Uninformed Search

Wenhu Chen

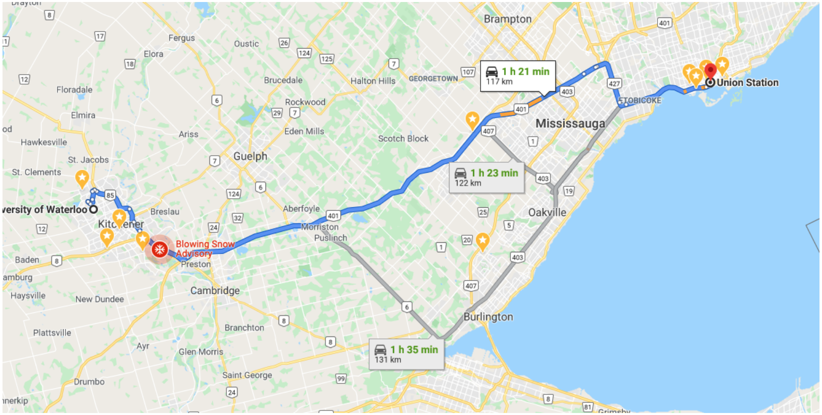
Lecture 17

Outline

Learning goals

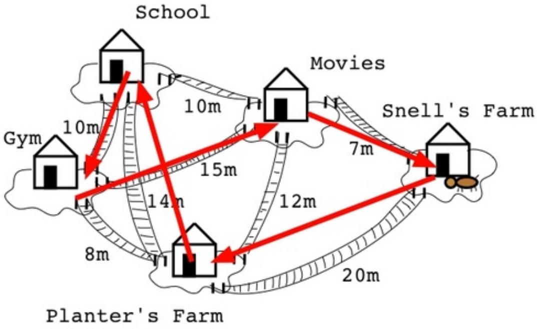
- Formulate a real-world problem as a search problem.
- Trace the execution of and implement uninformed search algorithms (Breadth-first search, Depth-first search, Iterative-deepening search).
- Given an uninformed search algorithm, explain its space complexity, time complexity, and whether it has any guarantees on the quality of the solution found.
- Given a scenario, explain whether and why it is appropriate to use an uninformed algorithm.

Example: Route Planning



Traveling Salesperson Problem

What is the shortest path that starts at city A, visits each city only once, and returns to A?



Applications: DNA sequencing. School bus routes. Parcel pickups.

See this article for a recent breakthrough.

8-puzzle

Initial State

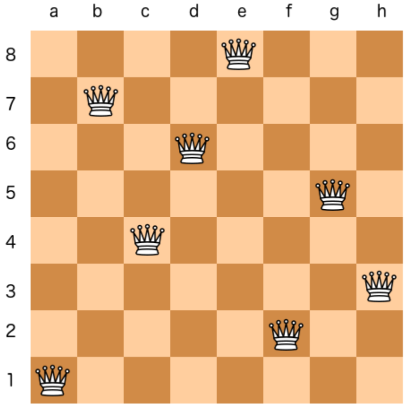
5	3	
8	7	6
2	4	1

Goal State

1	2	3
4	5	6
7	8	

N-Queens Problem

The n-queens problem: Place n queens on an $n \times n$ board so that no pair of queens attacks each other.

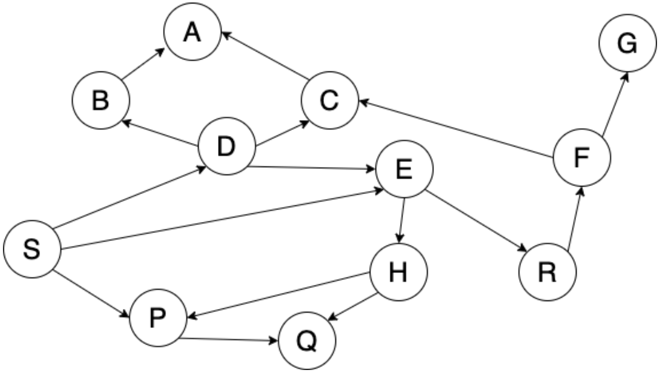


http://yue-guo.com/wp-content/uploads/2019/02/N_queen.png

Motivation for Search Algorithms

- Problems that involve sequential decision making
- Last resort: No efficient Algorithm (e.g. solve a linear equation)
- Easy to verify a solution, but to find a solution
- NP-hard problem
- Solve complex problems in Real-Life, trial-and-error
- On a computer, a search algorithm will explore all paths systematically

Graph Searching



Any search problem can be visualized as a graph, where each node represents a state.

- S is the initial state, G is the goal state
- Search == Traversing the graph to find a path from S to G

A Search Problem

Definition (Search Problem)

A **search problem** is defined by:

- A set of **states**
- An **initial state**
- **Goal states** or **a goal test**
  - a boolean function which tells us whether a given state is a goal state
- A **successor (neighbour) function**
  - an action which takes us from one state to other states
- (Optionally) a **cost** associated with each action

A solution to this problem is a path from the start state
to a goal state (optionally with the smallest total cost).

Example: 8-Puzzle

Initial State

5	3	
8	7	6
2	4	1

Goal State

1	2	3
4	5	6
7	8	

Formulating 8-Puzzle as a Search Problem

- State:
- Initial state:
- Goal states:
- Successor function:
- Cost function:

Formulating 8-Puzzle as a Search Problem

- State: $x_{00}x_{01}x_{02}, x_{10}x_{11}x_{12}, x_{20}x_{21}x_{22}$
  x_{ij} is the number in row i and column j. $i, j \in \{0, 1, 2\}$.
  $x_{ij} \in \{0, \ldots, 8\}$. $x_{ij} = 0$ denotes the empty square.

Formulating 8-Puzzle as a Search Problem

- State: $x_{00}x_{01}x_{02}, x_{10}x_{11}x_{12}, x_{20}x_{21}x_{22}$
  x_{ij} is the number in row i and column j. $i, j \in \{0, 1, 2\}$.
  $x_{ij} \in \{0, \ldots, 8\}$. $x_{ij} = 0$ denotes the empty square.

- Initial state: $530, 876, 241$.

Formulating 8-Puzzle as a Search Problem

- State: $x_{00}x_{01}x_{02}, x_{10}x_{11}x_{12}, x_{20}x_{21}x_{22}$
  x_{ij} is the number in row i and column j. $i, j \in \{0, 1, 2\}$.
  $x_{ij} \in \{0, \ldots, 8\}$. $x_{ij} = 0$ denotes the empty square.
- Initial state: $530, 876, 241$.
- Goal states: $123, 456, 780$.

Formulating 8-Puzzle as a Search Problem

- State: $x_{00}x_{01}x_{02}, x_{10}x_{11}x_{12}, x_{20}x_{21}x_{22}$
  x_{ij} is the number in row i and column j. $i, j \in \{0, 1, 2\}$.
  $x_{ij} \in \{0, \ldots, 8\}$. $x_{ij} = 0$ denotes the empty square.

- Initial state: $530, 876, 241$.

- Goal states: $123, 456, 780$.

- Successor function: Consider the empty square as a tile. State B is a successor of state A if and only if we can convert A to B by moving the empty tile up, down, left, or right by one step.

Formulating 8-Puzzle as a Search Problem

- State: $x_{00}x_{01}x_{02}, x_{10}x_{11}x_{12}, x_{20}x_{21}x_{22}$
  x_{ij} is the number in row i and column j. $i, j \in \{0, 1, 2\}$.
  $x_{ij} \in \{0, \ldots, 8\}$. $x_{ij} = 0$ denotes the empty square.

- Initial state: $530, 876, 241$.

- Goal states: $123, 456, 780$.

- Successor function: Consider the empty square as a tile. State B is a successor of state A if and only if we can convert A to B by moving the empty tile up, down, left, or right by one step.

- Cost function: Each move has a cost of 1.

Q: The successor function

Q: Which of the following is a successor of $530, 876, 241$?

(A) $350, 876, 241$
(B) $536, 870, 241$
(C) $537, 806, 241$
(D) $538, 076, 241$

Initial State

5	3	
8	7	6
2	4	1

Q: The successor function

Q: Which of the following is a successor of $530, 876, 241$?

Initial State

5	3	
8	7	6
2	4	1

(A) $350, 876, 241$
(B) $536, 870, 241$
(C) $537, 806, 241$
(D) $538, 076, 241$

$\rightarrow$ (B) is correct.

Choosing among multiple formulations

Choosing among multiple formulations

- The state definition determines the nodes.
  The successor function determines the directed edges.

Choosing among multiple formulations

- The state definition determines the nodes.
  The successor function determines the directed edges.

- Ideally, we want to minimize the number of nodes and edges in the graph.

Choosing among multiple formulations

- The state definition determines the nodes.
  The successor function determines the directed edges.

- Ideally, we want to minimize the number of nodes and edges in the graph.

- Choosing a state definition may make it easier or harder to implement the successor function.

  An alternative state definition for the 8-puzzle:
  A state is defined by 8 coordinates.
  (x_i, y_i) is the coordinates for tile i where $1 \leq i \leq 8$.

Choosing among multiple formulations

Initial State

5	3	
8	7	6
2	4	1

- State: (0, 2), (2, 2), (2, 0), (0, 1), (2, 1), (0, 0), (1, 2), (1, 1), (2, 0)

Choosing among multiple formulations

Initial State

5	3	
8	7	6
2	4	1

- State: (0, 2), (2, 2), (2, 0), (0, 1), (2, 1), (0, 0), (1, 2), (1, 1), (2, 0)
- Space Consumption will increase by 2x.

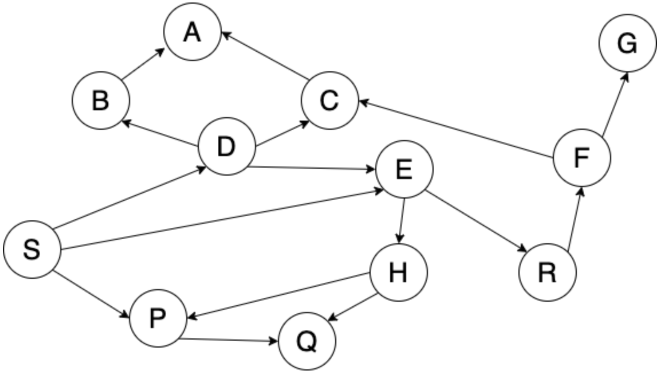
Choosing among multiple formulations

Initial State

5	3	
8	7	6
2	4	1

- State: (0, 2), (2, 2), (2, 0), (0, 1), (2, 1), (0, 0), (1, 2), (1, 1), (2, 0)
- Space Consumption will increase by 2x.
- Successor Function: Swap the 0 tile coordinate with another neighbor tile

The Search Graph



- normally no duplicate of nodes
- label each edge with its cost
- this lecture will assume each edge has an equal cost of 1

Searching for a Solution

$\rightarrow$ A formulation gives us enough info to generate the search graph. However, we often don't generate the search graph explicitly and store it. It may be large or infinite and it is not necessary.

Instead, we will generate a search tree as we explore the search graph.

- Construct the search tree as we explore paths incrementally from the start node.

Searching for a Solution

$\rightarrow$ A formulation gives us enough info to generate the search graph. However, we often don't generate the search graph explicitly and store it. It may be large or infinite and it is not necessary.

Instead, we will generate a search tree as we explore the search graph.

- Construct the search tree as we explore paths incrementally from the start node.
- Maintain a frontier of paths from the start node.

Searching for a Solution

$\rightarrow$ A formulation gives us enough info to generate the search graph. However, we often don't generate the search graph explicitly and store it. It may be large or infinite and it is not necessary.

Instead, we will generate a search tree as we explore the search graph.

- Construct the search tree as we explore paths incrementally from the start node.
- Maintain a frontier of paths from the start node.
- Frontier contains all the paths available for expansion.

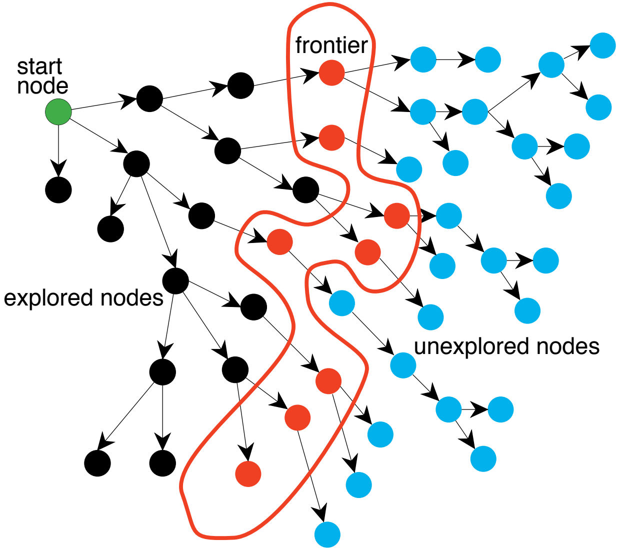
Searching for a Solution

$\rightarrow$ A formulation gives us enough info to generate the search graph. However, we often don't generate the search graph explicitly and store it. It may be large or infinite and it is not necessary.

Instead, we will generate a search tree as we explore the search graph.

- Construct the search tree as we explore paths incrementally from the start node.
- Maintain a frontier of paths from the start node.
- Frontier contains all the paths available for expansion.
- Expanding a path: removing it from the frontier, generating all the neighbors of the last node, and adding the paths ending with each neighbor to the frontier.

The Search Tree



Generic Search Algorithm

Algorithm 1 A Generic Search Algorithm

```
1: procedure SEARCH(Graph, Start node $s$, Goal test $goal(n)$)
```

Generic Search Algorithm

Algorithm 2 A Generic Search Algorithm

```
1: procedure SEARCH(Graph, Start node $s$, Goal test $goal(n)$)
2:     frontier := $\{\langle s \rangle\}$
```

Generic Search Algorithm

Algorithm 3 A Generic Search Algorithm

```
1: procedure SEARCH(Graph, Start node $s$, Goal test $goal(n)$)
2:     frontier := $\{\langle s \rangle\}$
3:     while frontier is not empty do
```

Generic Search Algorithm

Algorithm 4 A Generic Search Algorithm

```
1: procedure SEARCH(Graph, Start node $s$, Goal test $goal(n)$)
2:     frontier := $\{\langle s \rangle\}$
3:     while frontier is not empty do
4:         select and remove path $\langle n_0, \ldots, n_k \rangle$ from frontier
```

Generic Search Algorithm

Algorithm 5 A Generic Search Algorithm

```
1: procedure SEARCH(Graph, Start node $s$, Goal test $goal(n)$)
2:     frontier := $\{\langle s \rangle\}$
3:     while frontier is not empty do
4:         select and remove path $\langle n_0, \ldots, n_k \rangle$ from frontier
5:         if goal($n_k$) then
6:             return $\langle n_0, \ldots, n_k \rangle$
```

Generic Search Algorithm

Algorithm 6 A Generic Search Algorithm

```
1: procedure SEARCH(Graph, Start node $s$, Goal test $goal(n)$)
2:     frontier := $\{\langle s \rangle\}$
3:     while frontier is not empty do
4:         select and remove path $\langle n_0, \ldots, n_k \rangle$ from frontier
5:         if goal($n_k$) then
6:             return $\langle n_0, \ldots, n_k \rangle$
7:         for every neighbour $n$ of $n_k$ do
8:             add $\langle n_0, \ldots, n_k, n \rangle$ to frontier
```

Generic Search Algorithm

Algorithm 7 A Generic Search Algorithm

```
1: procedure SEARCH(Graph, Start node $s$, Goal test $goal(n)$)
2:     frontier := $\{\langle s \rangle\}$
3:     while frontier is not empty do
4:         select and remove path $\langle n_0, \dots, n_k \rangle$ from frontier
5:         if goal($n_k$) then
6:             return $\langle n_0, \dots, n_k \rangle$
7:         for every neighbour $n$ of $n_k$ do
8:             add $\langle n_0, \dots, n_k, n \rangle$ to frontier
9:     return no solution
```

Depth-First Search

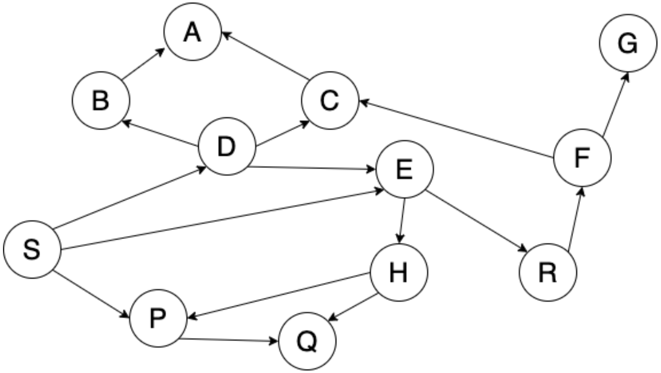
- Treats the frontier as a stack (LIFO).
- Expands the last/most recent node added to the frontier.

$\rightarrow$ Search one path to completion before trying another path.

Backtracks to another alternative after it has explored all the paths from the first alternative.

Trace DFS on a Search Graph

- Trace DFS on the search graph below, with S as the initial state.
- Add nodes to the frontier in alphabetical order.

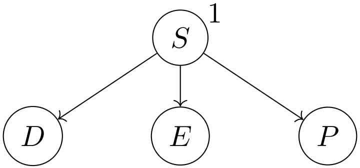


Trace DFS on a Search Graph

Frontier: (S):

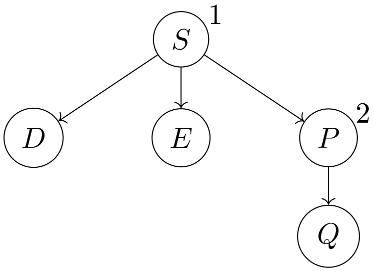
Trace DFS on a Search Graph

Frontier: (S) $\rightarrow$ (D, E, P):



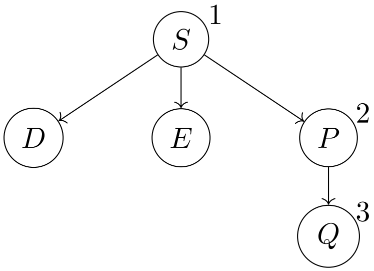
Trace DFS on a Search Graph

Frontier: (D, E, P) $\rightarrow$ (D, E, Q):



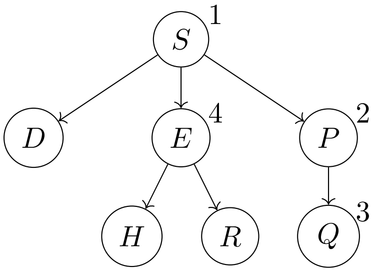
Trace DFS on a Search Graph

Frontier: (D, E, Q) $\rightarrow$ (D, E):



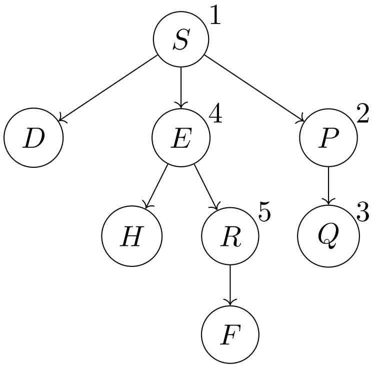
Trace DFS on a Search Graph

Frontier: (D, E) $\rightarrow$ (D, H, R):



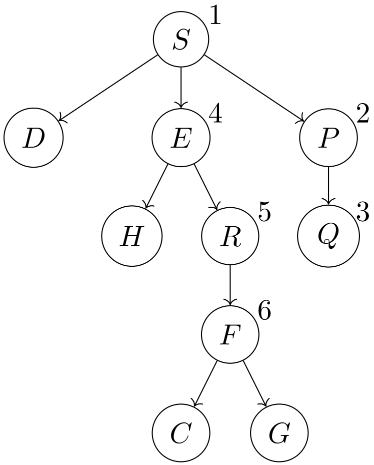
Trace DFS on a Search Graph

Frontier: (D, H, R) $\rightarrow$ (D, H, F):



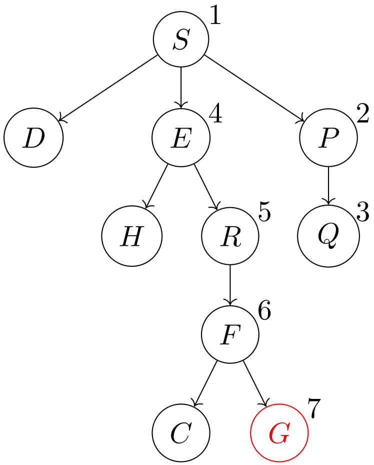
Trace DFS on a Search Graph

Frontier: (D, H, F) → (D, H, C, G):



Trace DFS on a Search Graph

Frontier: (D, H, C, G) $\rightarrow$ (D, H, C):



It takes 7 steps to reach the goal state.

Properties of DFS

Properties

- *Space Complexity:* size of frontier in worst case
- *Time Complexity:* # nodes visited in worst case
- *Completeness:* does it find a solution when one exists?
- *Optimality:* if solution found, is it the one with the least cost?

Useful Quantities

- b is the branching factor.
- m is the maximum depth of the search tree.
- d is the depth of the shallowest goal node.

Properties of DFS - Space Complexity

Space Complexity

Properties of DFS - Space Complexity

Space Complexity

- $O(bm)$

  b is the branching factor.
  m is the max depth of the search tree.

- Linear in m

- Remembers m nodes on current path and at most b siblings for each node.

Properties of DFS - Time Complexity

Time Complexity

Properties of DFS - Time Complexity

Time Complexity

- $O(b^m)$
- Exponential in m.
- Visit the entire search tree in the worst case.

Properties of DFS - Completeness

Is DFS guaranteed to find a solution if a solution exists?

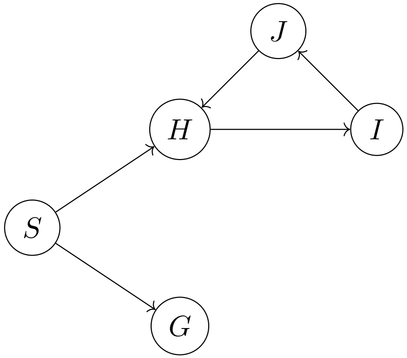
Properties of DFS - Completeness

Is DFS guaranteed to find a solution if a solution exists?

- No.
- Will get stuck in an infinite path.
- An infinite path may or may not be a cycle.

Properties of DFS - Completeness

Is DFS guaranteed to find a solution if a solution exists?



- Solution G exists
- Loop: $H \to I \to J \to H$, cannot find the solution

Properties of DFS - Optimality

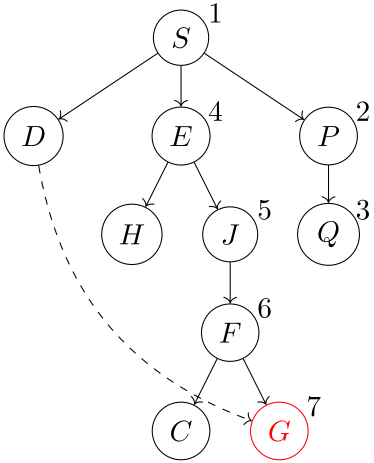
Is DFS guaranteed to return an optimal solution if it terminates?

Properties of DFS - Optimality

Is DFS guaranteed to return an optimal solution if it terminates?

- No.
- Pays no attention to the costs and
  makes no guarantee on the solution's quality.

Properties of DFS - Optimality



When should we use DFS?

DFS is useful when:

- Space is restricted.
- Many solutions exist, perhaps with long paths.

DFS is a poor method when:

- There are infinite paths.
- Solutions are shallow.
- There are multiple paths to a node.

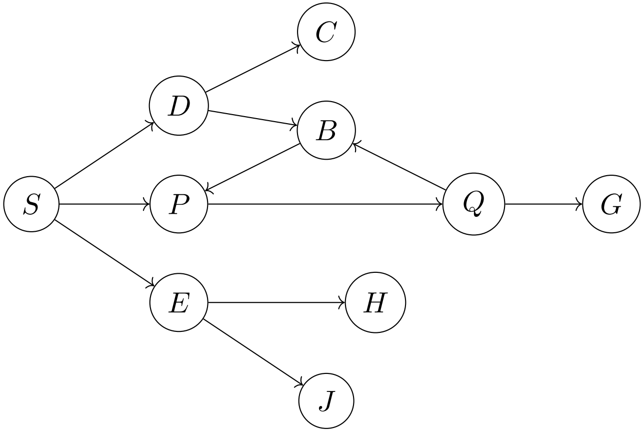
Breadth-First Search

- Treats the frontier as a queue (FIFO).
- Expands the first/oldest node added to the frontier.

$\rightarrow$ Select a path with the fewest arcs at each step.

Trace BFS on a Search Graph

- Trace the BFS algorithm

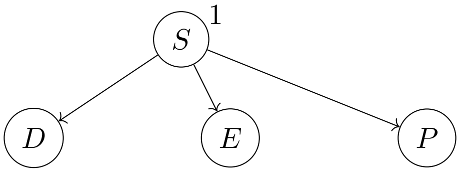


Trace BFS on a Search Graph

Frontier: (S)

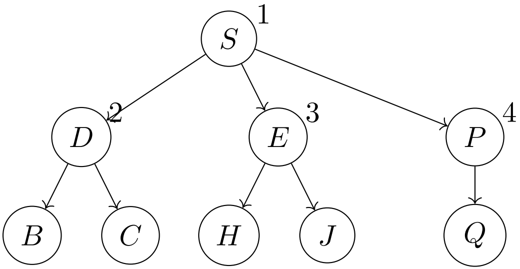
Trace BFS on a Search Graph

Frontier: (S) $\rightarrow$ (D, E, P)



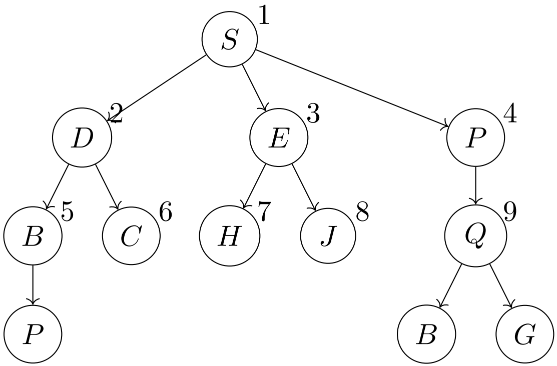
Trace BFS on a Search Graph

Frontier: (D, E, P) $\rightarrow$ (B, C, H, J, Q)



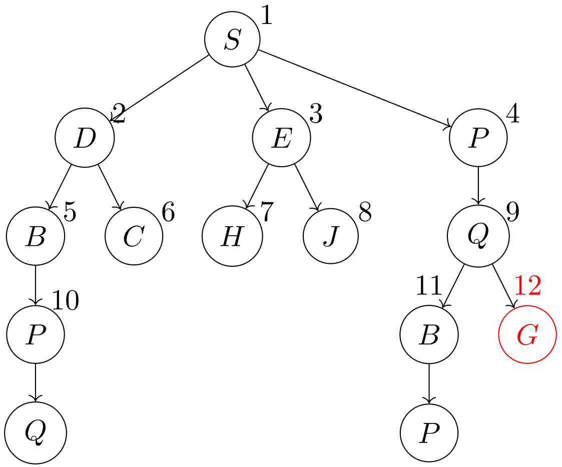
Trace BFS on a Search Graph

Frontier: (B, C, H, J, Q) $\rightarrow$ (P, B, G)



Trace BFS on a Search Graph

Frontier: (P, B, G) $\rightarrow$ (G, Q, P)



It takes 12 steps to reach the goal state. More time than DFS!

Properties of BFS - Space Complexity

Space Complexity

Properties of BFS - Space Complexity

Space Complexity

- $O(b^d)$

  b is the branching factor.
  d is the depth of the shallowest goal node.

- Exponential in d.

- Must visit the top d levels.
  The size of the frontier is dominated by the size of level d.

Properties of BFS - Time Complexity

Time Complexity

Properties of BFS - Time Complexity

Time Complexity

- $O(b^d)$
- Exponential in d.
- Visit the entire search tree in the worst case.

Properties of BFS - Completeness

Is BFS guaranteed to find a solution if a solution exists?

Properties of BFS - Completeness

Is BFS guaranteed to find a solution if a solution exists?

- Yes.
- Explores the tree level by level until it finds a goal.

Properties of BFS - Optimality

Is BFS guaranteed to return an optimal solution if it terminates?

Properties of BFS - Optimality

Is BFS guaranteed to return an optimal solution if it terminates?

- Yes and No.
- Guaranteed to find the shallowest goal node. It is the lowest-cost solution if all edge has the same cost.

When should we use BFS?

BFS is useful when:

- Space is not a concern.
- Want a solution with the fewest arcs.

BFS is a poor method when:

- All the solutions are deep in the tree.
- The problem is large and the graph is dynamically generated.

Q: BFS v.s. DFS

Q #1: Suppose that memory is very limited.
Which of BFS and DFS would you choose?

(A) BFS is a better choice.

(B) DFS is a better choice.

(C) Both are good choices.

(D) Neither is a good choice.

Q: BFS v.s. DFS

Q #1: Suppose that memory is very limited.
Which of BFS and DFS would you choose?

(A) BFS is a better choice.

(B) **DFS is a better choice.**

(C) Both are good choices.

(D) Neither is a good choice.

$\rightarrow$ (B) DFS is better.
BFS requires exponential memory.
DFS requires linear memory.

Q: BFS v.s. DFS

Q #2: Suppose that all the solutions are deep in the search tree. Which of BFS and DFS would you choose?

(A) BFS is a better choice.

(B) DFS is a better choice.

(C) Both are good choices.

(D) Neither is a good choice.

Q: BFS v.s. DFS

Q #2: Suppose that all the solutions are deep in the search tree. Which of BFS and DFS would you choose?

(A) BFS is a better choice.

(B) **DFS is a better choice.**

(C) Both are good choices.

(D) Neither is a good choice.

$\rightarrow$ (B) DFS is better.
DFS will explore long paths first and find a solution faster.
BFS will explore shallow nodes first and will likely require more time to find a solution.

Q: BFS v.s. DFS

Q #3: Suppose that the search graph contains cycles. Which of BFS and DFS would you choose?

(A) BFS is a better choice.

(B) DFS is a better choice.

(C) Both are good choices.

(D) Neither is a good choice.

Q: BFS v.s. DFS

Q #3: Suppose that the search graph contains cycles. Which of BFS and DFS would you choose?

(A) **BFS is a better choice.**

(B) DFS is a better choice.

(C) Both are good choices.

(D) Neither is a good choice.

$\rightarrow$ (A) BFS is better.
DFS may get stuck in cycles.
BFS has no problem with cycles.

Q: BFS v.s. DFS

Q #4: Suppose that the branching factor is large/infinite. Which of BFS and DFS would you choose?

(A) BFS is a better choice.

(B) DFS is a better choice.

(C) Both are good choices.

(D) Neither is a good choice.

Q: BFS v.s. DFS

Q #4: Suppose that the branching factor is large/infinite. Which of BFS and DFS would you choose?

(A) BFS is a better choice.

(B) **DFS is a better choice.**

(C) Both are good choices.

(D) Neither is a good choice.

$\rightarrow$ (B) DFS is better.
BFS will be slow for large b and will not terminate with infinite b.
DFS requires linear memory.

Q: BFS v.s. DFS

Q #5: Suppose that we must find the shallowest goal node. Which of BFS and DFS would you choose?

(A) BFS is a better choice.

(B) DFS is a better choice.

(C) Both are good choices.

(D) Neither is a good choice.

Q: BFS v.s. DFS

Q #5: Suppose that we must find the shallowest goal node. Which of BFS and DFS would you choose?

(A) **BFS is a better choice.**

(B) DFS is a better choice.

(C) Both are good choices.

(D) Neither is a good choice.

$\rightarrow$ (A) BFS is better.
BFS is guaranteed to find the shallowest goal node.
DFS will likely find a deep goal node first.

Q: BFS v.s. DFS

Q #6: Suppose that all the solutions are very shallow. Which of BFS and DFS would you choose?

(A) BFS is a better choice.

(B) DFS is a better choice.

(C) Both are good choices.

(D) Neither is a good choice.

Q: BFS v.s. DFS

Q #6: Suppose that all the solutions are very shallow.
Which of BFS and DFS would you choose?

(A) **BFS is a better choice.**

(B) DFS is a better choice.

(C) Both are good choices.

(D) Neither is a good choice.

$\rightarrow$ (A) BFS is better.
BFS is guaranteed to find the shallowest goal node.
DFS will likely find a deep goal node first.

Combining The Best of BFS and DFS

Can we create a search algorithm
that combines the best of BFS and DFS?

BFS	**DFS**
$O(b^d)$ exponential space	$O(bm)$ linear space
Guaranteed to find a solution if one exists	May get stuck on infinite paths

Can we bring the best of two together?

Combining The Best of BFS and DFS

Can we create a search algorithm
that combines the best of BFS and DFS?

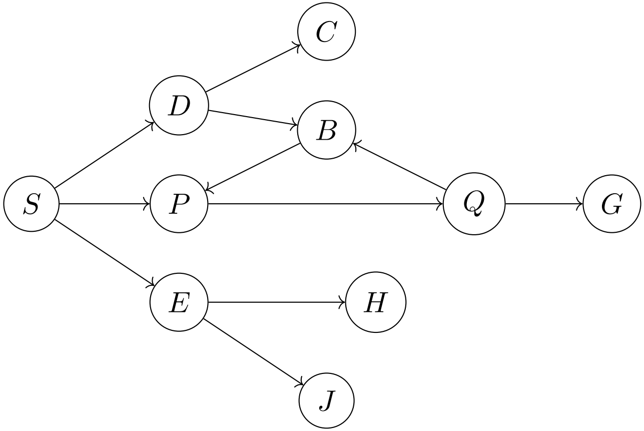
BFS	**DFS**
$O(b^d)$ exponential space	$O(bm)$ linear space
Guaranteed to find a solution if one exists	May get stuck on infinite paths

Can we bring the best of two together?

Iterative-Deepening Search:
For every depth limit,
perform depth-first search until the depth limit is reached.

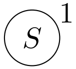
Trace IDS on a Search Graph

- Trace IDS on the search graph below.
- Add nodes to the frontier in alphabetical order.



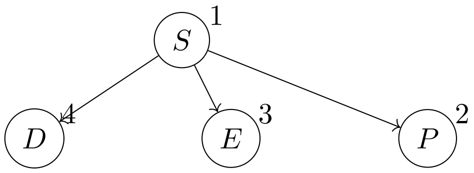
Trace IDS on a Search Graph

Depth: 1



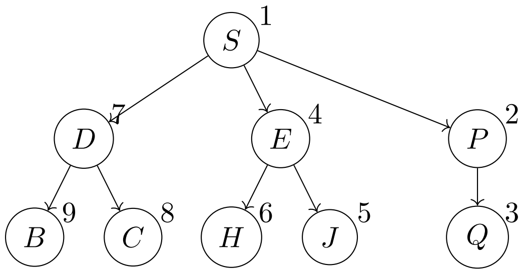
Trace IDS on a Search Graph

Depth: 2



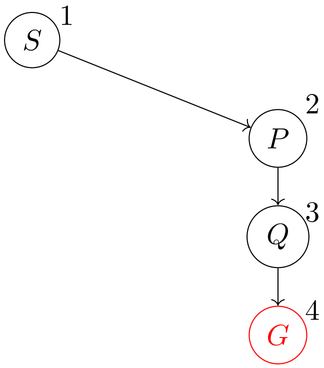
Trace IDS on a Search Graph

Depth: 3



Trace IDS on a Search Graph

Depth: 4



Properties of IDS - Space Complexity

Space Complexity

Properties of IDS - Space Complexity

Space Complexity

- $O(bd)$

  b is the branching factor.
  d is the depth of the shallowest goal node.

- Linear in d.
  Similar to DFS.

- Executes DFS for each depth limit.
  Guaranteed to terminate at depth d.

Properties of IDS - Time Complexity

Time Complexity

Properties of IDS - Time Complexity

Time Complexity

- $O(b^d)$
- Exponential in d.
  Same as BFS.
- Visits all the nodes on the top d levels in the worst case.

Properties of IDS - Completeness

Is IDS guaranteed to find a solution if a solution exists?

Properties of IDS - Completeness

Is IDS guaranteed to find a solution if a solution exists?

- Yes.
  Same as BFS.

- Explores the tree level by level until it finds a goal.

Properties of IDS - Optimality

Is IDS guaranteed to return an optimal solution if it terminates?

Properties of IDS - Optimality

Is IDS guaranteed to return an optimal solution if it terminates?

- Yes and No.
- Guaranteed to find the shallowest goal node. It's the lowest-cost solution if all edge has the same cost.
  Same as BFS.

A Summary of IDS Properties

- Space Complexity:

  $O(bd)$, linear in d. Similar to DFS.

- Time Complexity:

  $O(b^d)$, exponential in d. Same as BFS.

- Completeness:

  Yes. Same as BFS.

- Optimality:

  No, but guaranteed to find the shallowest goal node.
  Same as BFS.

Learning goals

- Formulate a real-world problem as a search problem.
- Trace the execution of and implement uninformed search algorithms (Breadth-first search, Depth-first search, Iterative-deepening search).
- Given an uninformed search algorithm, explain its space complexity, time complexity, and whether it has any guarantees on the quality of the solution found.
- Given a scenario, explain whether and why it is appropriate to use an uninformed algorithm.