# COMP338: **ARTIFICIAL INTELLIGENCE**

Solving Problems by Searching

***Uninformed Search***

Dr. Radi Jarrar

STUDENTS-HUB.com

Uploaded By: anonymous

# Solving Problems by Searching

- *In which we see how an agent can find a sequence of actions that achieves its goals when no single action will do*
- Simple agents base their actions on a direct mapping from states to actions
- Goal-based agents, on the other hand, consider future actions and the desirability of their outcomes
- Problem-solving agents are Goal-based agents

STUDENTS-HUB.com
Uploaded By: anonymous

# Solving Problems by Searching

- Two main types of search
- **Uninformed** search algorithms—algorithms that are given no information about the problem other than its definition (blind search)
- **Informed** search algorithms, agents have some background knowledge about the problem

STUDENTS-HUB.com Uploaded By: anonymous

# Problem Solving Agents

- Intelligent agents aim to maximise their performance measure
- This is achieved if an agent adopt a goal and aim at satisfying it
- **Goal formulation**, based on the current situation and the agent's performance measure, is the first step in problem solving

STUDENTS-HUB.com
Uploaded By: anonymous

# Problem Solving Agents

- The agent's task is to find out how to act so that to reach a goal state
- **Problem formulation** is the process of deciding what actions and states to consider, given a goal
- *Example*: Assume you want to go to Jenin from BZU and you have never gone there before
- How many ways to go from? The agent has never been there (i.e., the agent doesn't yet know about the state that results from taking each action—each path) that is unknown environment and will take actions

STUDENTS-HUB.com
Uploaded By: anonymous

# Problem Solving Agents

- If the agent knows the map, this will provide the agent about the states it might get itself into and the actions it can take
- We will consider the problem of designing goal-based agents in fully observable, deterministic, discrete, and known environments

STUDENTS-HUB.com

Uploaded By: anonymous

# Problem Solving Agents

- **Search**: the process of looking for a sequence of actions that reaches the goal
- A search algorithm takes a problem as input and returns a solution in the form of an action sequence
- **Execution phase**: the execution of the solution that has been found by the search algorithm
- Thus, a simple design of an agent is “formulate, search, execute”

STUDENTS-HUB.com
Uploaded By: anonymous

# Well-defined problems

- A problem can be defined by 5 components:
- **Initial state**: the state the agent starts in. e.g., In(Birzeit)
- **Actions**: a description of the possible **actions** available to the agent.

  Given a particular state *s*, *ACTIONS(s)* returns the set of actions that can

  be executed in *s*. We say that each of these actions is **applicable** in *s*.
- For example, from the state *In(Birzeit)*, the applicable actions are

  *{Go(Atara), Go(Surda), Go(Jefna)}*.

STUDENTS-HUB.com
Uploaded By: anonymous

# Well-defined problems

- **Transition Model**: A description of what each action does; the formal name for this is the **transition model**, specified by a function *RESULT(s, a)* that returns the state that results from doing action a in state *s*.
- We also use the term **successor** to refer to any state reachable from a given state by a single action
- For example, we have *RESULT(In(Birzeit),Go(Surda)) = In(Surda)*

STUDENTS-HUB.com Uploaded By: anonymous

# Well-defined problems

- **Goal test**: determines whether a given state is a goal state
- There might be an explicit set of possible goal states, and the test simply checks whether the given state is one of them
- The agent's goal in our example is the singleton set {In(Jenin)}

STUDENTS-HUB.com
Uploaded By: anonymous

# Well-defined problems

- **Path Cost**: A function that assigns a numeric cost to each path. The problem-solving agent chooses a cost function that reflects its own performance measure
- E.g., To reach Jenin, time is of essence. So the cost of the path might be its length in kilometers
- Assume the cost of the path is the sum of the cost of individual actions along the path
- The **step cost** of taking action a in state sto reach state s' is denoted by c(s, a,s')

STUDENTS-HUB.com

Uploaded By: anonymous

# Problem Solving Agents

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
              state, some description of the current world state
              goal, a goal, initially null
              problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

STUDENTS-HUB.com
Uploaded By: anonymous

# Infrastructure for search algorithms

- STATE: the state in the state space to which the nodecorresponds
- PARENT: the node in the search tree that generated this node
- ACTION: the action that was applied to the parent to generate the node
- PATH-COST: the cost, traditionally denoted by g(n), of the path from the initial state to the node, as indicated by the parent pointers

STUDENTS-HUB.com
Uploaded By: anonymous

# Performance measures

- Search strategies are evaluated based on the following performance measures
- **Completeness**: Is there a solution to the problem? And is the algorithm guaranteed to find?
- **Optimality**: Does the strategy find the optimal (best) solution?
- **Time complexity**: How long does it take to find a solution?
- **Space complexity**: How much memory is needed to perform the search?

STUDENTS-HUB.com
Uploaded By: anonymous

# Performance measures

- Time and space complexity are measured in terms of
    - *b*: maximum branching factor of the search tree
    - d: depth of the least-cost solution (i.e., the number of steps along the path from the root)
  - m: the maximum length of any path in the state space

STUDENTS-HUB.com
Uploaded By: anonymous

# Performance measures

- Time and space complexity are always considered with respect to some measure of the problem difficulty
- Typically measured as the size of the state space graph, |V| + |E|, where V is the set of vertices (nodes) of the graph and E is the set of edges (links)
- This is appropriate when the graph is an explicit data structure that is input to the search program (e.g., a map)

STUDENTS-HUB.com Uploaded By: anonymous

# Uninformed Search

- **Uninformed search (blind search)**
  - No domain knowledge: number of edges (steps), path cost is still unknown
  - Agents know when it reaches a goal
- **Strategies** :

1. Breadth-first search (BFS): Expand shallowest node
2. Depth-first search (DFS): Expand deepest node
3. Depth-limited search (DLS): Depth first with depth limit
4. Iterative-deepening search (IDS): DLS with increasing limit
5. Uniform-cost search (UCS): Expand least cost node

STUDENTS-HUB.com
Uploaded By: anonymous

# Breadth First Search (BFS)

STUDENTS-HUB.com
Uploaded By: anonymous

# Breadth-First Search

- The root node is expanded first, then all the successors of the root node are expanded next, then *their* successors, and so on
- Generally, all nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded
- The *shallowest* unexpanded node is chosen for expansion. This is achieved very simply by using a FIFO queue for the frontier

STUDENTS-HUB.com
Uploaded By: anonymous

# Breadth-First Search

- Thus, new nodes (which are always deeper than their parents) go to the back of the queue, and old nodes, which are shallower than the new nodes, get expanded first

STUDENTS-HUB.com
Uploaded By: anonymous

# Breadth-First Search

- *Algorithm:*

1. Enqueue the root (i.e., initial node)
2. Dequeue a node and examine it:
    1. If the element sought is found in this node, quit the search and return a result
    2. Otherwise enqueue any successors (the direct child nodes) that have not been discovered yet
3. If the queue is empty, every node on the graph has been examined – quit the search and return "not found“
4. Repeat from Step 2.

STUDENTS-HUB.com

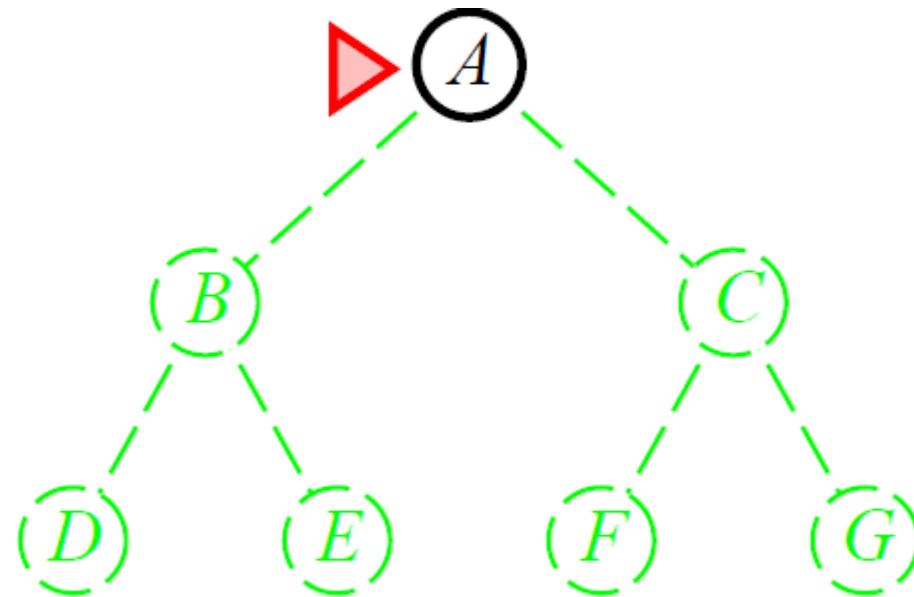
Uploaded By: anonymous

# Breadth-First Search

```
function BREADTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Queue.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.dequeue()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.enqueue(neighbor)

    return FAILURE
```

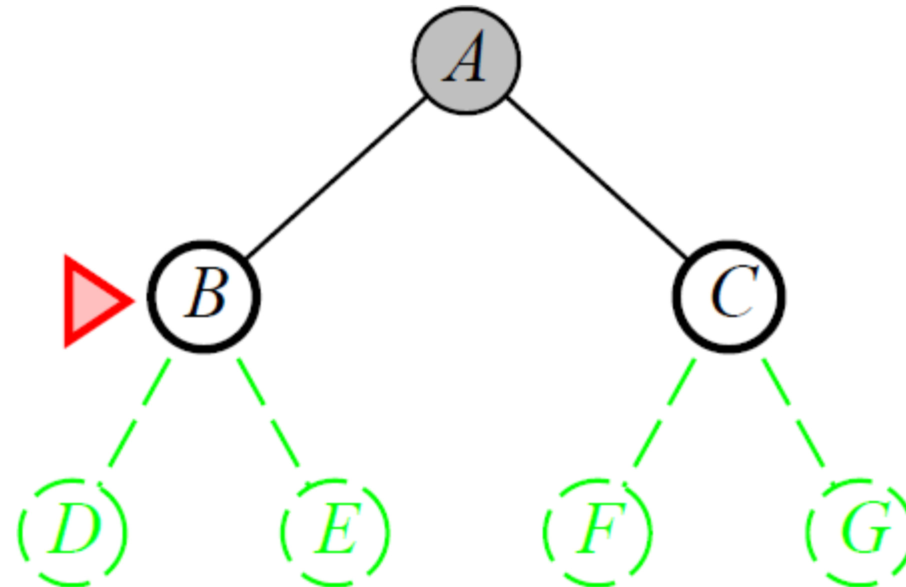
STUDENTS-HUB.com
Uploaded By: anonymous

# Breadth-First Search




STUDENTS-HUB.com
Uploaded By: anonymous

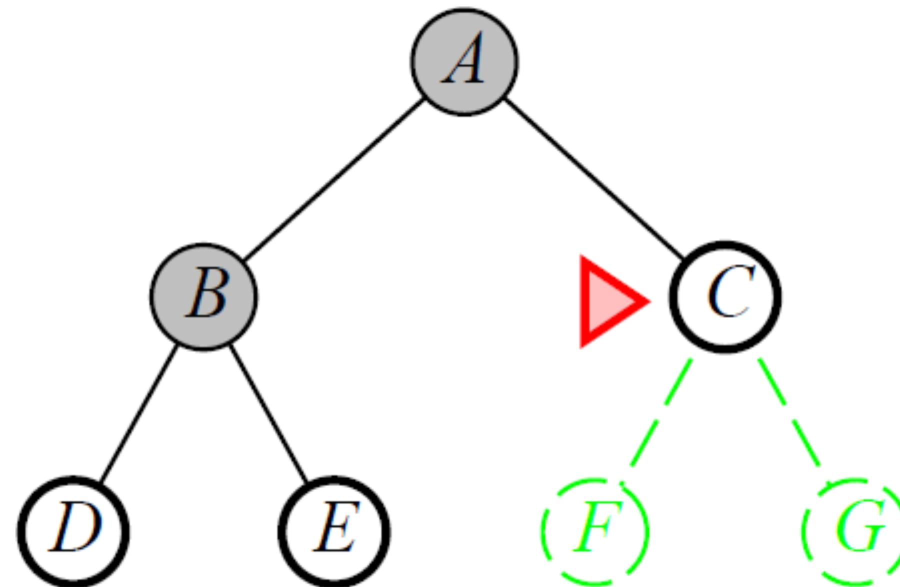
# Breadth-First Search





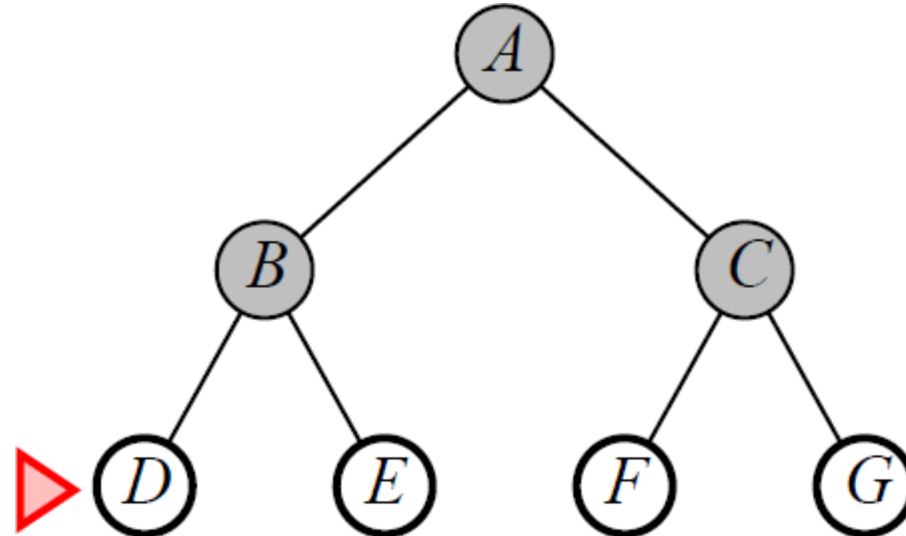
Uploaded By: anonymous

# Breadth-First Search




STUDENTS-HUB.com
Uploaded By: anonymous

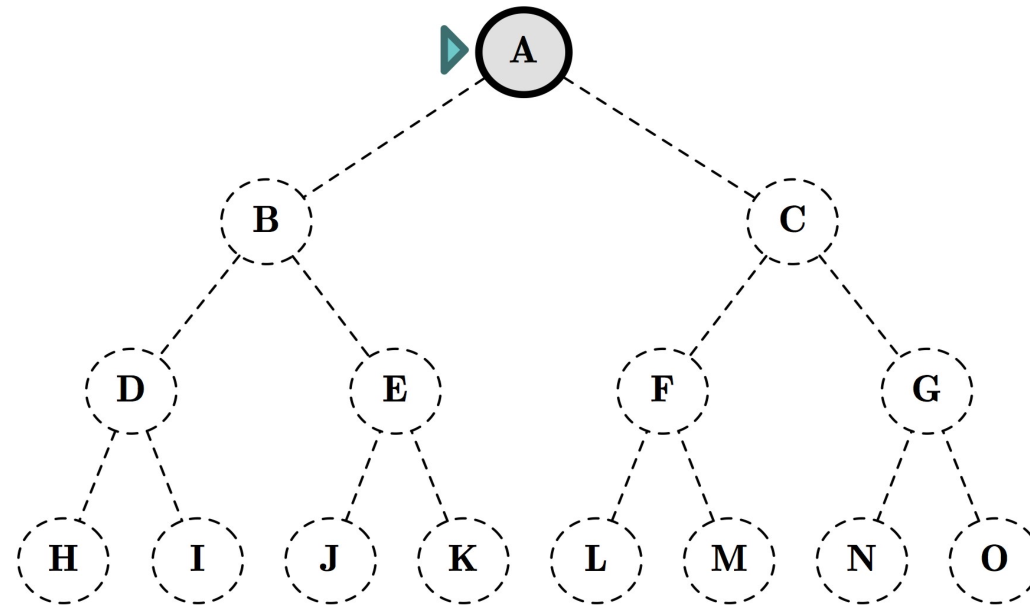
# Breadth-First Search




STUDENTS-HUB.com

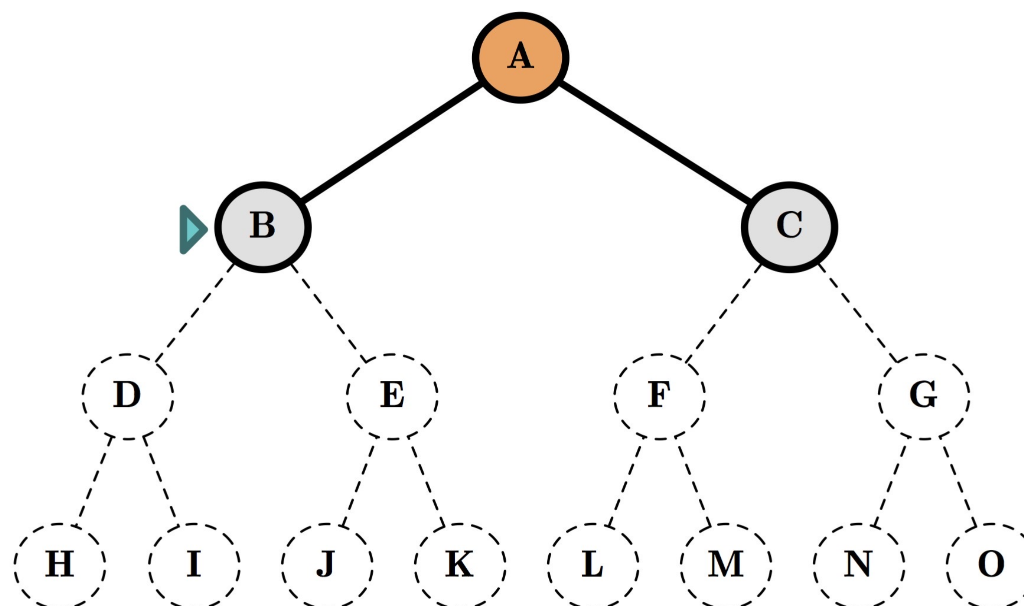
Uploaded By: anonymous

# Breadth-First Search




STUDENTS-HUB.com
Uploaded By: anonymous

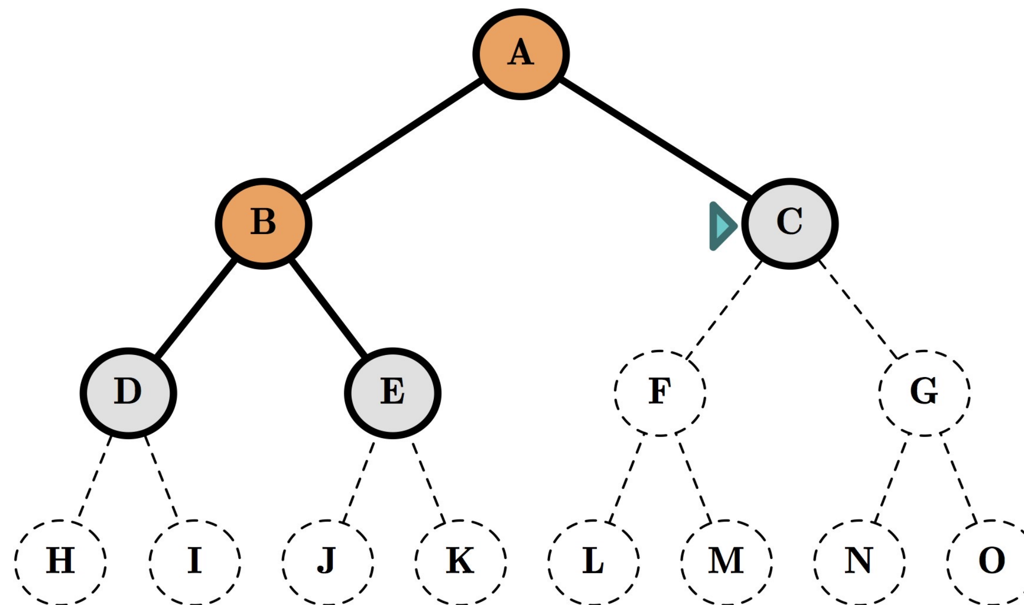
# Breadth-First Search




STUDENTS-HUB.com

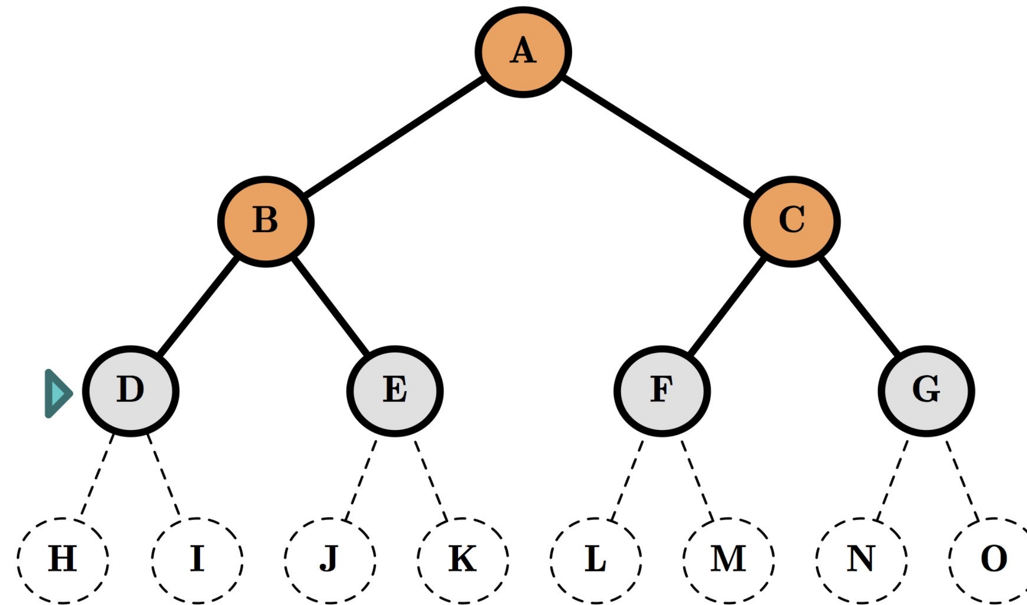
Uploaded By: anonymous

# Breadth-First Search



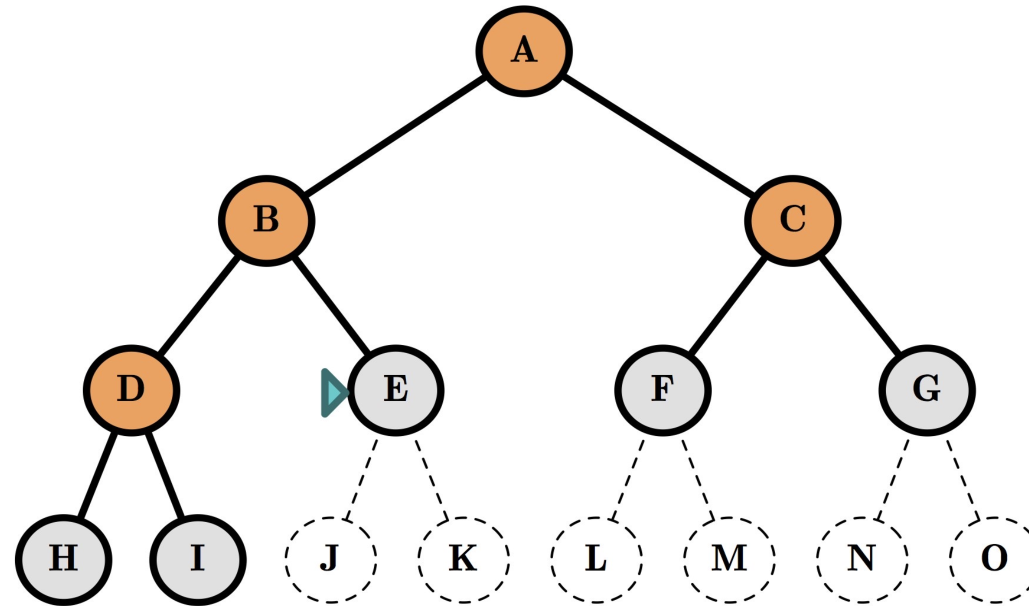

STUDENTS-HUB.com
Uploaded By: anonymous

# Breadth-First Search



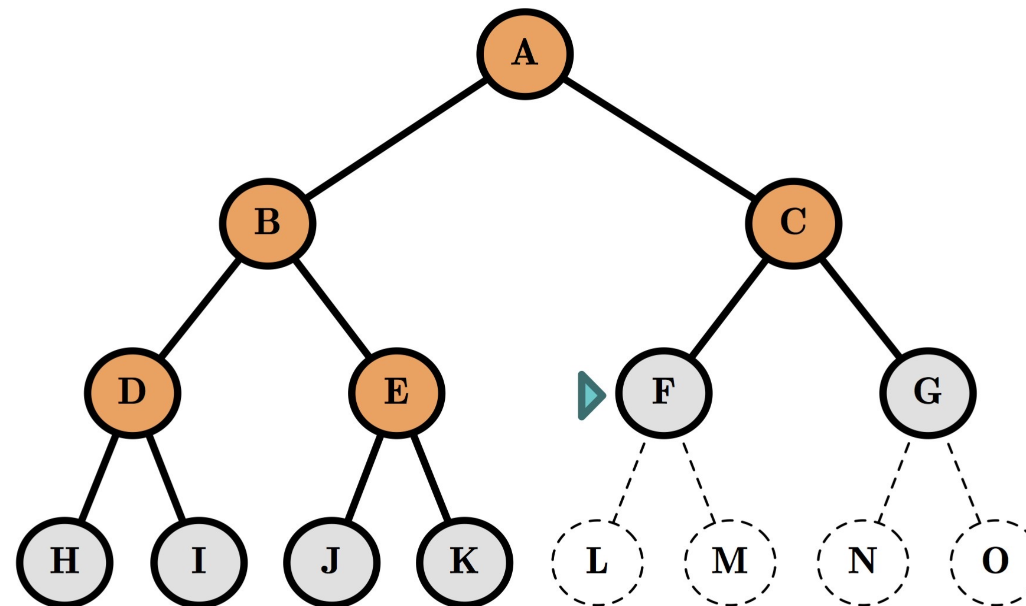

STUDENTS-HUB.com
Uploaded By: anonymous

# Breadth-First Search




STUDENTS-HUB.com
Uploaded By: anonymous

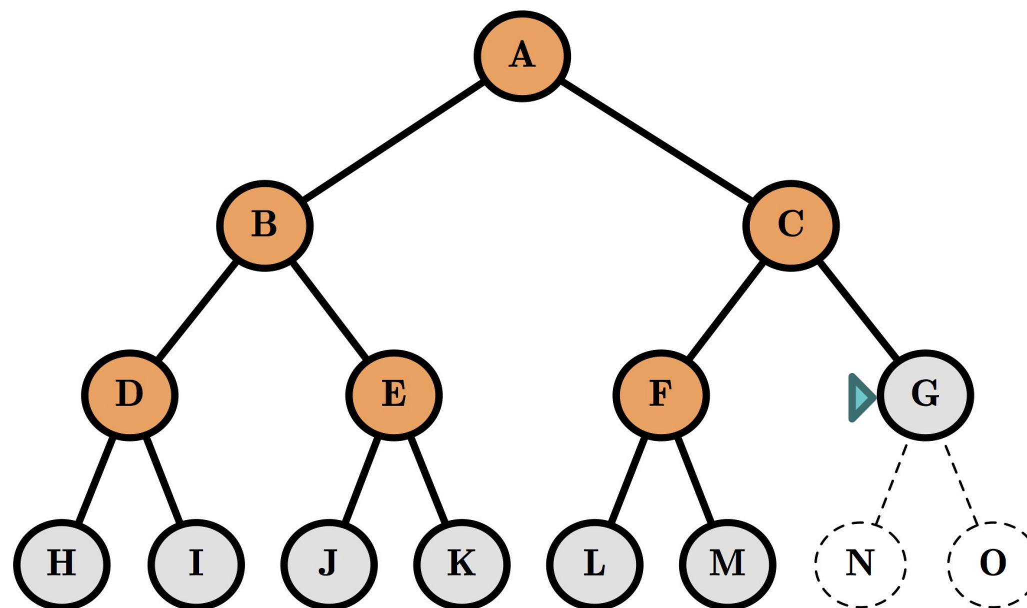
# Breadth-First Search





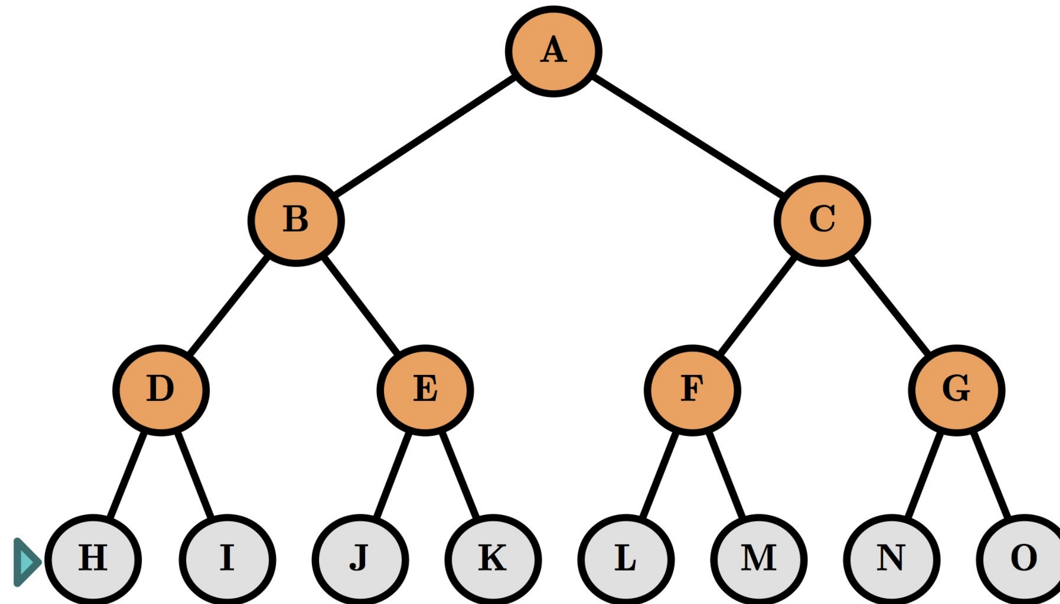
Uploaded By: anonymous

# Breadth-First Search




STUDENTS-HUB.com Uploaded By: anonymous

# Breadth-First Search




STUDENTS-HUB.com
Uploaded By: anonymous

# Properties of breadth-first search

- Complete?

STUDENTS-HUB.com
Uploaded By: anonymous

# Properties of breadth-first search

- Complete?
  - Yes. If b is finite
- Optimal?

STUDENTS-HUB.com

Uploaded By: anonymous

# Properties of breadth-first search

- Complete?
  - Yes. If b is finite
- Optimal?
  - Yes (if cost = 1 per step)
- Time??

STUDENTS-HUB.com Uploaded By: anonymous

# Properties of breadth-first search

- Complete?
  - Yes. If b is finite
- Optimal?
  - Yes (if cost = 1 per step)
- Time?
  - $b + b^2 + b^3 + \cdots + b^d = O(b^d)$
- Space?

STUDENTS-HUB.com
Uploaded By: anonymous

# Properties of breadth-first search

- Complete?
  - Yes. If b is finite
- Optimal?
  - Yes (if cost = 1 per step)
- Time?
  - $b + b^2 + b^3 + \cdots + b^d = O(b^d)$
- Space?
  - $O(b^{d+1})$ (keeps every node in memory)
- Implementation?
  - FIFO (Queue)

STUDENTS-HUB.com

Uploaded By: anonymous

# Uniform-Cost Search (UCS)

STUDENTS-HUB.com
Uploaded By: anonymous

# Uniform-cost search (UCS)

- The edges in the search graph may have weights (costs)
- We want the cheapest not shallowest solution.
- UCS visits the next node which has the least total cost from the root, until a goal state is reached.
- Similar to BFS but with an evaluation of the cost for each reachable node.
- $g(n)$ = path cost(n) = sum of individual edge costs to reach the current node.

STUDENTS-HUB.com Uploaded By: anonymous

# Uniform-cost search

- BFS is optimal because it always expands the *shallowest* unexpanded node
- **Uniform-cost search** expands the node n with the *lowest path cost* g(n)
- This is done by storing the frontier as a priority queue ordered by g
- Uniform-cost search on a graph is identical to the general graph search algorithm except for the use of a priority queue and the addition of an extra check in case a shorter path to a frontier state is discovered

STUDENTS-HUB.com
Uploaded By: anonymous

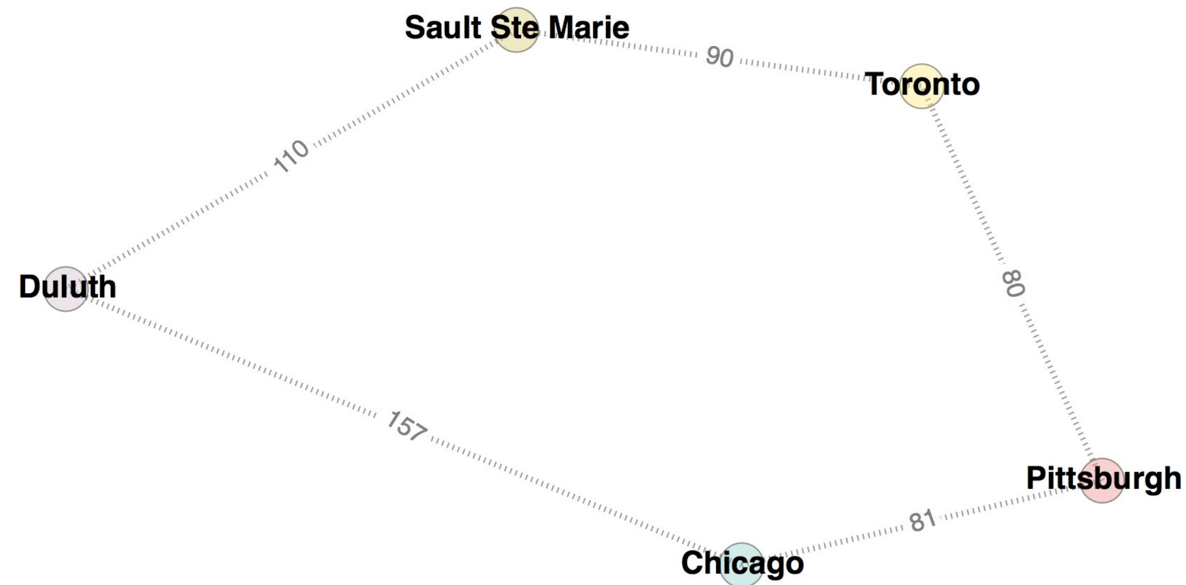
# Uniform-cost search - Algorithm

```
function UNIFORM-COST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :  /* Cost f(n) = g(n) */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

STUDENTS-HUB.com

Uploaded By: anonymous

# Uniform-cost search - Example




Go from Chicago to Sault Ste Marie. Using BFS, we would find Chicago-Duluth-Sault Ste Marie. However, using UCS, we would find Chicago-Pittsburgh-Toronto-Sault Ste Marie, which is actually the shortest path!

STUDENTS-HUB.com

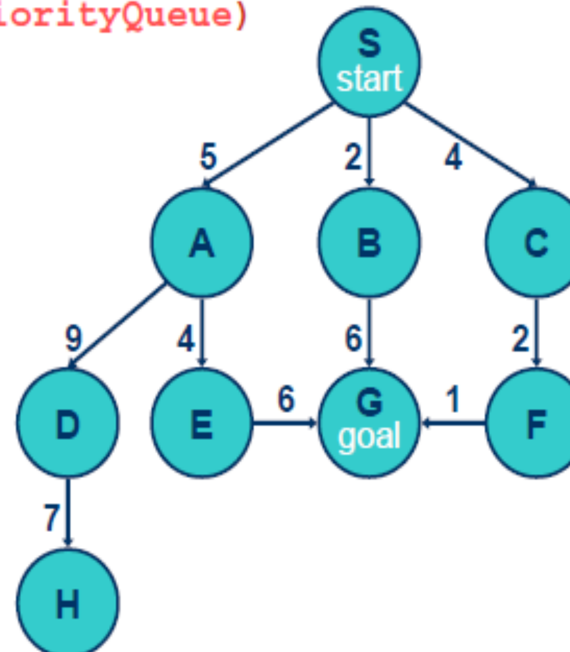
Uploaded By: anonymous

# Uniform-cost search - Example

generalSearch(problem, priorityQueue)

# of nodes tested: 0, expanded: 0

| expnd. node | nodes list |
| --- | --- |
| | {S} |





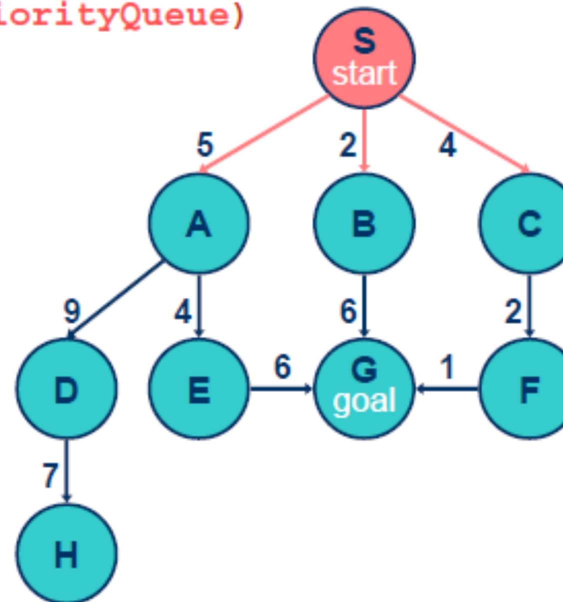

Uploaded By: anonymous

# Uniform-cost search - Example

```
generalSearch(problem, priorityQueue)
```

# of nodes tested: 1, expanded: 1

| expnd. node | nodes list |
| --- | --- |
| | {S:0} |
| S not goal | {B:2,C:4,A:5} |



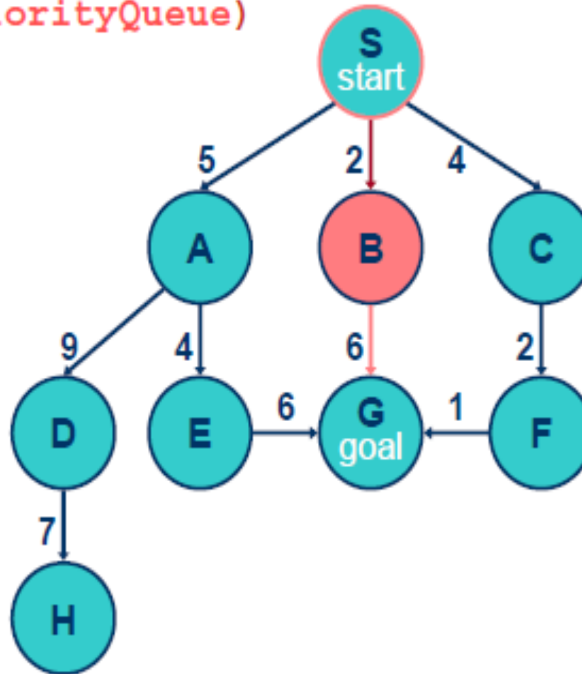

 Uploaded By: anonymous

# Uniform-cost search - Example

```
generalSearch(problem, priorityQueue)
```

# of nodes tested: 2, expanded: 2

| expnd. node | nodes list |
|---|---|
| | {S} |
| S | {B:2,C:4,A:5} |
| B not goal | {C:4,A:5,G:2+6} |



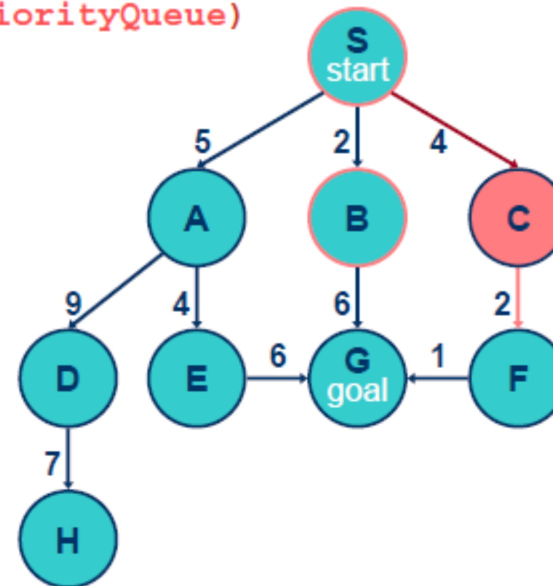

STUDENTS-HUB.com
Uploaded By: anonymous

# Uniform-cost search - Example

`generalSearch(problem, priorityQueue)`

\# of nodes tested: 3, expanded: 3

| expnd. node | nodes list |
|---|---|
| | {S} |
| S | {B:2,C:4,A:5} |
| B | {C:4,A:5,G:8} |
| C not goal | {A:5,F:4+2,G:8} |





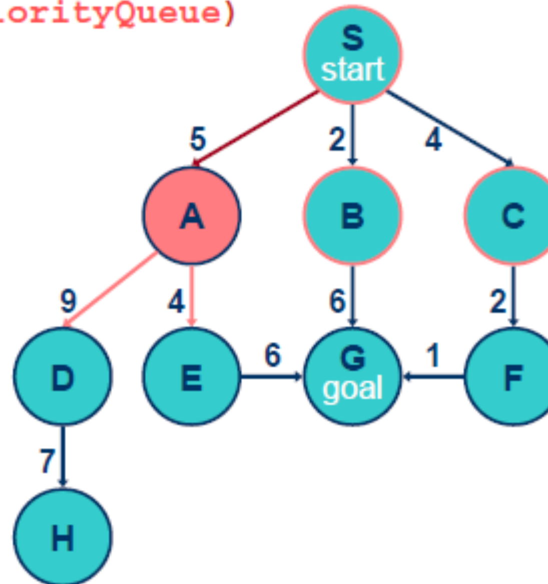
Uploaded By: anonymous

# Uniform-cost search - Example

generalSearch(problem, priorityQueue)

# of nodes tested: 4, expanded: 4

| expnd. node | nodes list |
|---|---|
| | {S} |
| S | {B:2,C:4,A:5} |
| B | {C:4,A:5,G:8} |
| C | {A:5,F:6,G:8} |
| A not goal | {F:6,G:8,E:5+4, D:5+9} |





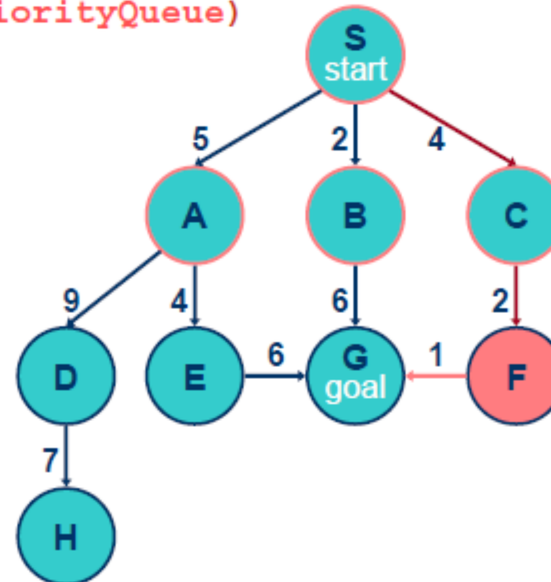
Uploaded By: anonymous

# Uniform-cost search - Example

**generalSearch(problem, priorityQueue)**

# of nodes tested: 5, expanded: 5

| expnd. node | nodes list |
|---|---|
| | {S} |
| S | {B:2,C:4,A:5} |
| B | {C:4,A:5,G:8} |
| C | {A:5,F:6,G:8} |
| A | {F:6,G:8,E:9,D:14} |
| F not goal | {G:4+2+1,G:8,E:9, D:14} |





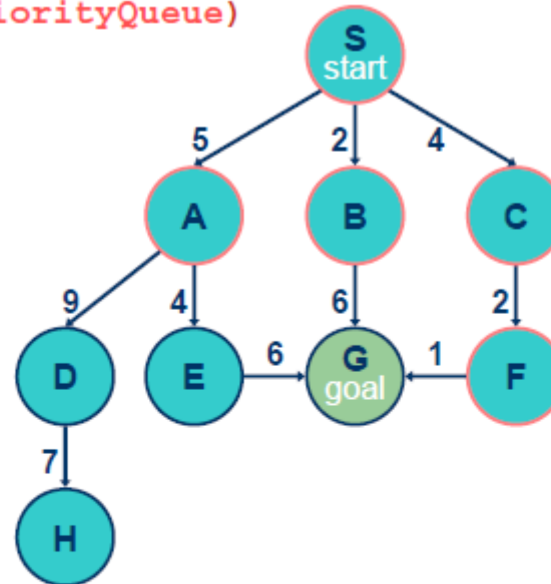
Uploaded By: anonymous

# Uniform-cost search - Example

```
generalSearch(problem, priorityQueue)
```

# of nodes tested: 6, expanded: 5

| expnd. node | nodes list |
|---|---|
| | {S} |
| S | {B:2,C:4,A:5} |
| B | {C:4,A:5,G:8} |
| C | {A:5,F:6,G:8} |
| A | {F:6,G:8,E:9,D:14} |
| F | {G:7,G:8,E:9,D:14} |
| G goal | {G:8,E:9,D:14}<br>no expand |





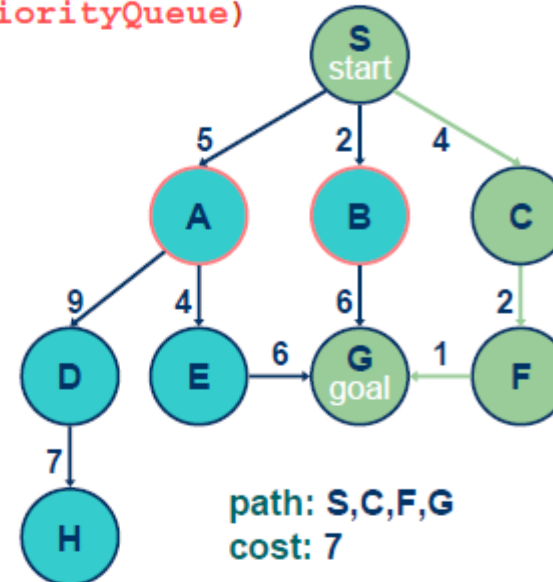
Uploaded By: anonymous

# Uniform-cost search - Example

```
generalSearch(problem, priorityQueue)
```

# of nodes tested: 6, expanded: 5

| expnd. node | nodes list |
|---|---|
| | {S} |
| S | {B:2,C:4,A:5} |
| B | {C:4,A:5,G:8} |
| C | {A:5,F:6,G:8} |
| A | {F:6,G:8,E:9,D:14} |
| F | {G:7,G:8,E:9,D:14} |
| G | {G:8,E:9,D:14} |




STUDENTS-HUB.com

Uploaded By: anonymous

# Uniform-cost search

- **Complete** Yes, if solution has a finite cost and the step cost is positive.
- **Time:** much larger than $\mathbf{b^d}$, and just $\mathbf{b^d}$ if all steps have the same cost.
- **Space:** same as time
- **Optimal:** Yes
- **Implementation**: fringe = queue ordered by path cost $g(n)$, lowest first = Heap!
- Requires that the goal test being applied when a node is removed from the nodes list rather than when the node is first generated while its parent node is expanded.
- While complete and optimal, UCS explores the search space in every direction because no information is provided about the goal!

STUDENTS-HUB.com Uploaded By: anonymous

# BSF vs. UCS

- Breadth-first search (BFS) is a special case of uniform-cost search when all edge costs are positive and identical.
- Breadth-first always expands the shallowest node
  - Only optimal if all step-costs are equal
- Uniform-cost considers the overall path cost
  - Optimal for any (reasonable) cost function
    - non-zero, positive
  - Gets stuck down in trees with many fruitless, short branches
    - Low path cost but no goal node
- Both are complete for non-extreme problems
  - Finite number of branches
  - Strictly positive search function

STUDENTS-HUB.com
Uploaded By: anonymous

# Depth-First Search (DFS)

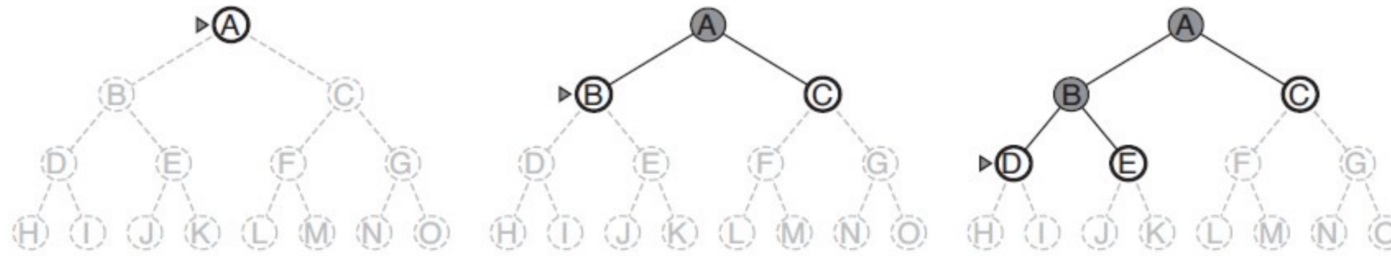
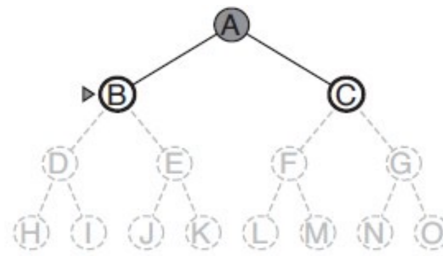
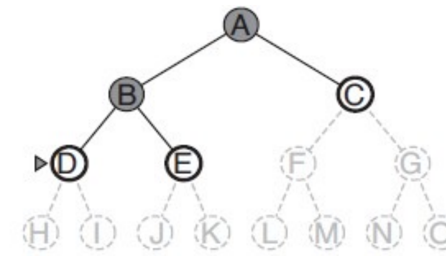
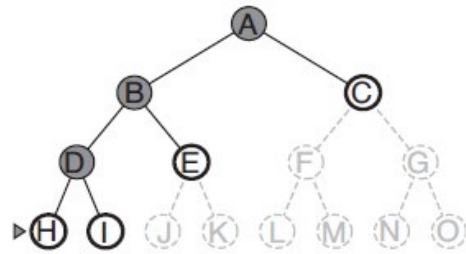
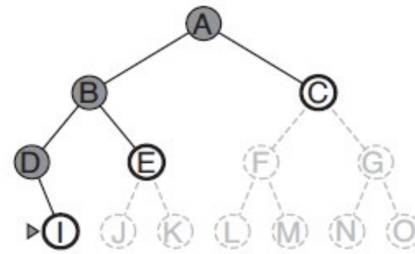
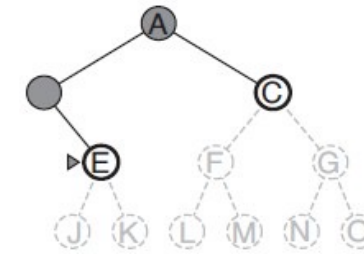
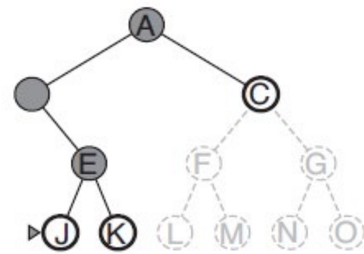
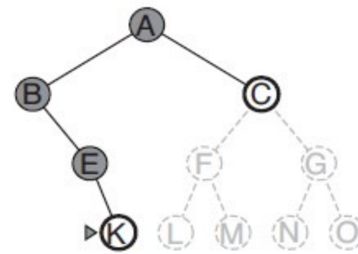
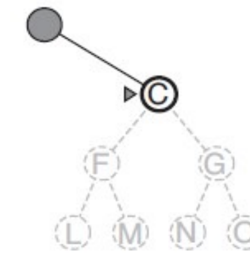
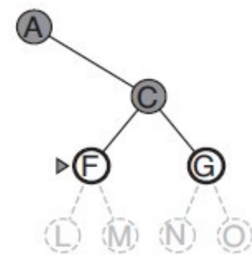
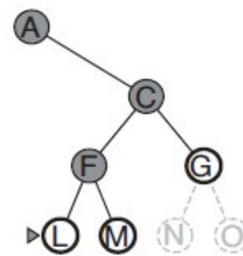
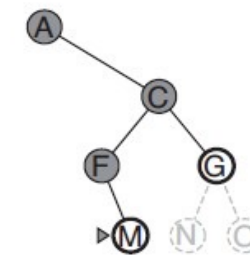
STUDENTS-HUB.com
Uploaded By: anonymous

# Depth-First Search

- Expands the *deepest* node in the current frontier of the search tree
- The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors
- BFS uses a FIFO queue, DFS uses a LIFO queue

STUDENTS-HUB.com
Uploaded By: anonymous

# Depth-First Search

STUDENTS-HUB.com
Uploaded By: anonymous

# Depth-First Search - Algorithm

```
function DEPTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Stack.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.pop()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.push(neighbor)

    return FAILURE
```

STUDENTS-HUB.com
Uploaded By: anonymous

# Depth-First Search - Properties

**Complete**: No: fails in infinite-depth spaces, spaces with loops

- Modify to avoid repeated states along path
- But it is complete in finite spaces

**Time**: $O(b^m) = 1 + b + b^2 + b^3 + \dots + b^m$

- bad if $m$ is much larger than $d$
- but if solutions are dense, may be much faster than breadth-first

**Space**: *O(bm)*, linear space complexity! (needs to store only a single path from the root to a leaf node, along with the remaining unexpanded sibling nodes for each node on thepath, hence the $m$ factor.)

**Optimal**: No

**Implementation:** fringe using LIFO principle (Stack)

*Notes:*
*b: maximum branching factor of the search tree*
*d: depth of the least-cost solution*
*m: maximum depth of the state space (may be ∞)*

STUDENTS-HUB.com

Uploaded By: anonymous

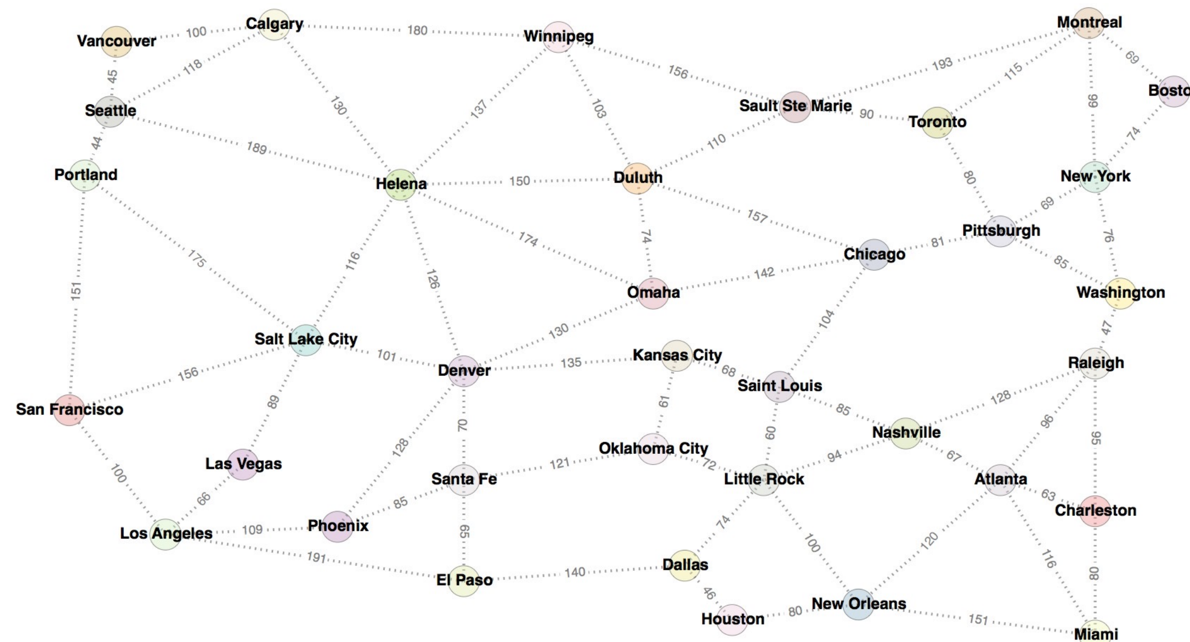
# BFS vs. DFS

- DFS goes off into one branch until it reaches a leaf node
  - Not good if the goal is on another branch
  - Neither complete nor optimal
  - Uses much less space than BFS
    - Much fewer visited nodes to keep track, smaller fringe
- BFS is more careful by checking all alternatives
  - Complete and optimal (Under most circumstances)
  - Very memory-intensive
- For a large tree, BFS memory requirements maybe excessive
- For a large tree, a depth-first search may take an excessively long time to find even a very nearby goal node.

STUDENTS-HUB.com Uploaded By: anonymous

# Depth-Limited Search

- If we know some knowledge about the problem, may be we don't need to go to a full depth.




Idea: any city can be reached from another city in at most $L$ steps with $L < 6$

STUDENTS-HUB.com
Uploaded By: anonymous

# Depth-Limited Search

- Overcame the problem of infinite-path in DFS by using a predetermined depth limit $L$
- Nodes at depth $L$ are treated as if they have no successors
- This solves the infinite-path problem but introduces an additional source of incompleteness if we choose $L < d$ that is, the shallowest goal is beyond the depth limit. (This is likely when d is unknown.)
- Must keep track of the depth
- Complete? no (if goal beyond L ($L<d$), or infinite branch length)
- Depth-limited search is non-optimal even if we choose $L > d$
- Time complexity is $O(b^L)$ and its space complexity is $O(bL)$
- DFS can be viewed as a special case of depth-limited search with $L = \infty$

STUDENTS-HUB.com Uploaded By: anonymous

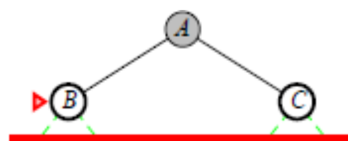
# Iterative-Deepening Search

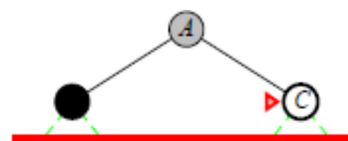
- **Iterative deepening search** (or iterative deepening depth-first search) is a general strategy, often used in combination with DFS, that finds the best depth limit
- Applies LIMITED-DEPTH with increasing depth limits
  - Combines advantages of BFS and DFS
  - It searches to depth 0 (root only), then if that fails it searches to depth 1, then depth 2, etc. (gradually increasing the limit—first 0, then 1, then 2, and so on—until a goal is found)
- This will occur when the depth limit reaches d, the depth of the shallowest goal node
- It terminates when a solution is found or if the depth limited search returns failure, meaning that no solution exists

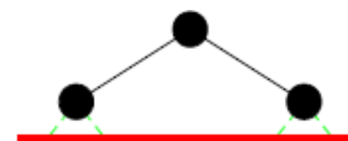
STUDENTS-HUB.com
Uploaded By: anonymous

# Iterative-Deepening Search

Limit = 1

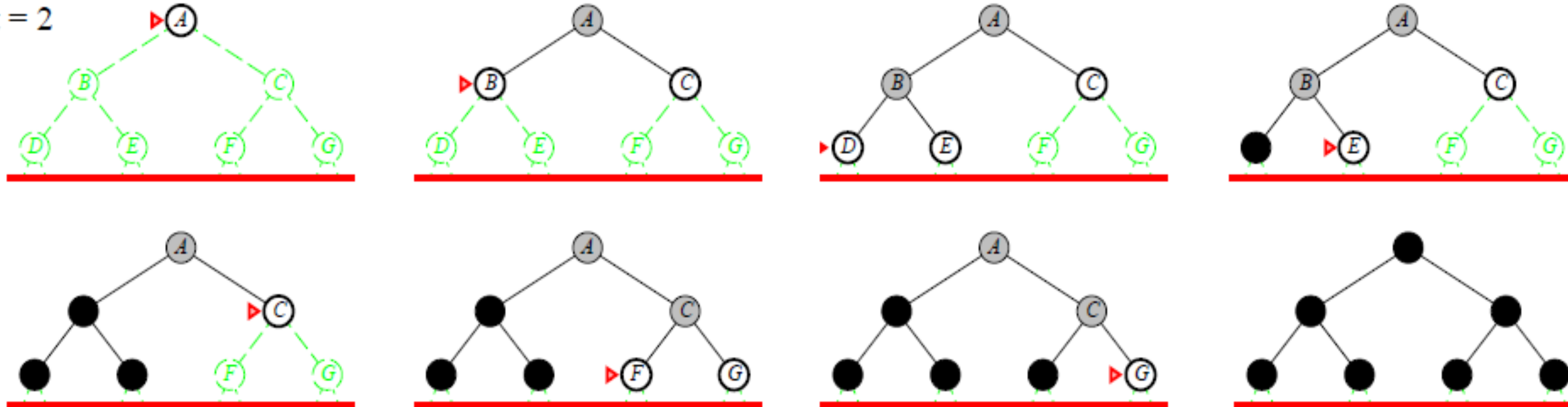


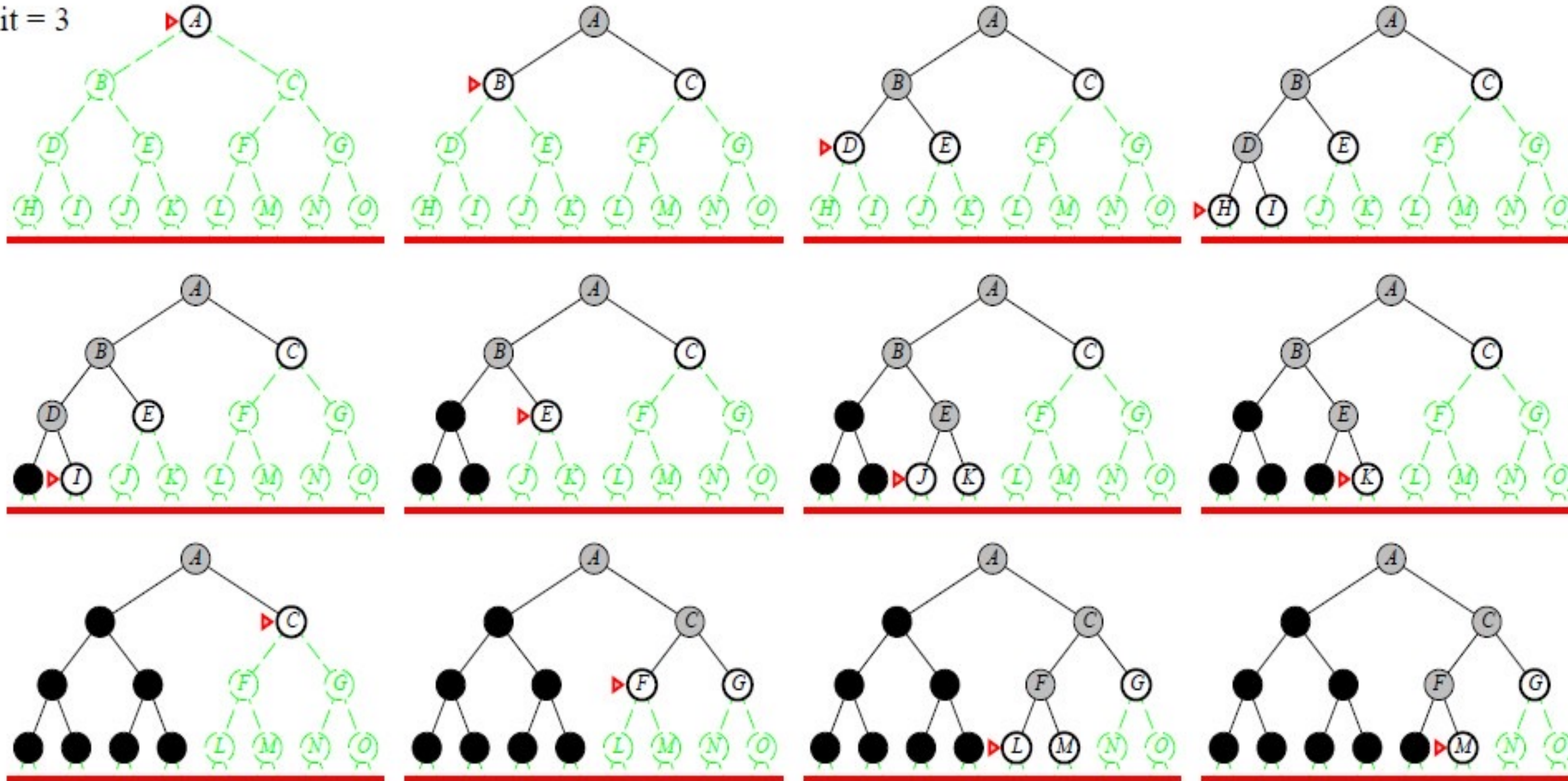
STUDENTS-HUB.com Uploaded By: anonymous

# Iterative-Deepening Search




STUDENTS-HUB.com
Uploaded By: anonymous

# Iterative-Deepening Search





Uploaded By: anonymous

# Iterative-Deepening Search

- If a goal node is found, it is a nearest node and the path to it pushed into the stack.
- Required stack size is limit of search depth (plus 1).
- Many states are expanded multiple times
  - doesn't really matter because the number of those nodes is small
- In practice, one of the best uninformed search methods
  - for large search spaces, unknown depth

STUDENTS-HUB.com
Uploaded By: anonymous

# Properties of Iterative-Deepening Search

- Iterative deepening combines the benefits of DFS and BFS
- Complete: Yes (when the branching factor is finite)
- Time: $(d+1)b^0 + db^1 + (d-1)b^2 + ... + b^d = O(b^d)$
- Space: Like DFS, its memory requirements are modest O(bd)
- Optimal: Yes, if step cost = 1

STUDENTS-HUB.com
Uploaded By: anonymous

# Summary

| | BFS | Uniform-Cost | DFS | Depth-Limited | Iterative Deepening |
|---|---|---|---|---|---|
| Complete | Yes | Yes | No | No | Yes |
| Optimal | Yes | Yes | No | No | Yes |
| Time | $O(b^{d+1})$ | $\lfloor b^{C^*/\varepsilon} \rfloor$ | $b^m$ | $b^L$ | $b^d$ |
| Space | $O(b^{d+1})$ | $\lfloor b^{C^*/\varepsilon} \rfloor$ | bm | bL | bd |

STUDENTS-HUB.com
Uploaded By: anonymous

# Summary

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored
- Variety of uninformed search strategies
- Iterative deepening search uses only linear space and not much more time than other uninformed algorithms
- Graph search can be exponentially more efficient than tree search

STUDENTS-HUB.com
Uploaded By: anonymous

# Summary

- Breadth-first search (BFS) and depth-first search (DFS) are the foundation for all other search techniques.
- We might have a weighted tree, in which the edges connecting a node to its children have differing "weights"
  - We might therefore look for a "least cost" goal
- The searches we have been doing are blind searches, in which we have no prior information to help guide the search

STUDENTS-HUB.com Uploaded By: anonymous

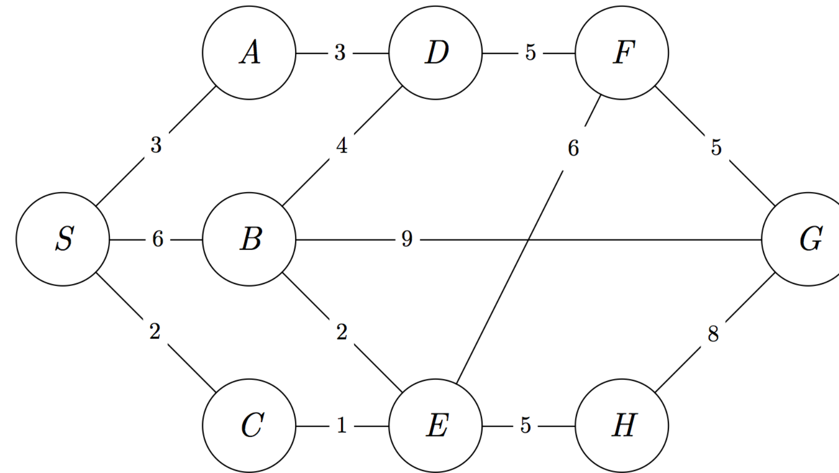
# Summary – tips on selecting an algorithm

**Breadth-First Search**:
– Solutions are known to be shallow
**Uniform-Cost Search**:
– Actions have varying costs
– Least cost solution is the required
*This is the only uninformed search that take costs into account.*
**Depth-First Search:**
– Many solutions exist
– Know (or have a good estimate of) the depth of solution
**Iterative-Deepening Search**:
– Space is limited and the shortest solution path is required

STUDENTS-HUB.com

Uploaded By: anonymous

# Exercise

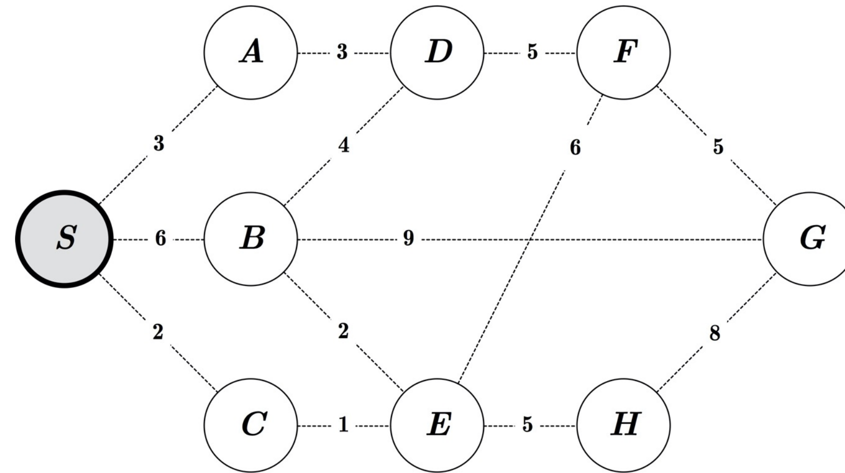



**Question:** What is the **order of visits of the nodes** and the **path** returned by BFS, DFS and UCS?

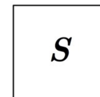
STUDENTS-HUB.com

Uploaded By: anonymous

# Exercise




**Queue:**

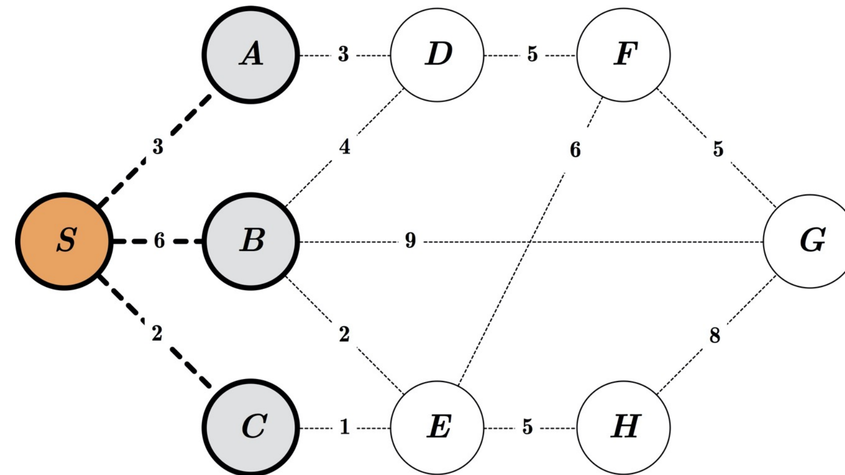



**Order of Visit:**



Uploaded By: anonymous

# Exercise




**Queue:**

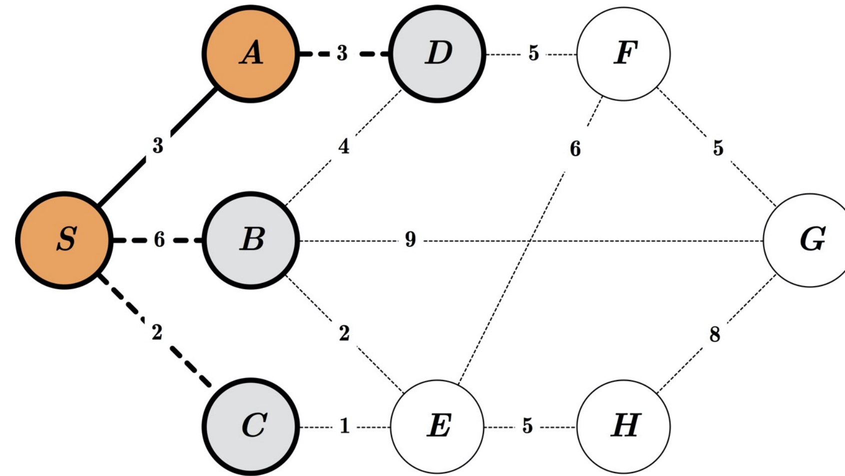
| S | A | B | C |
|---|---|---|---|

**Order of Visit:**

*S*

STUDENTS-HUB.com
Uploaded By: anonymous

# Exercise




**Queue:**

| S | A | B | C | D |
|---|---|---|---|---|

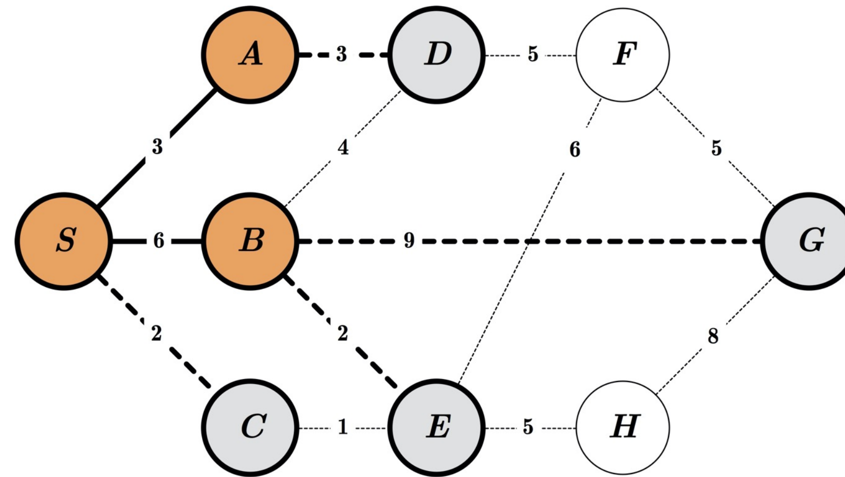
**Order of Visit:**

*S* *A*

STUDENTS-HUB.com

Uploaded By: anonymous

# Exercise




**Queue:**

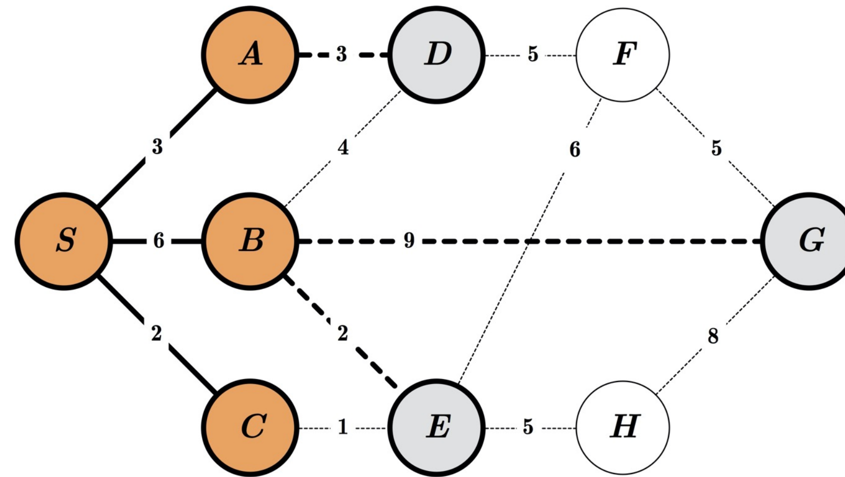
| *S* | *A* | *B* | *C* | *D* | *E* | *G* |
|---|---|---|---|---|---|---|

**Order of Visit:**

*S* *A* *B*


Uploaded By: anonymous

# Exercise




**Queue:**

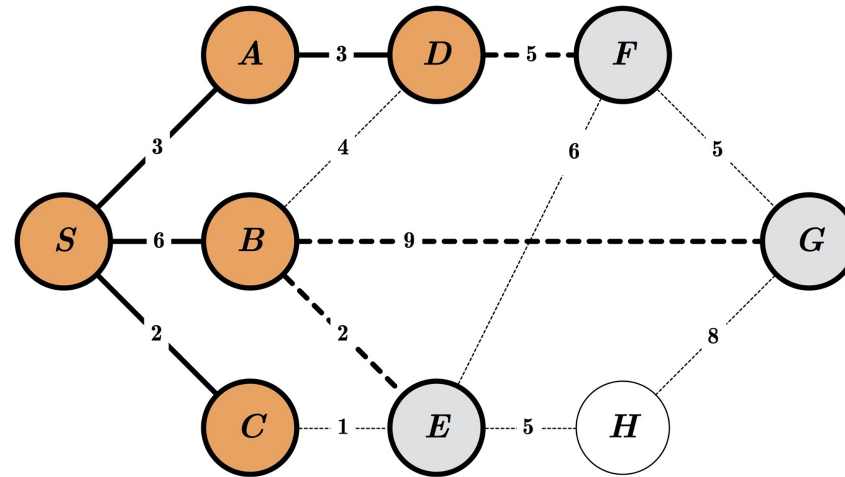
| S | A | B | C | D | E | G |
|---|---|---|---|---|---|---|

**Order of Visit:**

*S* *A* *B* *C*

 Uploaded By: anonymous

# Exercise




**Queue:**

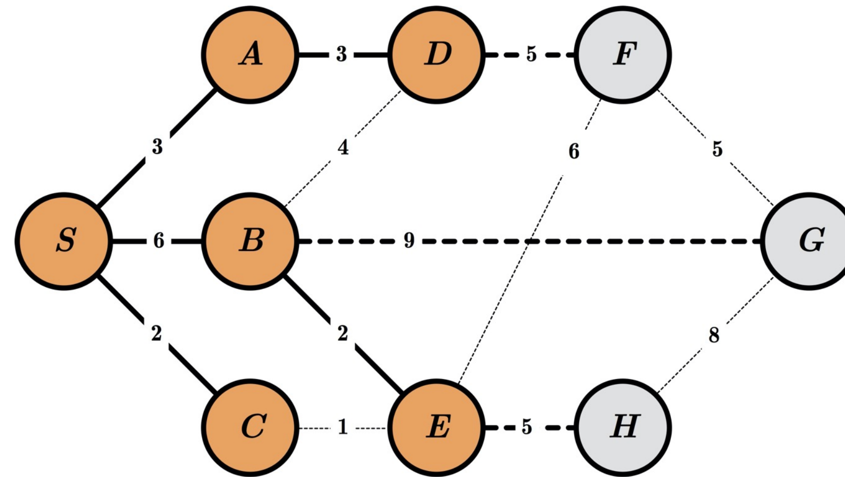
| $S$ | $A$ | $B$ | $C$ | $D$ | $E$ | $G$ | $F$ |
|---|---|---|---|---|---|---|---|

**Order of Visit:**

$S \quad A \quad B \quad C \quad D$

STUDENTS-HUB.com

Uploaded By: anonymous



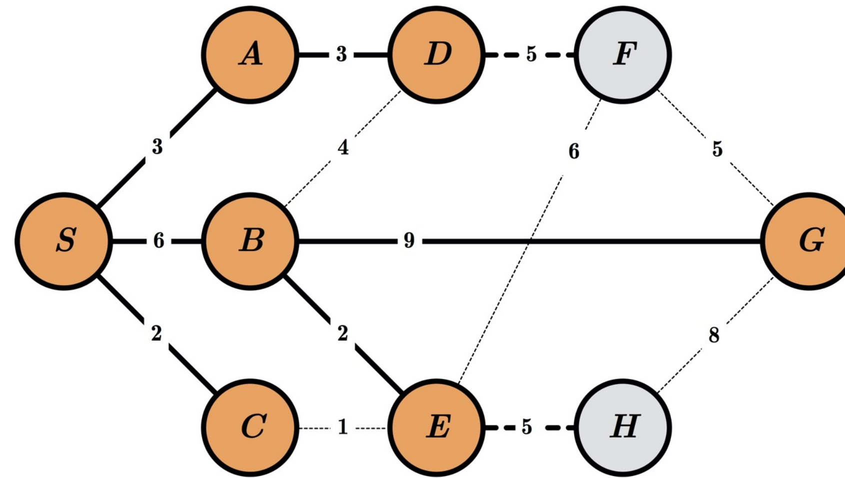

**Queue:**

| S | A | B | C | D | E | G | F | H |
|---|---|---|---|---|---|---|---|---|

**Order of Visit:**

*S* *A* *B* *C* *D* *E*

STUDENTS-HUB.com
Uploaded By: anonymous




**Queue:**

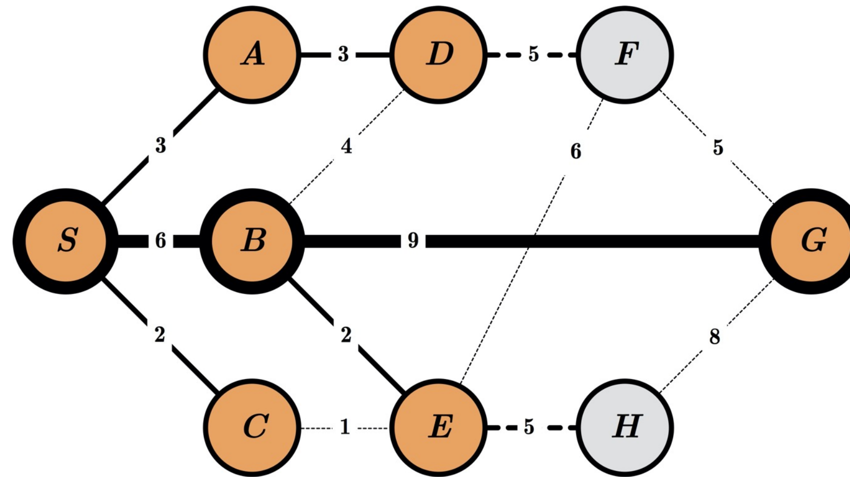
| S | A | B | C | D | E | G | F | H |
|---|---|---|---|---|---|---|---|---|

**Order of Visit:**

*S* *A* *B* *C* *D* *E* *G*

STUDENTS-HUB.com

Uploaded By: anonymous




**Queue:**

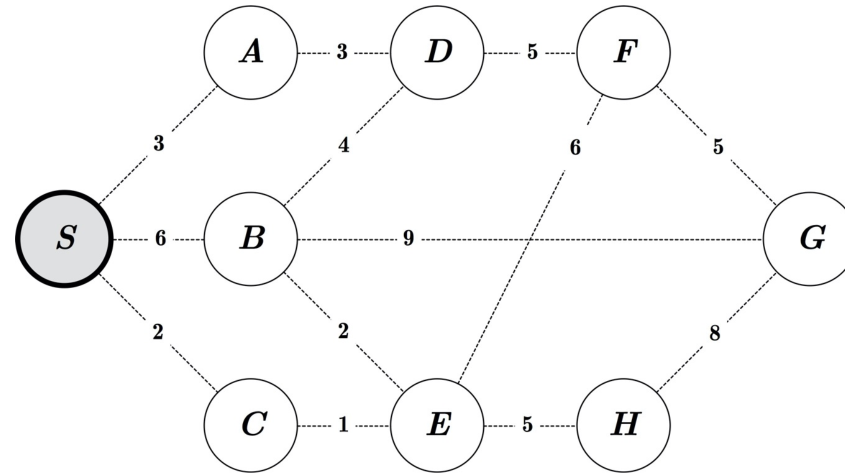
| $S$ | $A$ | $B$ | $C$ | $D$ | $E$ | $G$ | $F$ | $H$ |
|---|---|---|---|---|---|---|---|---|

**Order of Visit:**

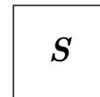
$S$ $A$ $B$ $C$ $D$ $E$ $G$


Uploaded By: anonymous

# Exercise




**Stack:**

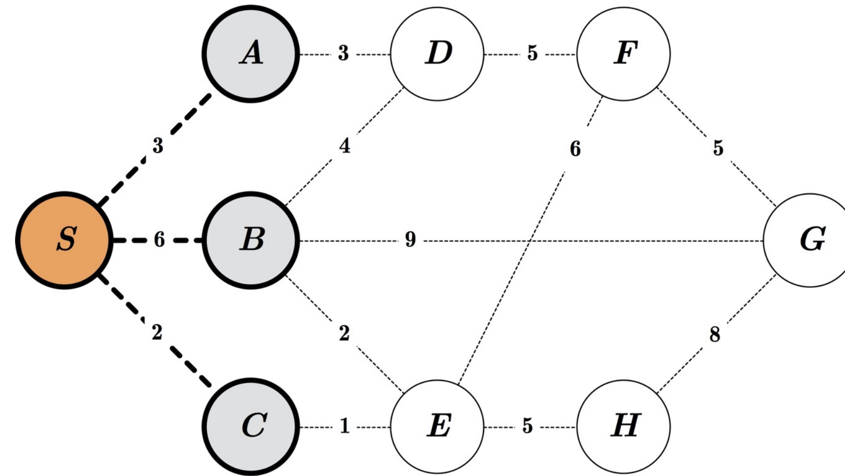



**Order of Visit:**

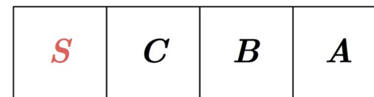
STUDENTS-HUB.com

Uploaded By: anonymous

# Exercise




**Stack:**



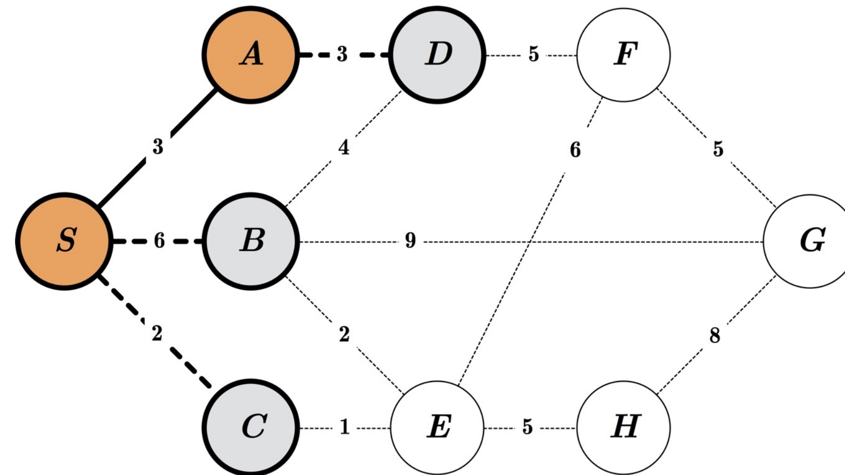

**Order of Visit:**

*S*

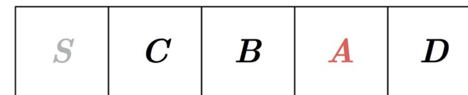
STUDENTS-HUB.com

Uploaded By: anonymous

# Exercise




**Stack:**



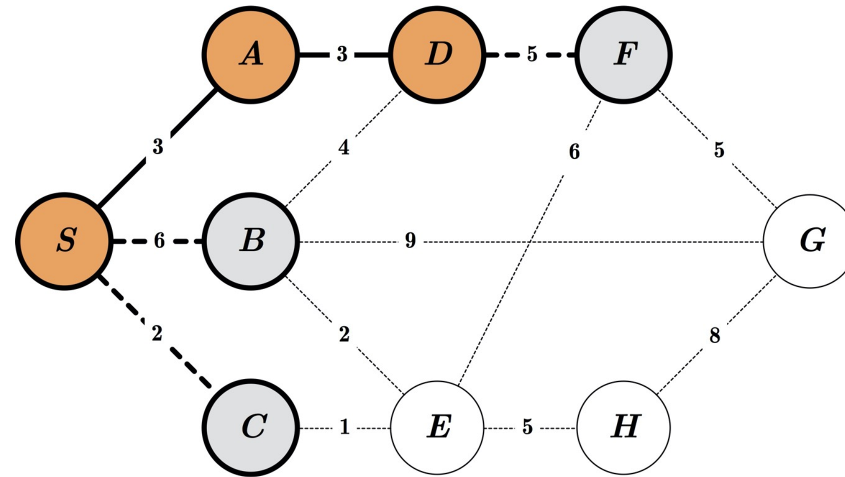

**Order of Visit:**

*S* *A*

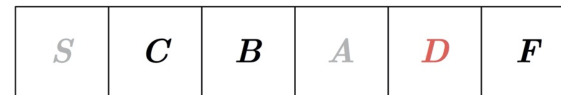
STUDENTS-HUB.com

Uploaded By: anonymous

# Exercise




**Stack:**

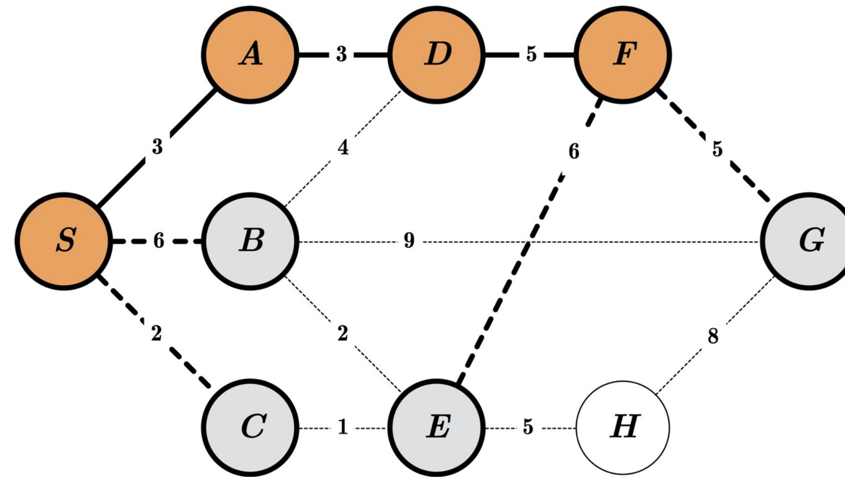



**Order of Visit:**

*S* *A* *D*


Uploaded By: anonymous

# Exercise




**Stack:**

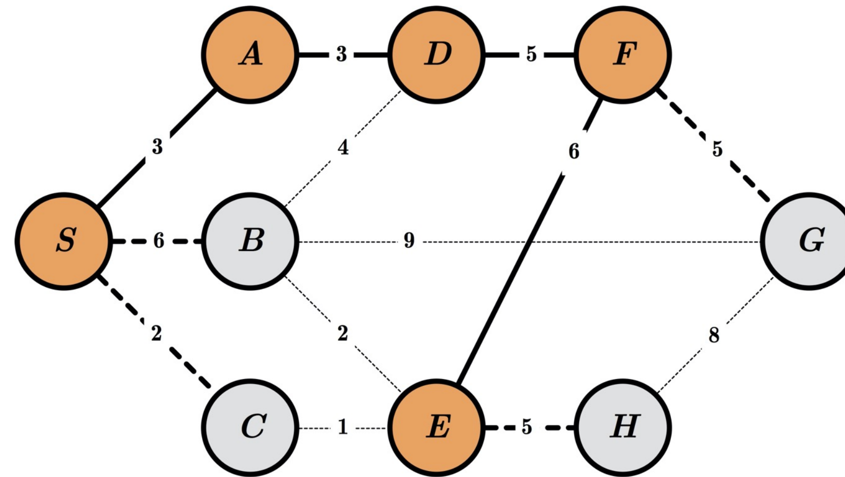
| S | C | B | A | D | F | G | E |
|---|---|---|---|---|---|---|---|

**Order of Visit:**

*S* *A* *D* *F*

STUDENTS-HUB.com
Uploaded By: anonymous

# Exercise




**Stack:**

| *S* | *C* | *B* | *A* | *D* | *F* | *G* | *E* | *H* |
|---|---|---|---|---|---|---|---|---|

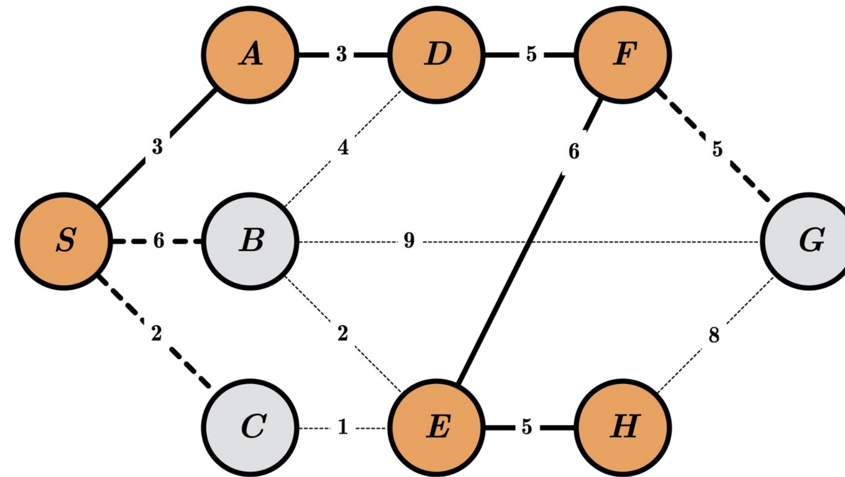
**Order of Visit:**

*S* *A* *D* *F* *E*



Uploaded By: anonymous

# Exercise




**Stack:**

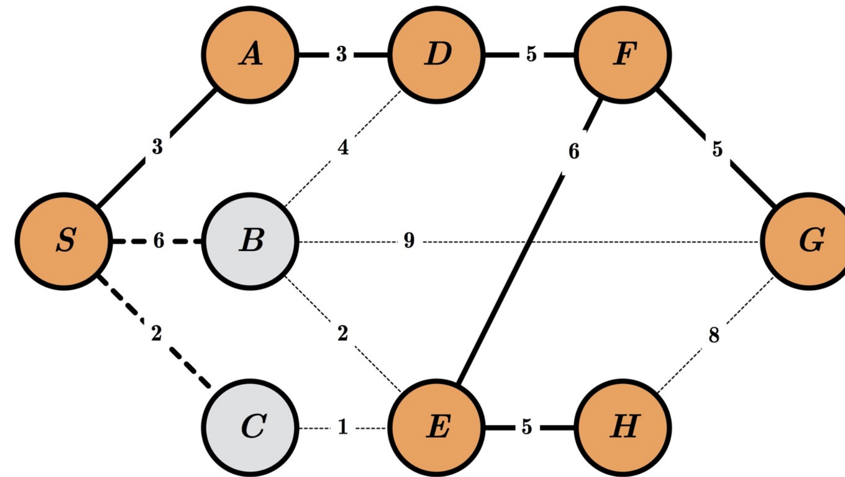
| $S$ | $C$ | $B$ | $A$ | $D$ | $F$ | $G$ | $E$ | $H$ |
|---|---|---|---|---|---|---|---|---|

**Order of Visit:**

$S \quad A \quad D \quad F \quad E \quad H$


Uploaded By: anonymous

# Exercise




**Stack:**

| S | C | B | A | D | F | G | E | H |
|---|---|---|---|---|---|---|---|---|

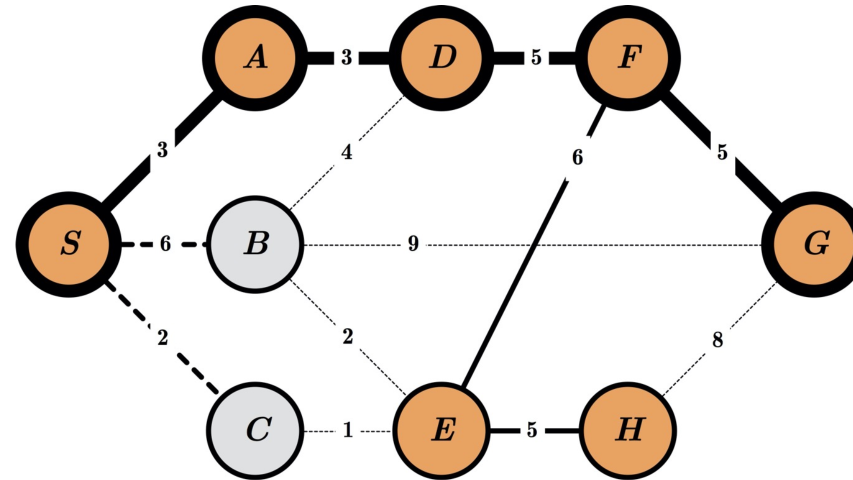
**Order of Visit:**

*S* *A* *D* *F* *E* *H* *G*



Uploaded By: anonymous

# Exercise




**Stack:**

| S | C | B | A | D | F | G | E | H |
|---|---|---|---|---|---|---|---|---|

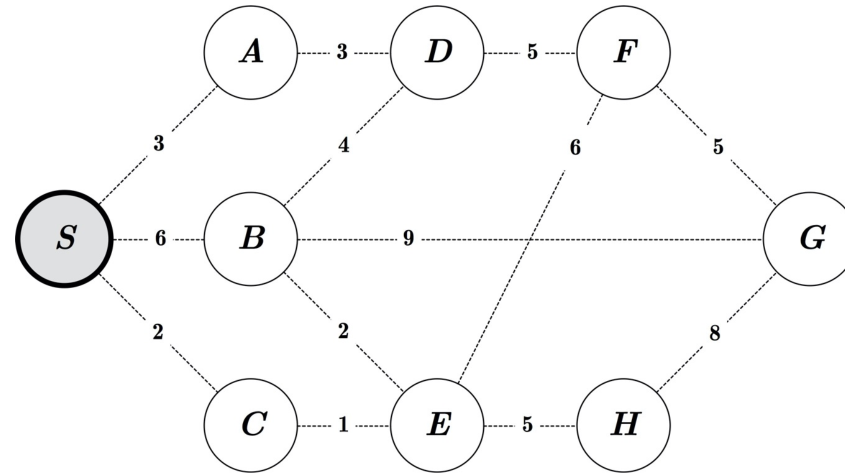
**Order of Visit:**

*S* *A* *D* *F* *E* *H* *G*

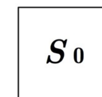
STUDENTS-HUB.com

Uploaded By: anonymous

# Exercise



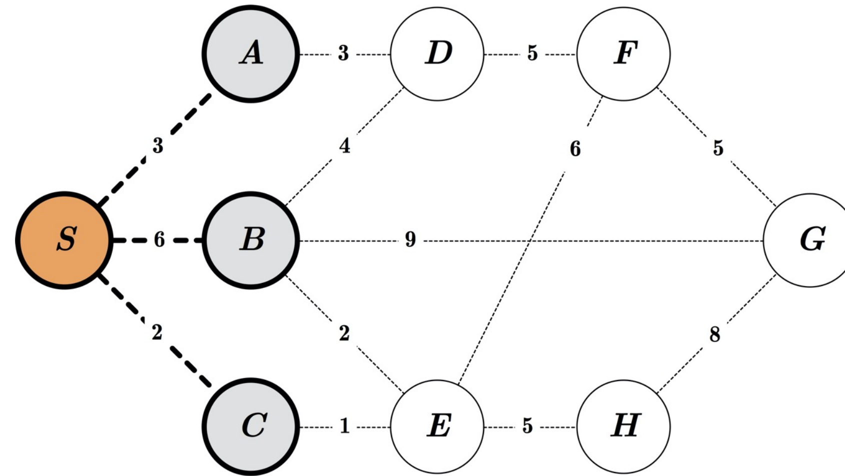

**Priority Queue:**




**Order of Visit:**

STUDENTS-HUB.com
Uploaded By: anonymous

# Exercise




**Priority Queue:**

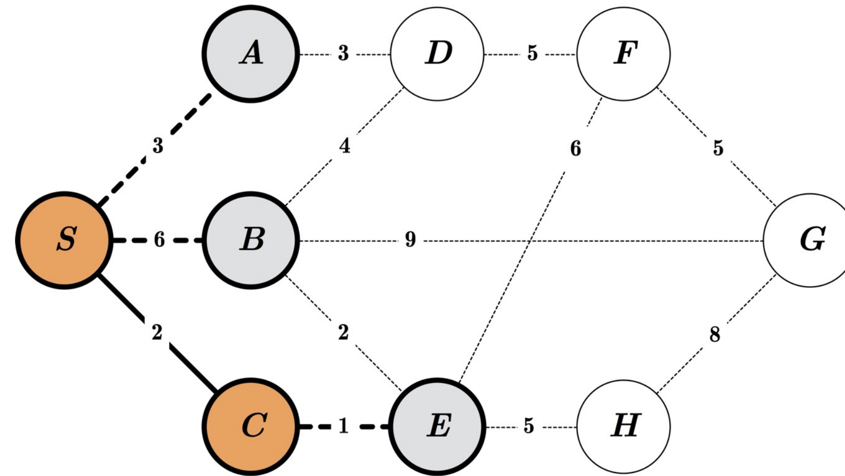
| $S_0$ | $C_2$ | $A_3$ | $B_6$ |
|---|---|---|---|

**Order of Visit:**

$S$


Uploaded By: anonymous

# Exercise




**Priority Queue:**

| $S_0$ | $C_2$ | $A_3$ | $E_3$ | $B_6$ |
|---|---|---|---|---|

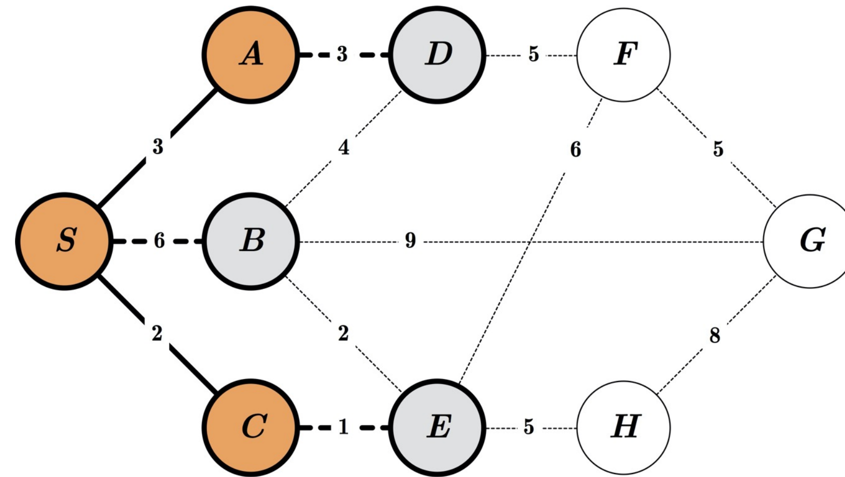
**Order of Visit:**

$S$ $C$



Uploaded By: anonymous

# Exercise




**Priority Queue:**

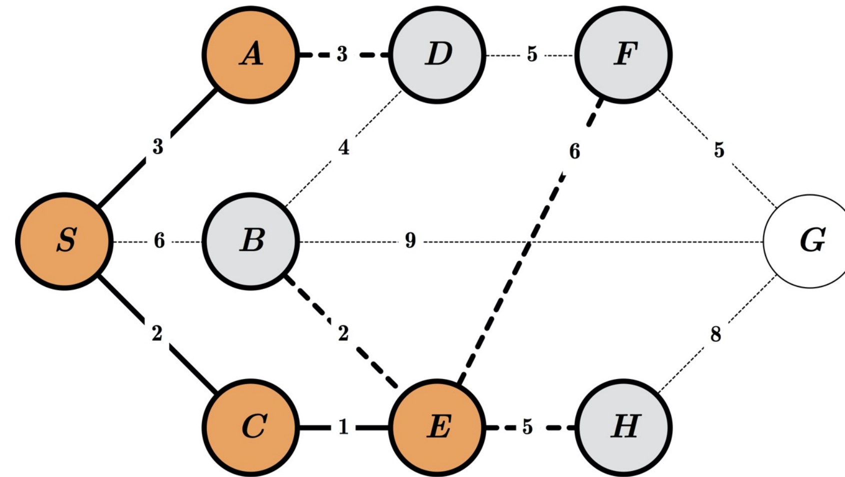
| $S$ 0 | $C$ 2 | $A$ 3 | $E$ 3 | $B$ 6 | $D$ 6 |
|---|---|---|---|---|---|

**Order of Visit:**

$S$ $C$ $A$


Uploaded By: anonymous

# Exercise




**Priority Queue:**

| $S$ 0 | $C$ 2 | $A$ 3 | $E$ 3 | $B$ 5 | $D$ 6 | $H$ 8 | $F$ 9 |
|---|---|---|---|---|---|---|---|

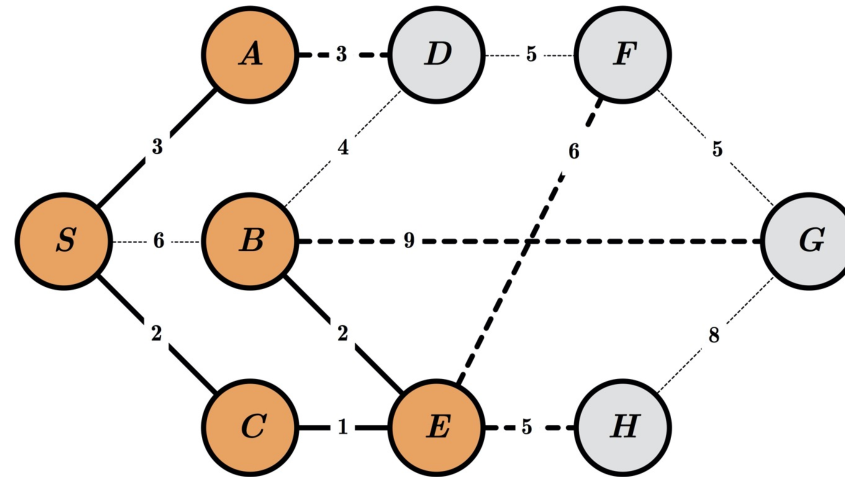
**Order of Visit:**

$S$ $C$ $A$ $E$

STUDENTS-HUB.com

Uploaded By: anonymous

# Exercise




**Priority Queue:**

| $S$ 0 | $C$ 2 | $A$ 3 | $E$ 3 | $B$ 5 | $D$ 6 | $H$ 8 | $F$ 9 | $G$ 14 |
|---|---|---|---|---|---|---|---|---|

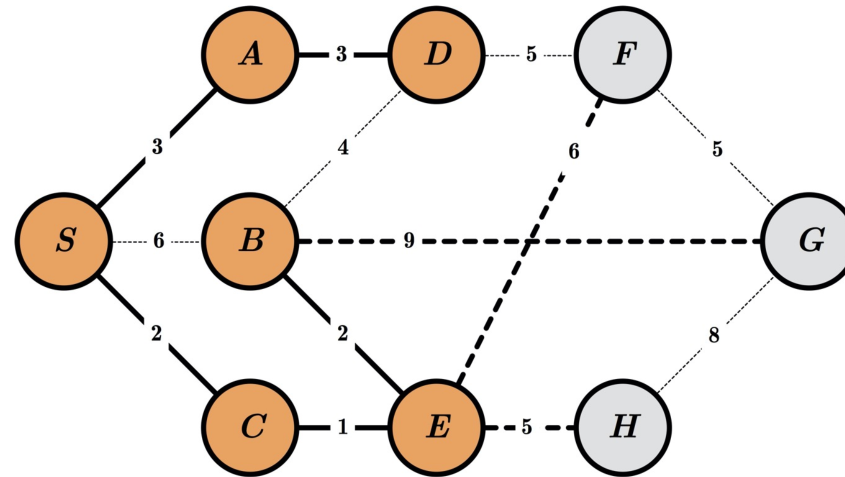
**Order of Visit:**

$S$ $C$ $A$ $E$ $B$

STUDENTS-HUB.com

Uploaded By: anonymous

# Exercise




**Priority Queue:**

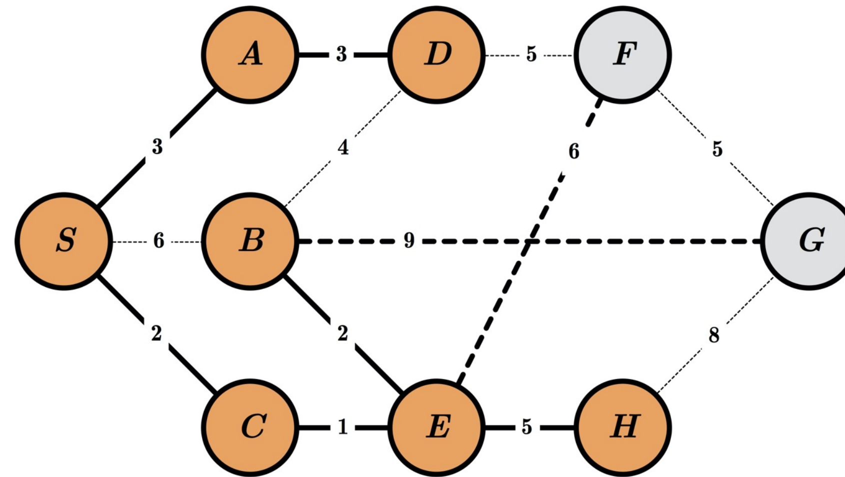
| $S$ 0 | $C$ 2 | $A$ 3 | $E$ 3 | $B$ 5 | $D$ 6 | $H$ 8 | $F$ 9 | $G$ 14 |
|---|---|---|---|---|---|---|---|---|

**Order of Visit:**

$S$ $C$ $A$ $E$ $B$ $D$

STUDENTS-HUB.com Uploaded By: anonymous

# Exercise



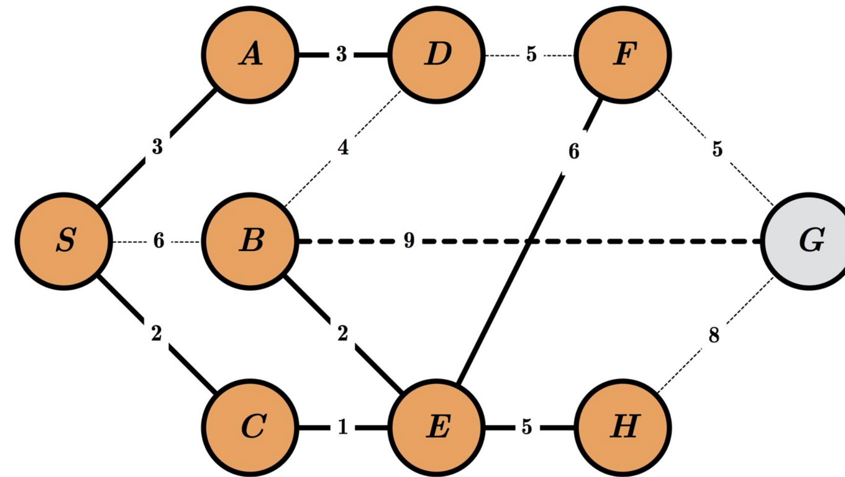

**Priority Queue:**

| $S_0$ | $C_2$ | $A_3$ | $E_3$ | $B_5$ | $D_6$ | $H_8$ | $F_9$ | $G_{14}$ |
|---|---|---|---|---|---|---|---|---|

**Order of Visit:**

*S* *C* *A* *E* *B* *D* *H*

# Exercise




**Priority Queue:**

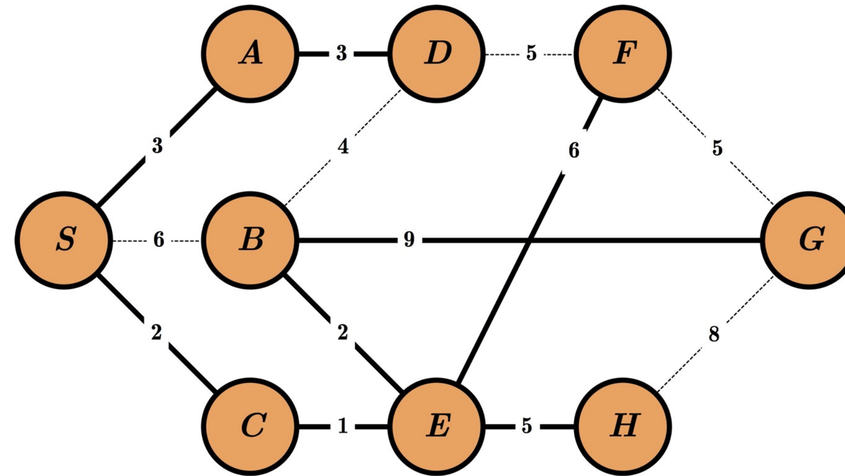
| $S$ 0 | $C$ 2 | $A$ 3 | $E$ 3 | $B$ 5 | $D$ 6 | $H$ 8 | $F$ 9 | $G$ 14 |
|---|---|---|---|---|---|---|---|---|

**Order of Visit:**

$S$ $C$ $A$ $E$ $B$ $D$ $H$ $F$

STUDENTS-HUB.com
Uploaded By: anonymous

# Exercise




**Priority Queue:**

| $S$ 0 | $C$ 2 | $A$ 3 | $E$ 3 | $B$ 5 | $D$ 6 | $H$ 8 | $F$ 9 | $G$ 14 |
|---|---|---|---|---|---|---|---|---|

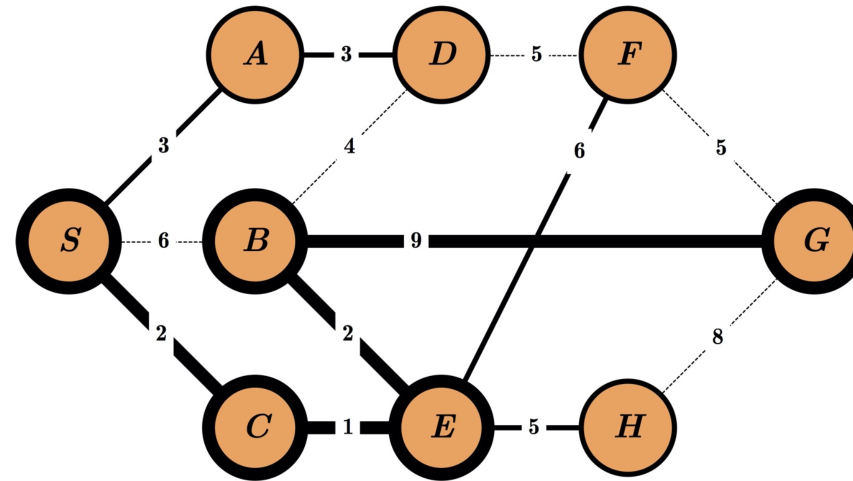
**Order of Visit:**

$S$ $C$ $A$ $E$ $B$ $D$ $H$ $F$ $G$

STUDENTS-HUB.com

Uploaded By: anonymous

# Exercise




**Priority Queue:**

| $S$ 0 | $C$ 2 | $A$ 3 | $E$ 3 | $B$ 5 | $D$ 6 | $H$ 8 | $F$ 9 | $G$ 14 |
|---|---|---|---|---|---|---|---|---|

**Order of Visit:**

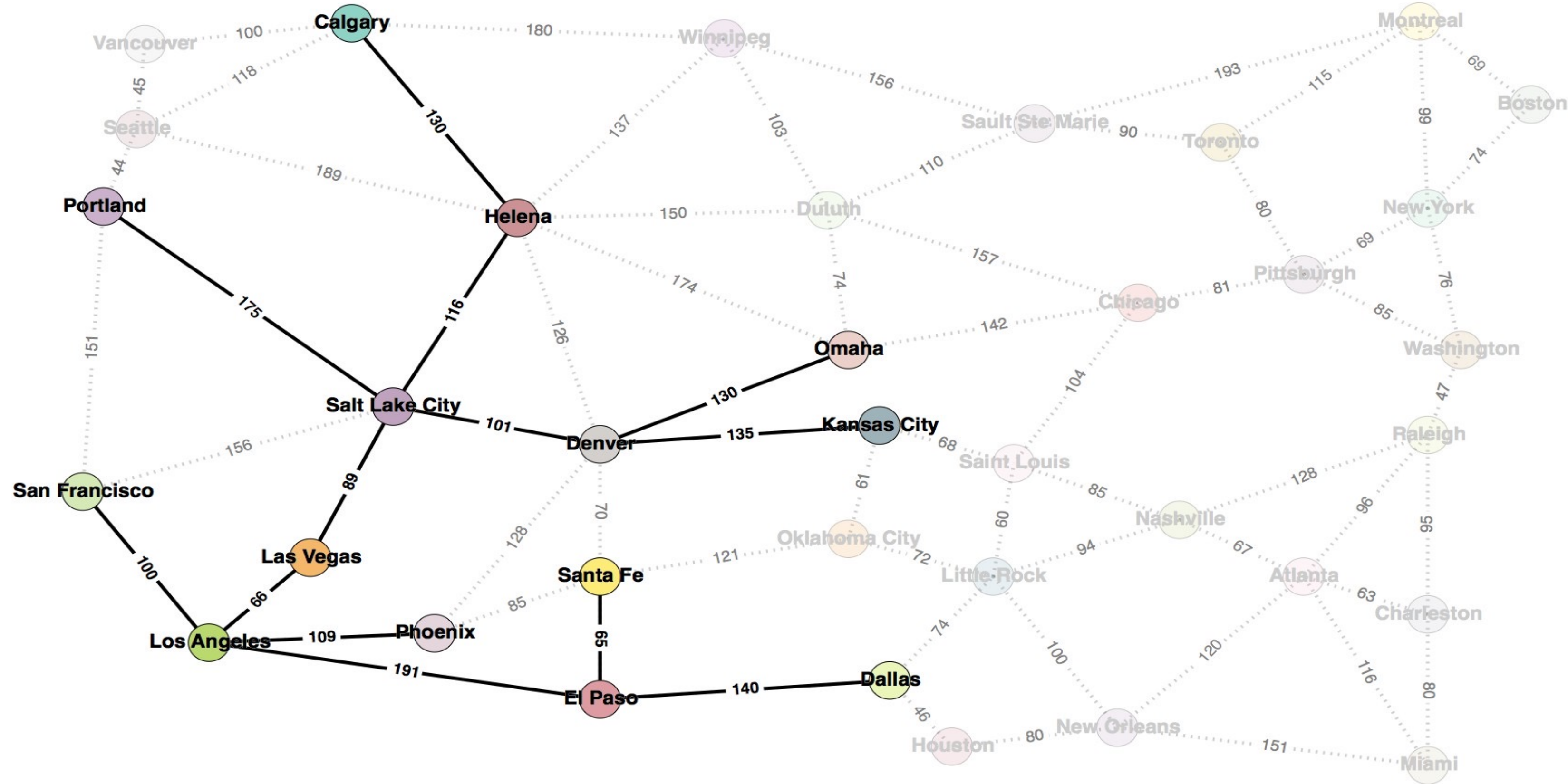
$S$ $C$ $A$ $E$ $B$ $D$ $H$ $F$ $G$

STUDENTS-HUB.com

Uploaded By: anonymous

# Examples using the map

**Start**: Las Vegas **Goal:** Calgary



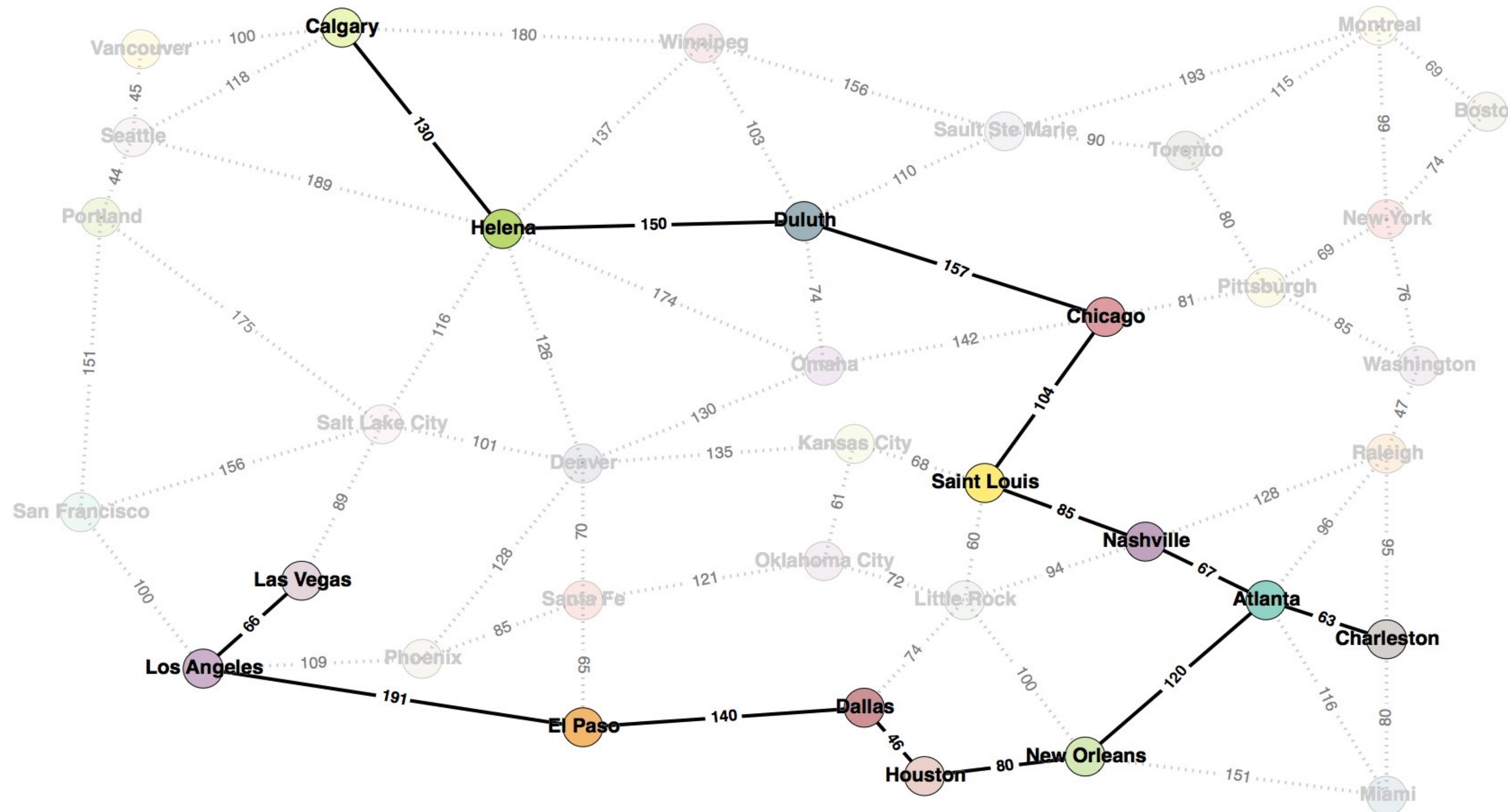

BFS

Order of Visit: Las Vegas, Los Angeles, Salt Lake City, El Paso, Phoenix, San Francisco, Denver, Helena, Portland, Dallas, Santa Fe, Kansas City, Omaha, Calgary.

STUDENTS-HUB.com
Uploaded By: anonymous

# Examples using the map

**Start:** Las Vegas **Goal:** Calgary




DFS

Order of Visit: Las Vegas, Los Angeles, El Paso, Dallas, Houston, New Orleans, Atlanta, Charleston, Nashville, Saint Louis, Chicago, Duluth, Helena, Calgary.

STUDENTS-HUB.com

Uploaded By: anonymous

# Examples using the map

**Start:** **Las Vegas** **Goal:** **Calgary**

UCS

Order of Visit: Las Vegas, Los Angeles, Salt Lake City, San Francisco, Phoenix, Denver, Helena, El Paso, Santa Fe, Portland, Seattle, Omaha, Kansas City, Calgary.

STUDENTS-HUB.com

Uploaded By: anonymous

# Credit and References


- Artificial Intelligence, A Modern Approach. Stuart Russell and Peter Norvig. Third Edition. Pearson Education: http://aima.cs.berkeley.edu/
- Mustafa Jarrar: Lecture notes of COMP338: Artificial Intelligence, Birzeit University: http://www.jarrar.info/courses/AI/
- Franz Kurfess: Notes on Artificial Intelligence: http://users.csc.calpoly.edu/~fkurfess/Courses/480/F03/Slides/3-Search.pdf
- David Lee Matuszek: Lecture Notes on Tree Searches: http://www.cs.nyu.edu/courses/fall06/V22.0102-001/lectures/treeSearching.ppt



STUDENTS-HUB.com

Uploaded By: anonymous