the subtrees.

Non-Heuristic or Uninformed Search

Depth-First Search (DFS): Preorder Traversal of a Binary Tree

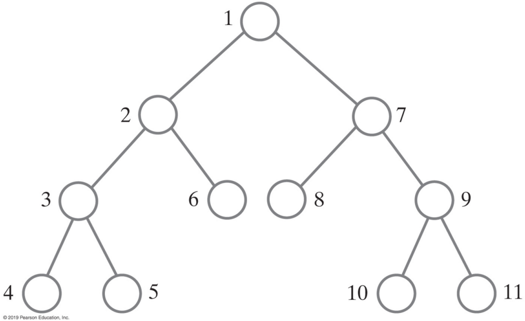


Figure: Visitation order of a `preorder` traversal

1. Visit the root.
2. Visit all the nodes in the root's left subtree.
3. Visit all the nodes in the root's right subtree.

Non-Heuristic or Uninformed Search

Depth-First Search (DFS): Inorder Traversal of a Binary Tree

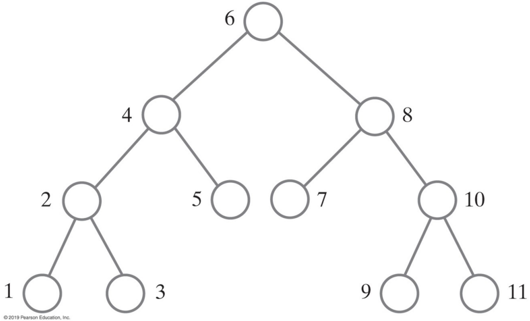


Figure: Visitation order of a `inorder` traversal

1. Visit all the nodes in the root's left subtree.
2. Visit the root.
3. Visit all the nodes in the root's right subtree.

Non-Heuristic or Uninformed Search

Depth-First Search (DFS): Postorder Traversal of a Binary Tree

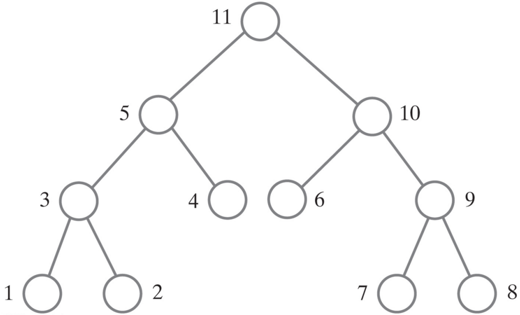


Figure: Visitation order of a `postorder` traversal

1. Visit all the nodes in the root's left subtree.
2. Visit all the nodes in the root's right subtree.
3. Visit the root.

Non-Heuristic or Uninformed Search

Depth-First Traversal and **Breadth-First Traversal** algorithms

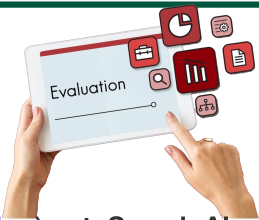
- **Breadth-First Traversal** algorithms:
  1. Level-order

- **Depth-First Traversal** algorithms:
  1. Preorder
  2. Inorder
  3. Postorder

- **NOTE**:
  1. `Inorder` traversal is usually NOT well-defined for a tree(General).

Non-Heuristic or Uninformed Search

Evaluation Criteria wrt. Search Algorithms



- **criteria(Evaluation) wrt. Search Algorithms:**
  - Completeness: checks if algorithm(Search) *always* find a solution, if one does exists?
  - Time Complexity: determines the number of nodes generated/expanded by algorithm(Search) wrt. time?
  - Space Complexity: the maximum number of nodes algorithm(Search) stores in memory wrt. its operation?
  - Optimality: checks if algorithm(Search) *always* find a `least-cost` soln.?
- **Variables herein wrt. complexity(Time & Space):**
  - *b*: Branching factor (average number of children per node).
  - *m*: Depth of the deepest node (maximum depth of the search tree).
  - *d*: Depth of the `least-cost` or `optimal` solution.

Non-Heuristic or Uninformed Search

Breadth-First Search (BFS): Traversal wrt. ADT(Graph)

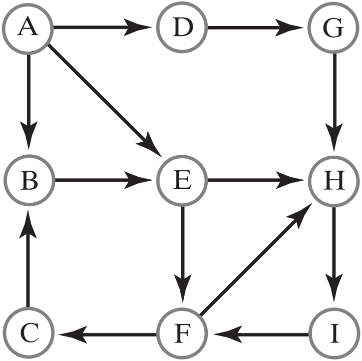


Figure: traversal(BreadthFirst) wrt. ADT(Graph)

- **traversal(BreadthFirst) wrt. Graph:**
  - Graph traversal can begin at any node(Vertex) called: **origin** node(Vertex)
  - node(Origin) $\underrightarrow{\textit{order}(\textit{Level\_by\_Level})}$ node(Adjacent/Neighbors)
- The order in which the Level-based adjacent/neighbor nodes are visited is NOT specific, and can it depend on the graph's implementation.

Non-Heuristic or Uninformed Search

Breadth-First Search (BFS): Traversal wrt. ADT(Graph)

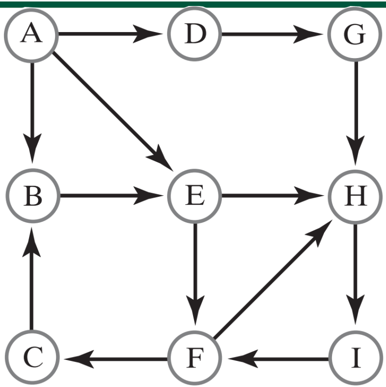


Figure: traversal(BreadthFirst) wrt. ADT(Graph)

- **Question 1:** Search/Traverse ADT(Graph) above by search(BreadthFirst) until node(I) is found.
  - **Solution 1.1:** origin(A) → `A, B, E, D, F, H, G, C, I`
  - **Solution 1.2:** origin(A) → `A, B, D, E, G, F, H, C, I`

Non-Heuristic or Uninformed Search

Breadth-First Search (BFS): Traversal wrt. ADT(Graph)

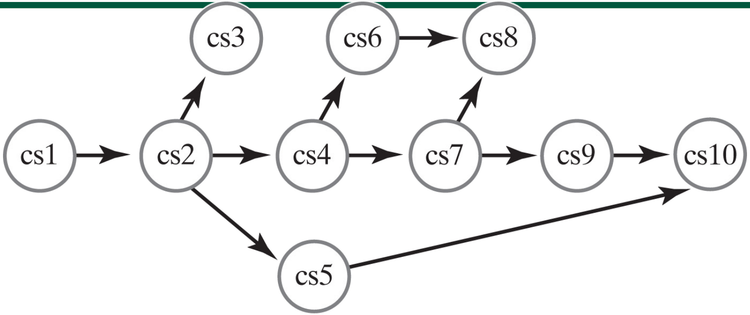


Figure: traversal(BreadthFirst) wrt. ADT(Graph)

- **Question 2:** Search/Traverse ADT(Graph) above by search(BreadthFirst) until node(cs8) is found.
  - **Solution 2.1:** origin(cs1) → `cs1, cs2, cs5, cs4, cs3, cs10, cs7, cs6, cs9, cs8`
  - **Solution 2.2:** origin(cs1) → `cs1, cs2, cs3, cs4, cs5, cs6, cs7, cs10, cs8`

Non-Heuristic or Uninformed Search

Breadth-First Search (BFS): Traversal wrt. ADT(Graph)

- **property(Optimality):**
  - YES: If all edges(Graph) has same cost. E.g. an unweighted(Graph) or weighted(Graph: edgeWeight=1).
  - YES: If node(Goal) wrt. graph is on the *shallowest* level.
  - NO: Otherwise, i.e. if above conditions not met, it is NOT optimal.

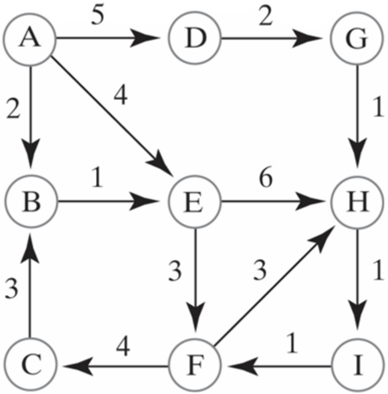
- **property(Run-time Complexity):**
  - $O(b^{d+1})$: where b = Branching factor (average number of children per node) AND d = Depth of the optimal solution.

- **property(Space Complexity):**
  - $O(b^{d+1})$: It stores all the nodes in memory, and b = Branching factor (average number of children per node) AND d = Depth of the optimal solution.

- **property(Other):**
  - Can lead to multiple visits (repeated-state checking) wrt. a node. Thus, can get stuck in loops.

Non-Heuristic or Uninformed Search

Uniform-Cost Search (UCS): Dijkstra's Algo. wrt. graph(Weight)



- **Question 3:** Find the Shortest Path from node(A) to node(I) wrt. *weighted* graph above.
  - **Solution:** Using Dijkstra's algorithm detailed via `Tree Traversal`.

Non-Heuristic or Uninformed Search

Uniform-Cost Search (UCS): Dijkstra's Algo. wrt. graph(Weight)

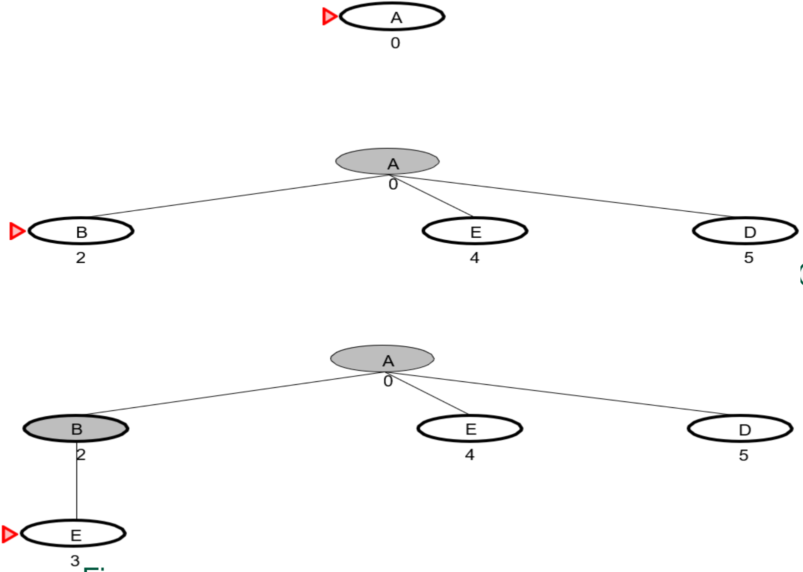


Figure: Trees(1..3) wrt. **Shortest Path** from node(A) to node(I)

Non-Heuristic or Uninformed Search

Uniform-Cost Search (UCS): Dijkstra’s Algo. wrt. graph(Weight)

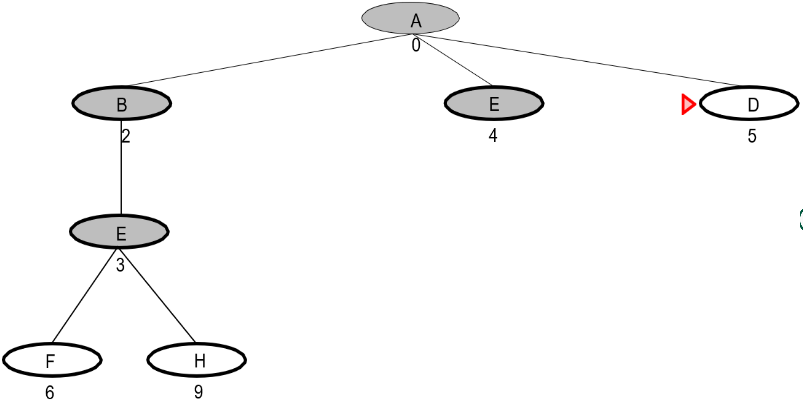


Figure: Tree(4) wrt. **Shortest Path** from node(A) to node(I)

Non-Heuristic or Uninformed Search

Uniform-Cost Search (UCS): Dijkstra's Algo. wrt. graph(Weight)

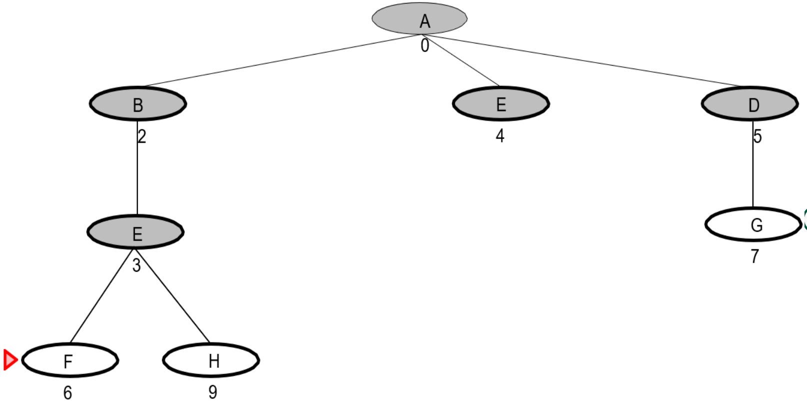


Figure: Tree(5) wrt. **Shortest Path** from node(A) to node(I)

Non-Heuristic or Uninformed Search

Uniform-Cost Search (UCS): Dijkstra's Algo. wrt. graph(Weight)

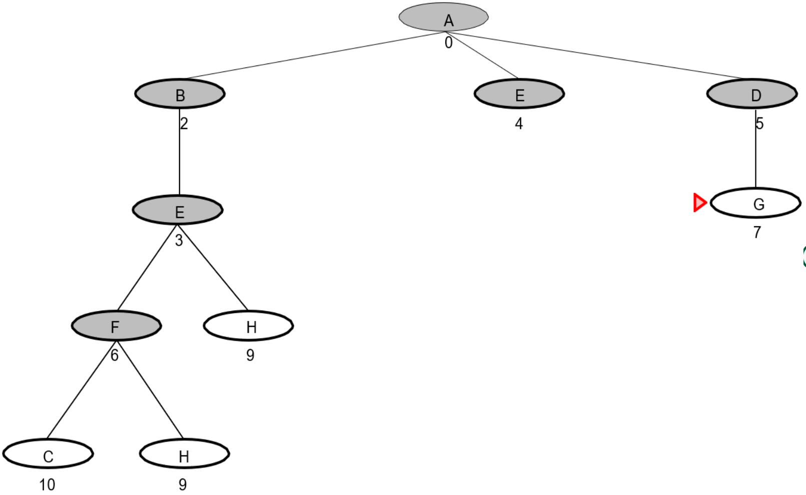


Figure: Tree(6) wrt. **Shortest Path** from node(A) to node(I)

Non-Heuristic or Uninformed Search

Uniform-Cost Search (UCS): Dijkstra's Algo. wrt. graph(Weight)

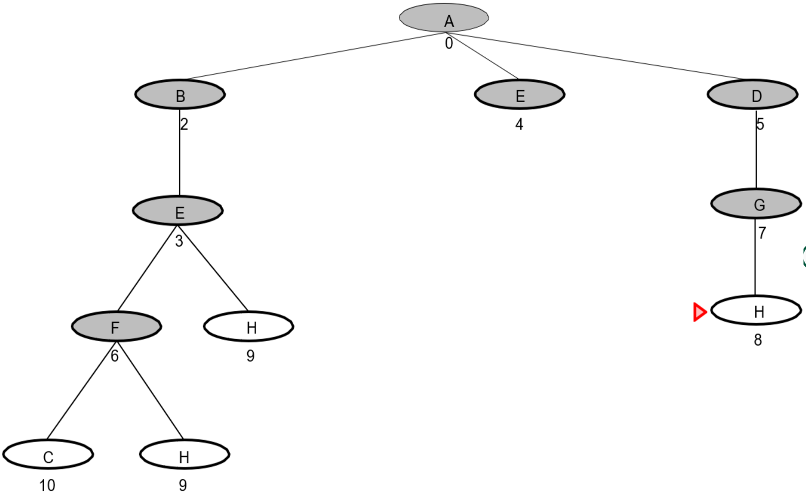


Figure: Tree(7) wrt. **Shortest Path** from node(A) to node(I)

Non-Heuristic or Uninformed Search

Uniform-Cost Search (UCS): Dijkstra's Algo. wrt. graph(Weight)

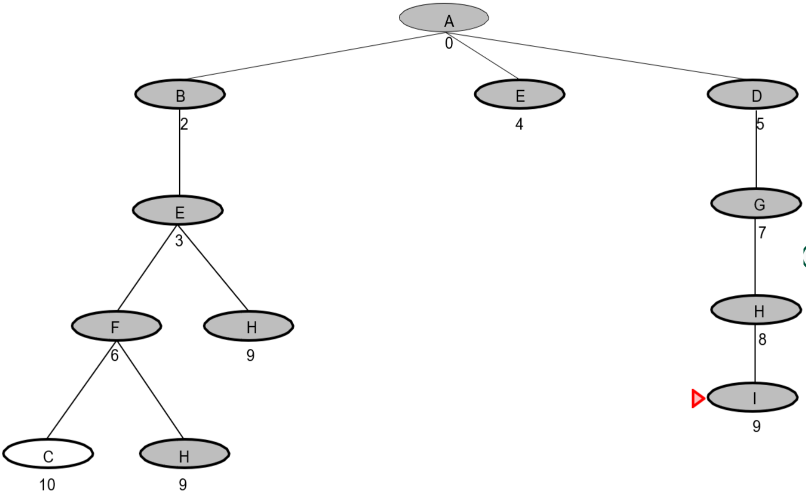
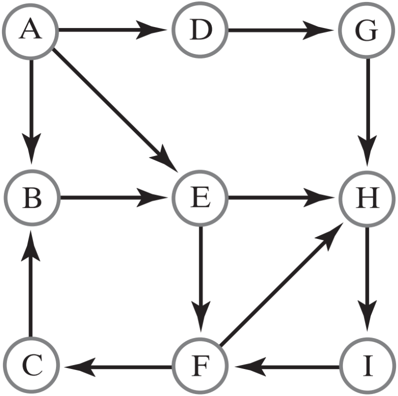


Figure: Tree(8) wrt. **Shortest Path** from node(A) to node(I)

Non-Heuristic or Uninformed Search

Uniform-Cost Search (UCS): Dijkstra’s Algo wrt. graph(Unweight)



- **Question 4:** Find the Shortest Path from node(A) to node(I) wrt. *unweighted* graph above.

  - **Solution:** Using Dijkstra’s algorithm detailed via `Tree Traversal`.

Non-Heuristic or Uninformed Search

Uniform-Cost Search (UCS): Dijkstra's Algo wrt. graph(Unweight)

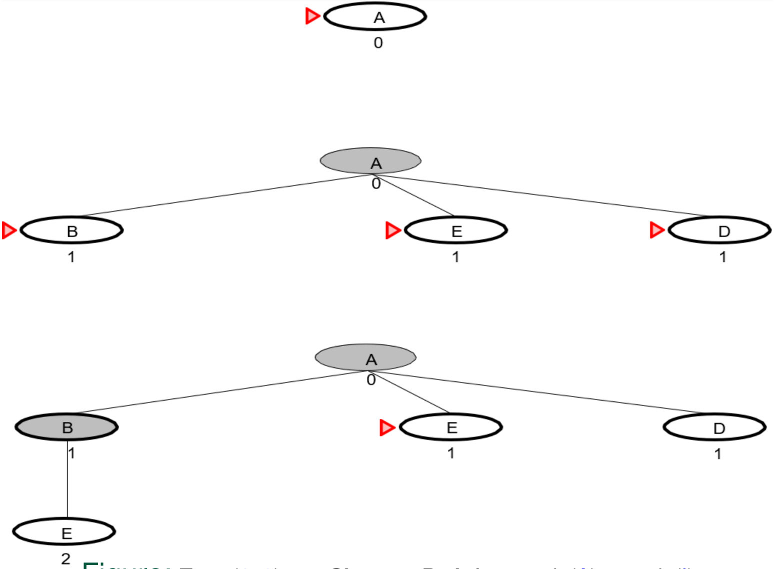


Figure: Trees(1..3) wrt. **Shortest Path** from node(A) to node(I)

Non-Heuristic or Uninformed Search

Uniform-Cost Search (UCS): Dijkstra's Algo wrt. graph(Unweight)

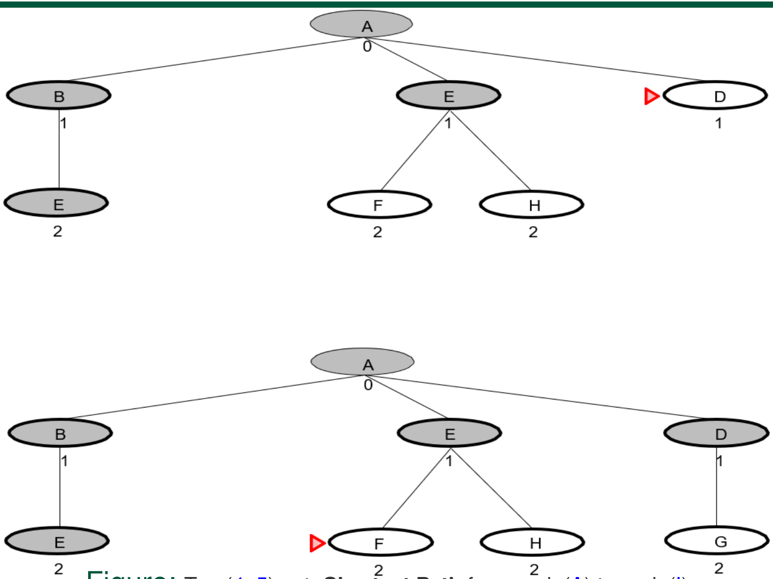


Figure: Tree(4..5) wrt. **Shortest Path** from node(A) to node(I)

Non-Heuristic or Uninformed Search

Uniform-Cost Search (UCS): Dijkstra's Algo wrt. graph(Unweight)

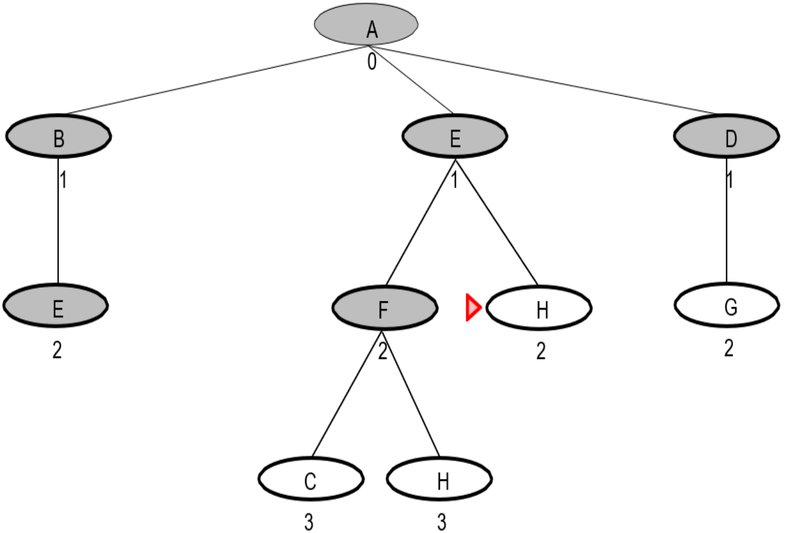


Figure: Tree(6) wrt. **Shortest Path** from node(A) to node(I)

Non-Heuristic or Uninformed Search

Uniform-Cost Search (UCS): Dijkstra's Algo wrt. graph(Unweight)

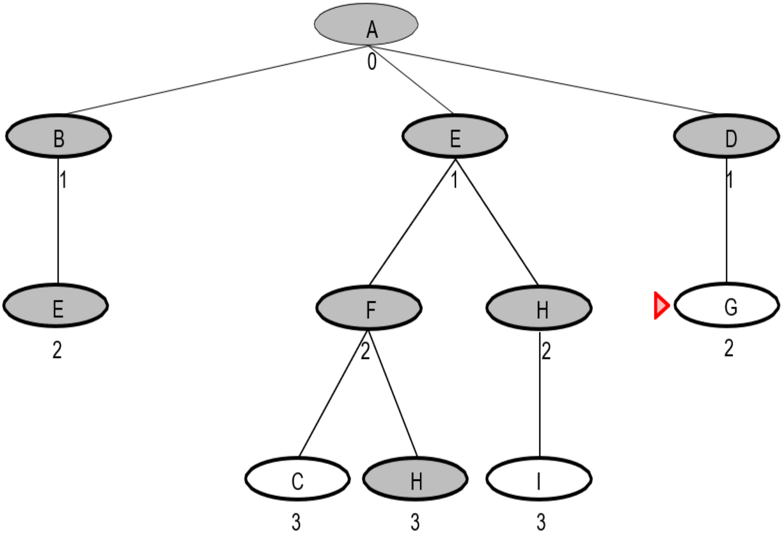


Figure: Tree(7) wrt. **Shortest Path** from node(A) to node(I)

Non-Heuristic or Uninformed Search

Uniform-Cost Search (UCS): Dijkstra's Algo wrt. graph(Unweight)

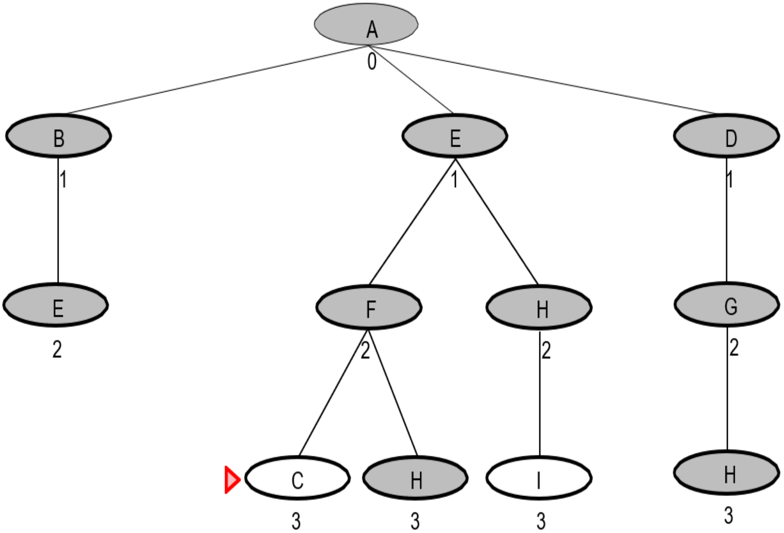


Figure: Tree(8) wrt. **Shortest Path** from node(A) to node(I)

Non-Heuristic or Uninformed Search

Uniform-Cost Search (UCS): Dijkstra's Algo wrt. graph(Unweight)

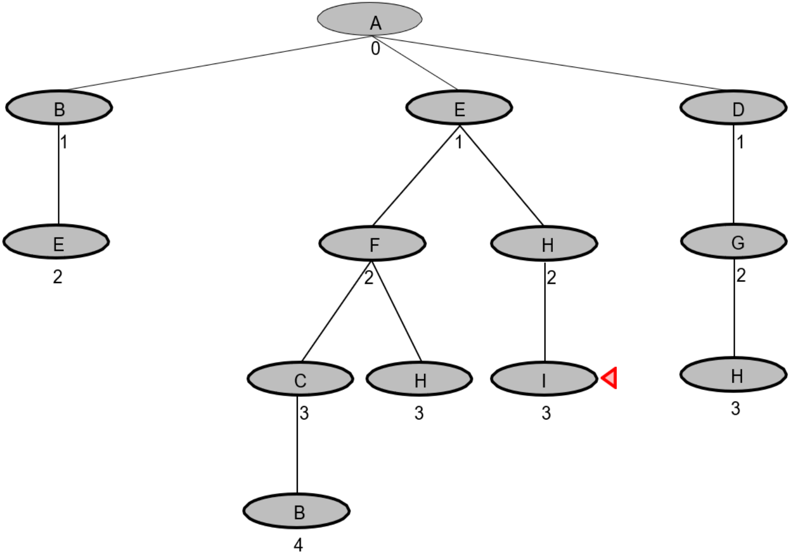


Figure: Tree(9) wrt. **Shortest Path** from node(A) to node(I)

Non-Heuristic or Uninformed Search

Uniform-Cost Search (UCS): Dijkstra's Algo. wrt. ADT(Graph)

- **property(Optimality):**
  - Yes, it is optimal because it always *threads* the least-cost path first wrt. finding a solution.
- **property(Run-time Complexity):**
  - $\underline{O((V+E)\log V)}$: If algorithm is implemented via dataStructure(Binary Heap); where V = Number of vertices/nodes AND E = Number of edges/ties.
  - $\underline{O(V^2)}$: If algorithm is implemented via dataStructure(Array); where V = Number of vertices/nodes AND E = Number of edges/ties.
- **property(Space Complexity):**
  - $\underline{O(V+E)}$: It stores all the vertices/nodes and edges/ties in memory. Thus, V = Number of vertices/nodes AND E = Number of edges/ties.
- **property(Other):**
  - It always finds a path, if one exists, provided that every node is reachable.

Non-Heuristic or Uninformed Search

Depth-First Search (DFS): Traversal wrt. ADT(Graph)

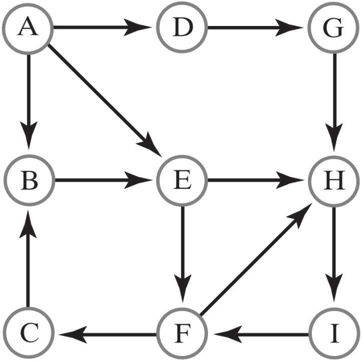


Figure: traversal(DepthFirst) wrt. ADT(Graph)

- **traversal(DepthFirst) wrt. Graph:**
  - Graph traversal can begin at any node(Vertex) called: **origin** node(Vertex)
  - node(Origin) *order*(*Indepth_by_Neighbor*) node(Adjacent/Neighbors)
- The order in which the Indepth adjacent/neighbor nodes are visited is NOT specific, and can it depend on the graph's implementation.

Non-Heuristic or Uninformed Search

Depth-First Search (DFS): Traversal wrt. ADT(Graph)

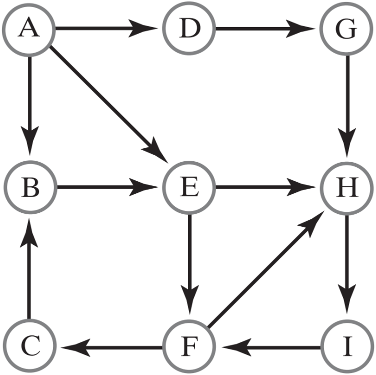
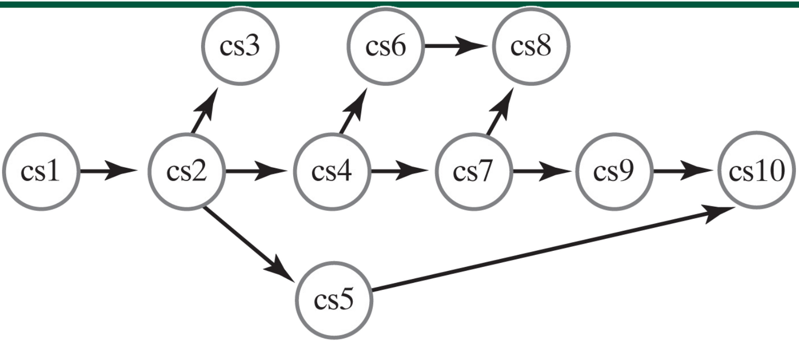


Figure: traversal(DepthFirst) wrt. ADT(Graph)

- **Question 5:** Search/Traverse ADT(Graph) above by search(DepthFirst) until node(G) is found.
  - **Solution 5.1:** origin(A) → `A, B, E, F, C, H, I, D, G`
  - **Solution 5.2:** origin(A) → order(Inorder, Postorder)

Non-Heuristic or Uninformed Search

Depth-First Search (DFS): Traversal wrt. ADT(Graph)



© 2019 Pearson Education, Inc

Figure: traversal(DepthFirst) wrt. ADT(Graph)

- **Question 6:** Search/Traverse ADT(Graph) above by search(DepthFirst) until node(cs3) is found.
  - **Solution 6.1:** origin(cs1) → `cs1, cs2, cs5, cs10, cs4, cs7, cs9, cs8, cs6, cs3`
  - **Solution 6.2:** origin(cs1) → order(Inorder, Postorder)

Non-Heuristic or Uninformed Search

Uniform-Cost Search (UCS): Dijkstra's Algo. wrt. ADT(Graph)

- **property(Optimality):**

  • No, it is not optimal because it does not always *thread* the least-cost path wrt. finding a solution.

- **property(Run-time Complexity):**

  • $O(V + E)$: where V = Number of vertices/nodes AND E = Number of edges/ties.

- **property(Space Complexity):**

  • $O(V)$: where V = Number of vertices/nodes.

- **property(Other):**

  • Usually goes deep into a particular path before *backtracking* & *advancing*. Also, search(DFS) uses lesser memory in comparison to search(BFS).

Non-Heuristic or Uninformed Search

Depth-Limited (Depth-First) Search (DL-DFS): wrt. ADT(Graph)

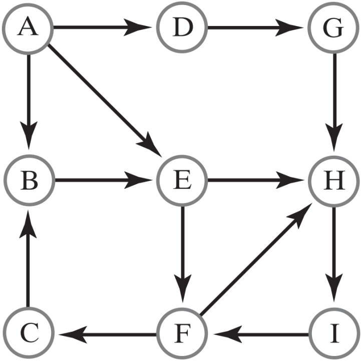


Figure: traversal(Depth-Limited Depth-First) wrt. ADT(Graph)

- **search(Depth-Limited Depth-First):**

- Graph traversal can begin at any node(Vertex) called: **origin** node(Vertex)
- node(Origin) *order(Indepth_by_Limit_&_Neighbor)* node(Adjacent/Neighbors)

- The order in which the Indepth adjacent/neighbor nodes are visited is NOT specific, and can it depend on the graph's implementation.

Non-Heuristic or Uninformed Search

Depth-Limited (Depth-First) Search (DL-DFS): wrt. ADT(Graph)

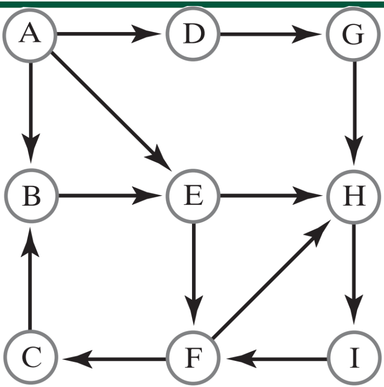


Figure: search(Depth-Limited Depth-First) wrt. ADT(Graph)

▶**Question 7:** Search/Traverse ADT(Graph) above by search(Depth-Limited) until node(I) is found wrt. a depth-limit = 3.

- **Solution 7.1:** origin(A) → `A, B, E, F, H, C, I`
- **Solution 7.2:** origin(A) → order(Inorder, Postorder)

Non-Heuristic or Uninformed Search

Depth-Limited (Depth-First) Search (DL-DFS): wrt. ADT(Graph)

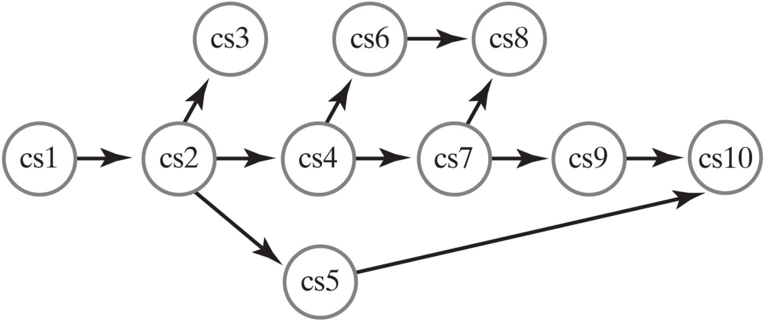


Figure: search(Depth-Limited Depth-First) wrt. ADT(Graph)

▶**Question 8:** Search/Traverse ADT(Graph) above by search(Depth-Limited) until node(cs9) is found wrt. a depth-limit = 3.

- **Solution 8.1:** origin(cs1) → `cs1, cs2, cs5, cs10, cs4, cs7, cs6, cs3`. Thus, NOT Found == cut-off.
- **Solution 8.2:** origin(cs1) → order(Inorder, Postorder)

Non-Heuristic or Uninformed Search

Depth-Limited (Depth-First) Search (DL-DFS): wrt. ADT(Graph)

- **property(Optimality):**
  - No, it is not optimal because it does not always *thread* the least-cost path wrt. finding a solution.
- **property(Run-time Complexity):**
  - $O(b^l)$: where b = Branching factor (average number of children per node) AND l = Depth-limit wrt. the search.
- **property(Space Complexity):**
  - $O(l)$: where l = Depth-limit wrt. the search.
- **property(Other):**
  - If the goal-node lies deeper than the depth/limit, it will not be found — even if it exists in the graph.

Non-Heuristic or Uninformed Search

Iterative Deepening (Depth-First) Search (ID-DFS): on Graphs

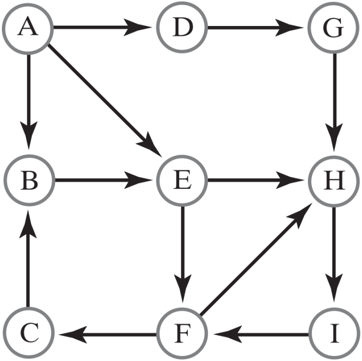


Figure: search(Iterative Deepening Depth-First) wrt. ADT(Graph)

- **search(Iterative Deepening Depth-First):**
- Graph traversal can begin at any node(Vertex) called: **origin** node(Vertex)
- node(Origin) $\underrightarrow{\textit{order}(\textit{Indepth\_by\_Step\_\&\_Neighbor})}$ node(Adjacent/Neighbors)

- The order in which the Indepth adjacent/neighbor nodes are visited is NOT specific, and can it depend on the graph's implementation.

Non-Heuristic or Uninformed Search

Iterative Deepening (Depth-First) Search (ID-DFS): on Graphs

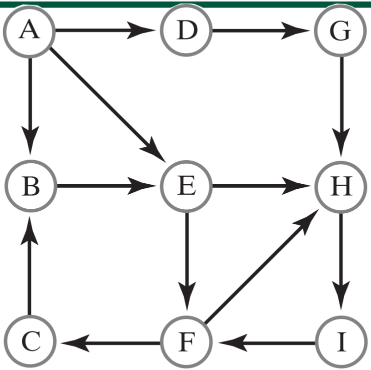


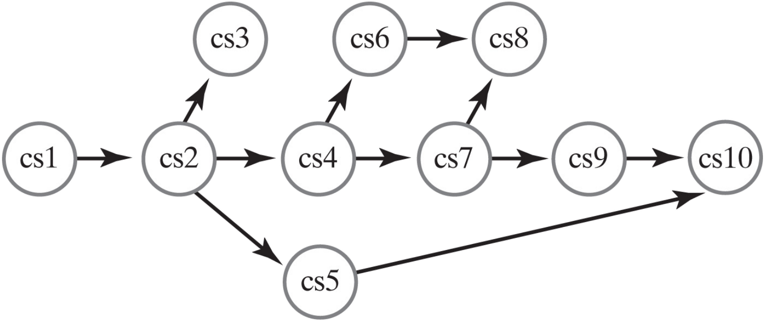
Figure: search(Iterative Deepening Depth-First) wrt. ADT(Graph)

- **Question 9:** Search/Traverse ADT(Graph) above by search(Iterative Deepening) until node(I) is found based on origin(A).

- **Solution:** depth/iteration-0(`A`); depth/iteration-1(`A, B, E, D`); depth/iteration-2(`A, B, E, F, H, D, G`); depth/iteration-3(`A, B, E, F, H, C, I`)

Non-Heuristic or Uninformed Search

Iterative Deepening (Depth-First) Search (ID-DFS): on Graphs



© 2019 Pearson Education, Inc.

Figure: search(Iterative Deepening Depth-First) wrt. ADT(Graph)

- **Question 10:** Search/Traverse ADT(Graph) above by search(Iterative Deepening) until node(cs6) is found based on origin(cs1).

- **Solution:** depth/iteration-0(`cs1`); depth/iteration-1(`cs1, cs2`); depth/iteration-2(`cs1, cs2, cs5, cs4, cs3`); depth/iteration-3(`cs1, cs2, cs5, cs10, cs4, cs7, cs6`)

Non-Heuristic or Uninformed Search

Iterative Deepening (Depth-First) Search (ID-DFS): on Graphs

- **property(Optimality):**
  - Yes, provided that each cost(Edge/Step) is uniform just like edges in graph(Unweighted)).
- **property(Run-time Complexity):**
  - $\underline{O(b^d)}$: where b = Branching factor (average number of children per node) AND d = Depth of the optimal solution.
- **property(Space Complexity):**
  - $\underline{O(b \cdot d)}$: where b = Branching factor (average number of children per node) AND d = Depth of the optimal solution.
- **property(Other):**
  - Just like traversal(DepthLimited), it's another variant of search(DepthFirst).

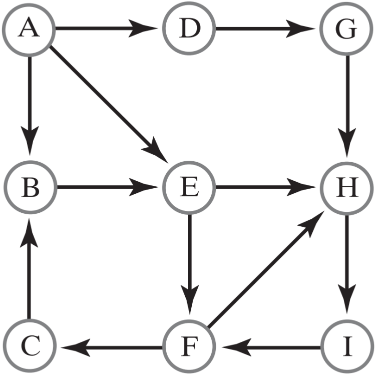
Non-Heuristic or Uninformed Search

Class/Game Activity

In-class Activity

Non-Heuristic or Uninformed Search

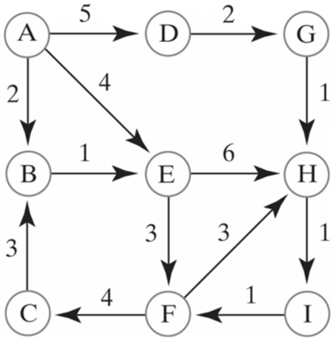
Class/Game Activity



1. Find **Shortest Path** from node(A → C) with respect to the *unweighted* graph above.

Non-Heuristic or Uninformed Search

Class/Game Activity



2. Find **Shortest Path** from node(A → C) with respect to the *weighted* graph above.

Questions?
&
Answers!