UIT2504 Artificial Intelligence

Basic Search Strategies

C. Aravindan
<AravindanC@ssn.edu.in>

Professor of Computer Science
SSN College of Engineering

August 07, 2024

State-Space Approach

- Given an initial state, and a set of actions, "search tree" can be visualized
- The number of possible actions determine the branching factor of the tree
- State space is the set of all states that are reachable from the initial state using a sequence of actions
- Which states will be leaf nodes in the search tree? — all goal states will be leaf nodes — there can also be "dead-end" states where no action is possible
- Where is the solution in the search tree? — normally a path from initial state to a goal state
- In some cases, goal state itself is a solution!

Problem Formulation

- A problem may be formulated in the state-space approach by defining the following four components:
  - Set of states, with the initial state
  - Set of possible actions, which may be abstracted by a successor function S. For any state x, $S(x)$ is the set of states reachable from x using an action
  - Goal test — a function that returns true if a given state is a goal state
  - Path cost — sum of all the costs of actions from initial state to a goal state

Illustration: 8-Puzzle

- *State*: Arrangement of eight tiles on a 3×3 grid — *Initial state* is some random arrangement of eight tiles on nine squares
- *Actions*: blank moves left, right, up, or down — branching factor of 4
- *Goal test*: state matches the given target configuration
- *Path cost*: unit cost for each action

Illustration: 8-Puzzle

3	2	7
5	8	
1	4	6

Illustration: 8-Puzzle

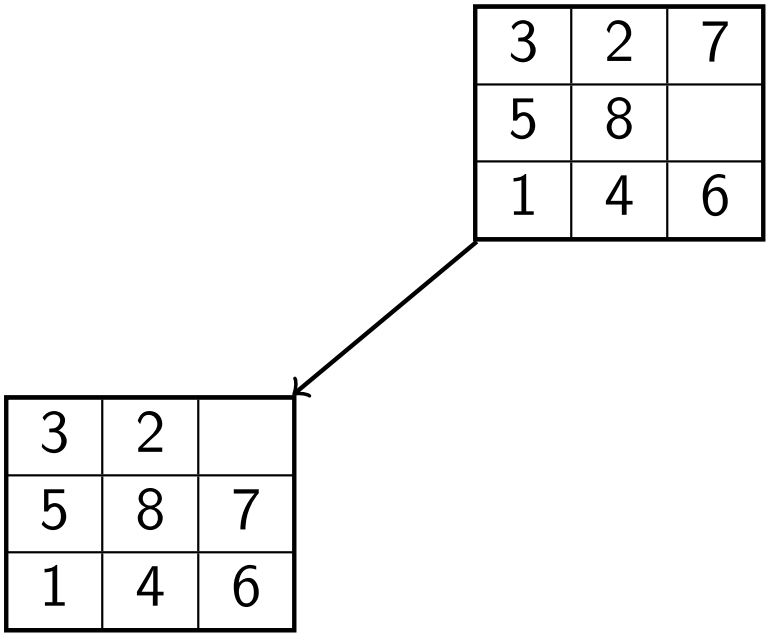


Illustration: 8-Puzzle

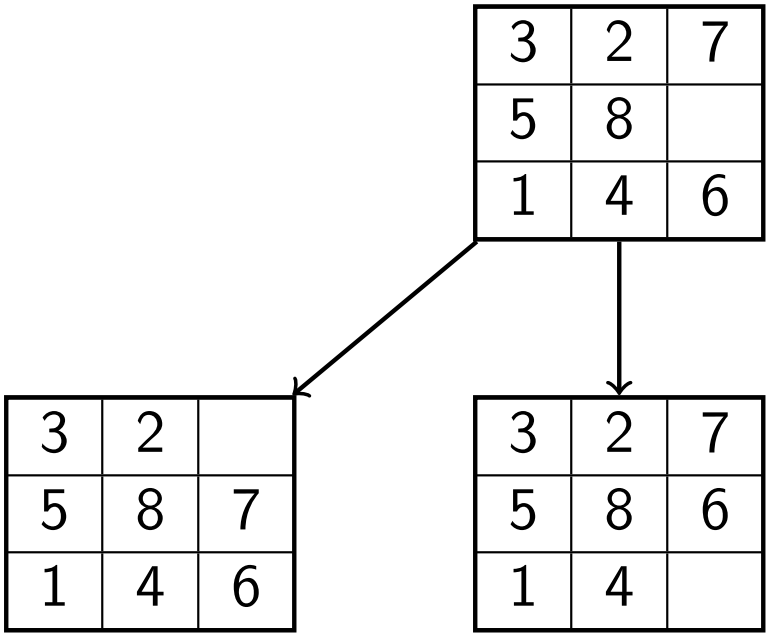


Illustration: 8-Puzzle

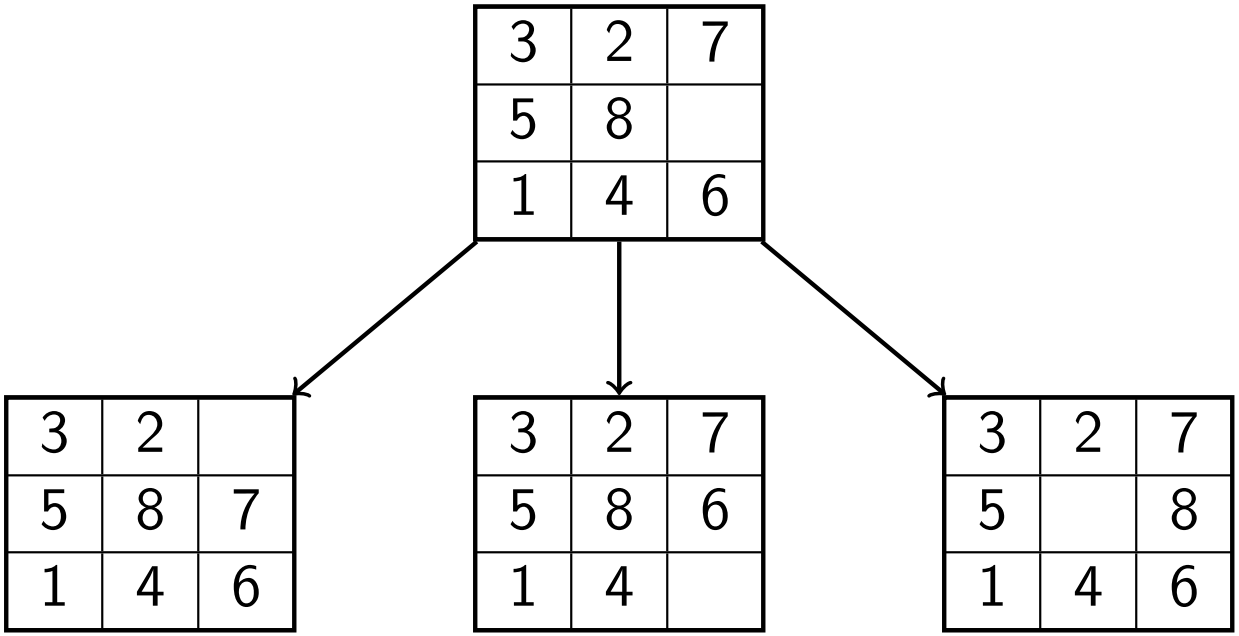


Illustration: 8-Puzzle

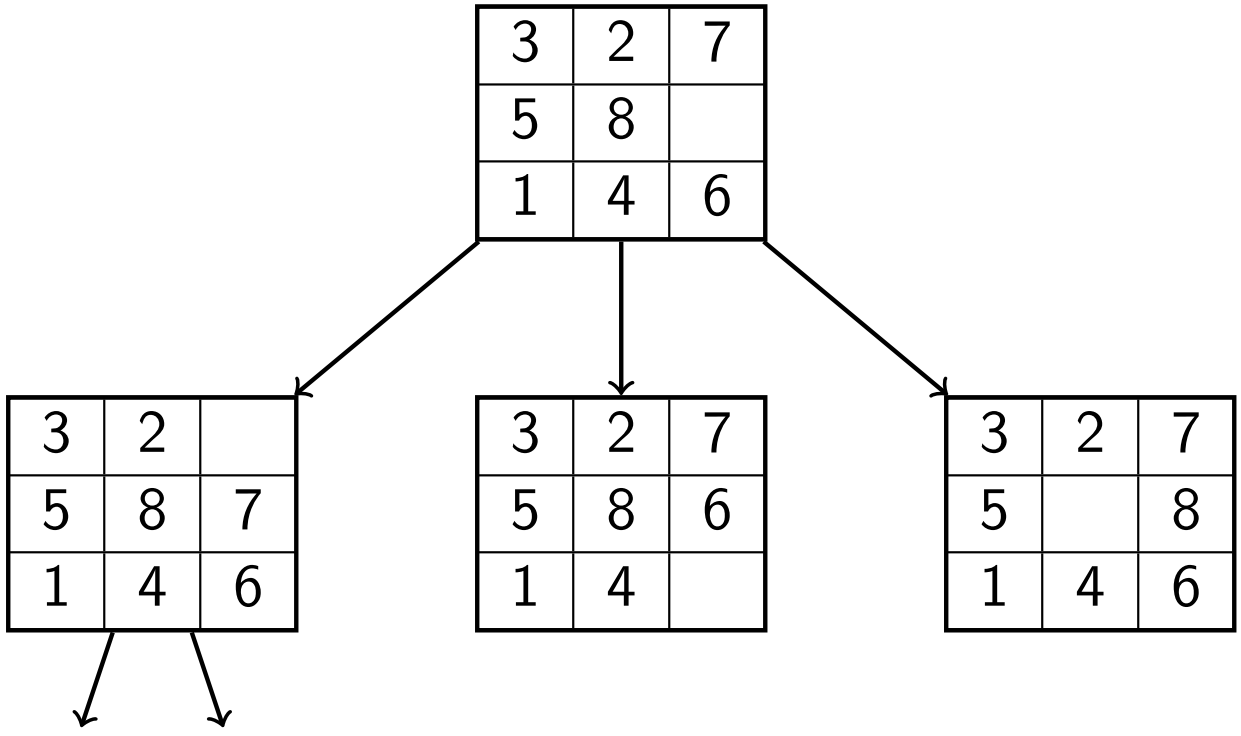


Illustration: 8-Puzzle

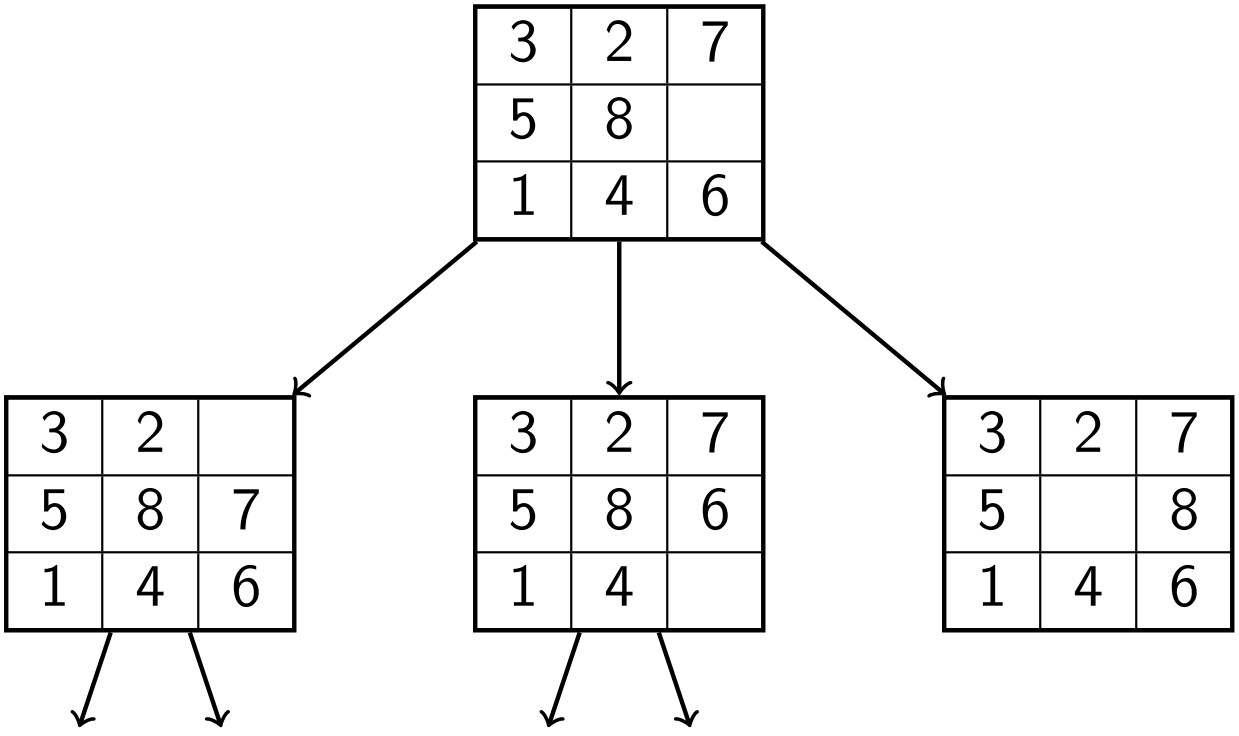


Illustration: 8-Puzzle

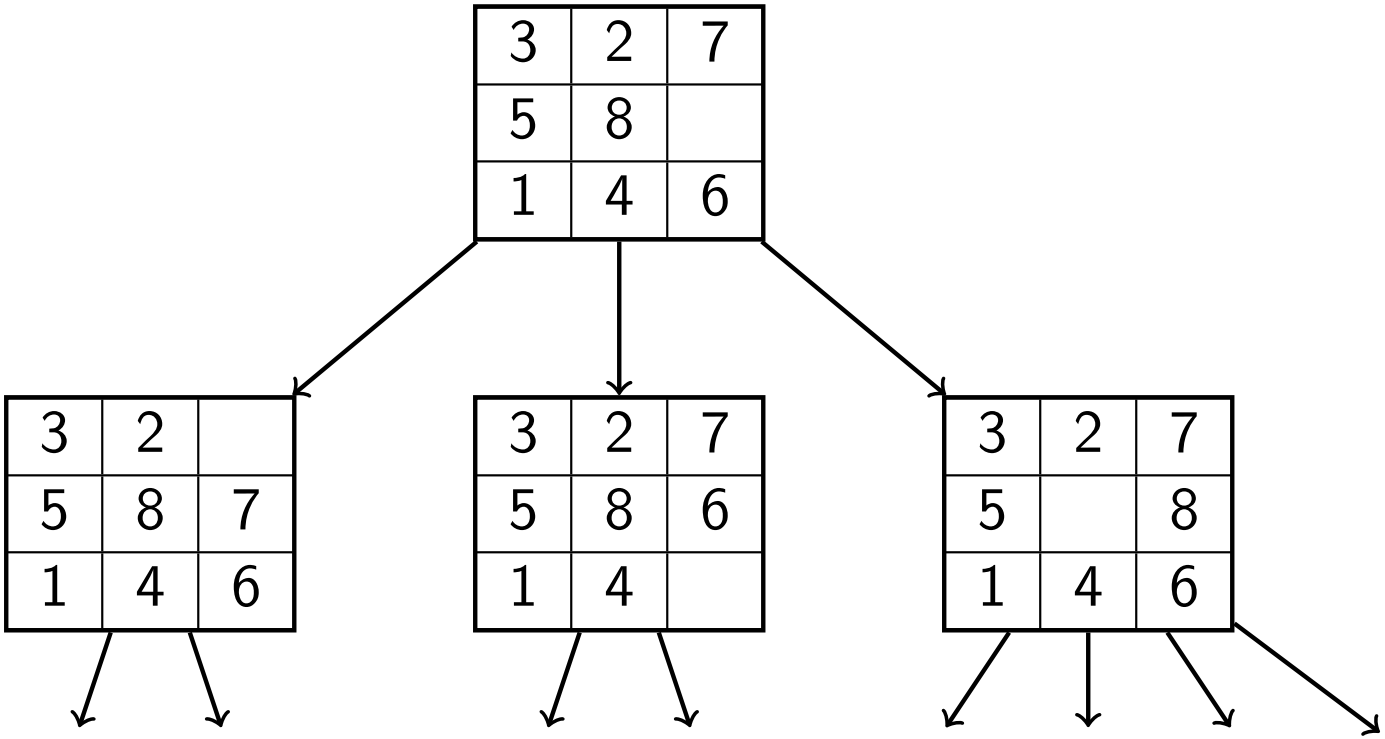
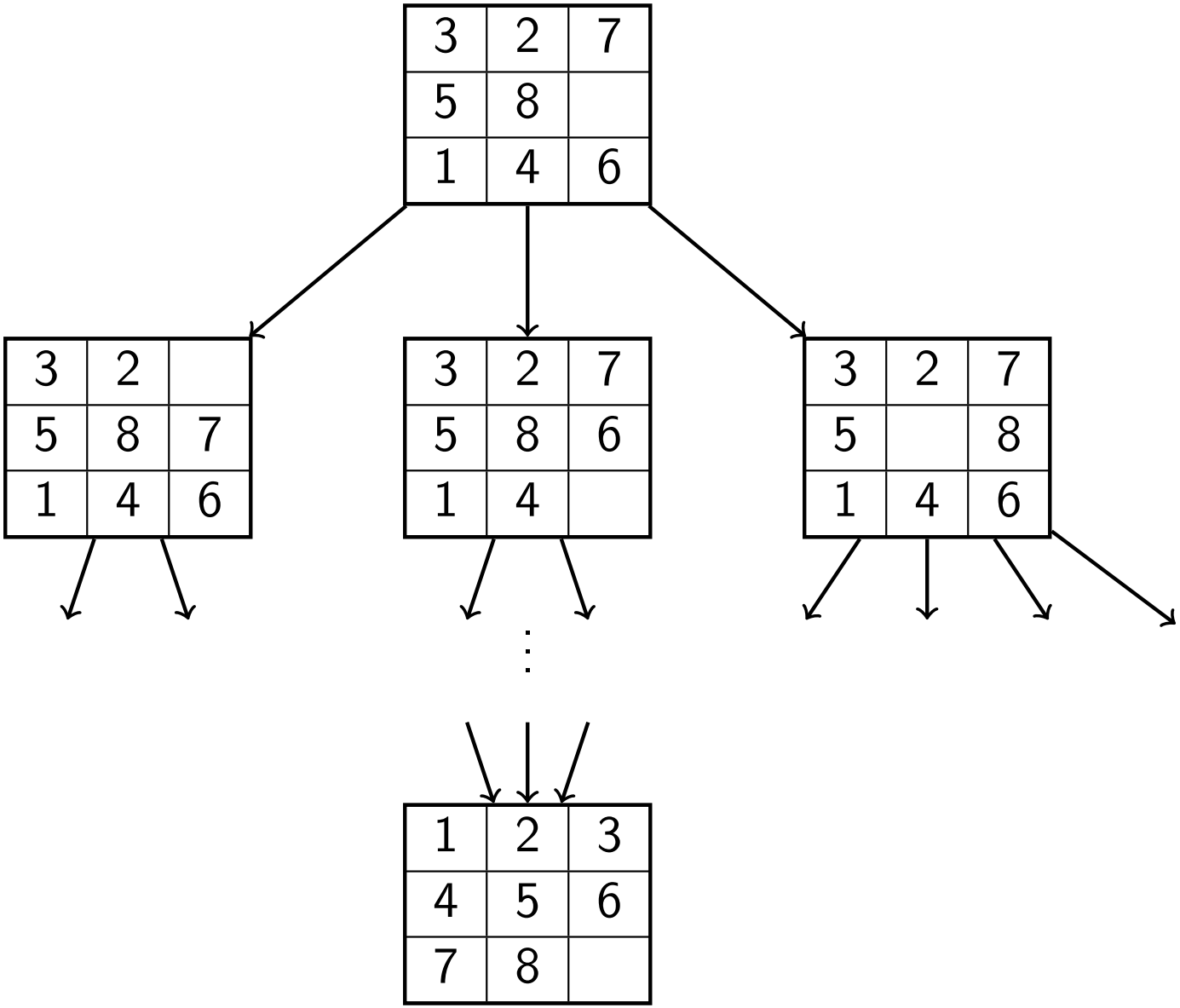


Illustration: 8-Puzzle



Searching for a solution

- Start with initial state in the working set

Searching for a solution

- Start with initial state in the working set
- Iterate:
  - Return failure if the working set is empty

Searching for a solution

- Start with initial state in the working set
- Iterate:
  - Return failure if the working set is empty
  - Choose and remove a state x from the working set

Searching for a solution

- Start with initial state in the working set
- Iterate:
  - Return failure if the working set is empty
  - Choose and remove a state x from the working set
  - If it is a goal state, return solution

Searching for a solution

- Start with initial state in the working set
- Iterate:
  - Return failure if the working set is empty
  - Choose and remove a state x from the working set
  - If it is a goal state, return solution
  - Else, expand x and add the successor states $S(x)$ to the working set

Search Strategies

- Uninformed:
  - Breadth-First
  - Depth-First
  - Depth-Limited
  - Bi-Directional Search
- Informed (Heuristics):
  - Best-first Greedy
  - A^*
  - Local Search Strategies
- Constraint Satisfaction

Performance Measures

- Completeness
- Time Complexity
- Space Complexity
- Optimality

Questions?

Breadth-First Search

- The working set is maintained as a FIFO Queue

Breadth-First Search

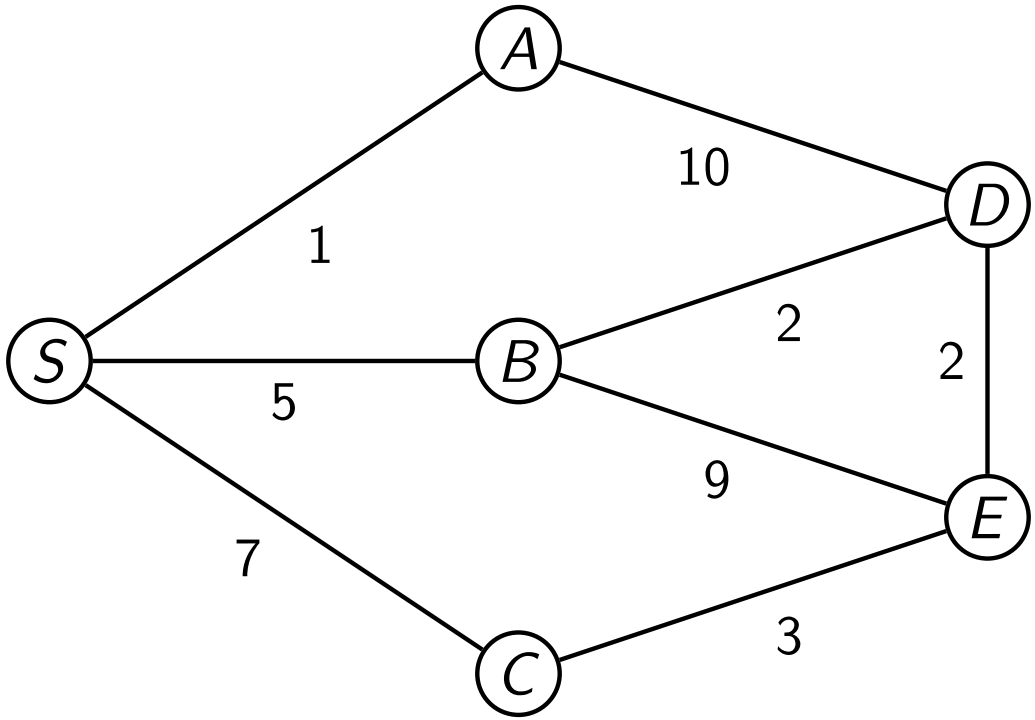
- The working set is maintained as a FIFO Queue
- Newly generated nodes are added at the end of the queue and always select the first node from the queue for examination

Breadth-First Search

- The working set is maintained as a FIFO Queue
- Newly generated nodes are added at the end of the queue and always select the first node from the queue for examination
- All nodes at depth d are expanded before any node at depth $d + 1$

BFS: Illustration

- How does it work on our toy example?



Find a route from S to E

BFS: Illustration

- Node examination order: $S, A, B, C, S, D, S, D, E$

BFS: Illustration

- Node examination order: $S, A, B, C, S, D, S, D, E$
- Solution found: $S \rightarrow B \rightarrow E$ with a path cost of 14

BFS: Illustration

- Node examination order: $S, A, B, C, S, D, S, D, E$
- Solution found: $S \rightarrow B \rightarrow E$ with a path cost of 14
- What is left in the queue? $\langle S, E, A, B, C, A, B, E, A, B, C, A, B, E \rangle$

BFS: Illustration

- Node examination order: $S, A, B, C, S, D, S, D, E$
- Solution found: $S \rightarrow B \rightarrow E$ with a path cost of 14
- What is left in the queue? $\langle S, E, A, B, C, A, B, E, A, B, C, A, B, E \rangle$
- From tree point of view, nodes are examined in the level order

BFS: Illustration

- Node examination order: $S, A, B, C, S, D, S, D, E$
- Solution found: $S \rightarrow B \rightarrow E$ with a path cost of 14
- What is left in the queue? $\langle S, E, A, B, C, A, B, E, A, B, C, A, B, E \rangle$
- From tree point of view, nodes are examined in the level order
- Each and every path is progressively examined!

BFS: Complexities

- Is BFS complete? — that is, will it find a solution if one exists?

BFS: Complexities

- Is BFS complete? — that is, will it find a solution if one exists? — Yes!

BFS: Complexities

- Is BFS complete? — that is, will it find a solution if one exists? — Yes! — but the branching factor b should be finite!

BFS: Complexities

- Is BFS complete? — that is, will it find a solution if one exists? — Yes! — but the branching factor b should be finite!
- Is it optimal?

BFS: Complexities

- Is BFS complete? — that is, will it find a solution if one exists? — Yes! — but the branching factor b should be finite!
- Is it optimal? — No, unless optimality is defined by the length of the path

BFS: Complexities

- Is BFS complete? — that is, will it find a solution if one exists? — Yes! — but the branching factor b should be finite!
- Is it optimal? — No, unless optimality is defined by the length of the path
- Time complexity?

BFS: Complexities

- Is BFS complete? — that is, will it find a solution if one exists? — Yes! — but the branching factor b should be finite!
- Is it optimal? — No, unless optimality is defined by the length of the path
- Time complexity? — $1 + b + b^2 + \cdots + b^d + (b^{d+1} - b)$

BFS: Complexities

- Is BFS complete? — that is, will it find a solution if one exists? — Yes! — but the branching factor b should be finite!
- Is it optimal? — No, unless optimality is defined by the length of the path
- Time complexity? — $1 + b + b^2 + \cdots + b^d + (b^{d+1} - b)$ — $O(b^{d+1})$

BFS: Complexities

- Is BFS complete? — that is, will it find a solution if one exists? — Yes! — but the branching factor b should be finite!
- Is it optimal? — No, unless optimality is defined by the length of the path
- Time complexity? — $1 + b + b^2 + \cdots + b^d + (b^{d+1} - b)$ — $O(b^{d+1})$
- Space complexity?

BFS: Complexities

- Is BFS complete? — that is, will it find a solution if one exists? — Yes! — but the branching factor b should be finite!
- Is it optimal? — No, unless optimality is defined by the length of the path
- Time complexity? — $1 + b + b^2 + \cdots + b^d + (b^{d+1} - b)$ — $O(b^{d+1})$
- Space complexity? — same as that of time complexity

BFS: Complexities

- Is BFS complete? — that is, will it find a solution if one exists? — Yes! — but the branching factor b should be finite!
- Is it optimal? — No, unless optimality is defined by the length of the path
- Time complexity? — $1 + b + b^2 + \cdots + b^d + (b^{d+1} - b)$ — $O(b^{d+1})$
- Space complexity? — same as that of time complexity — $O(b^{d+1})$

Questions?

Uniform-Cost Search

- Generalization of breadth-first search

Uniform-Cost Search

- Generalization of breadth-first search
- Working set (fringe) is a partial order arranged in ascending order of cost (minheap)

Uniform-Cost Search

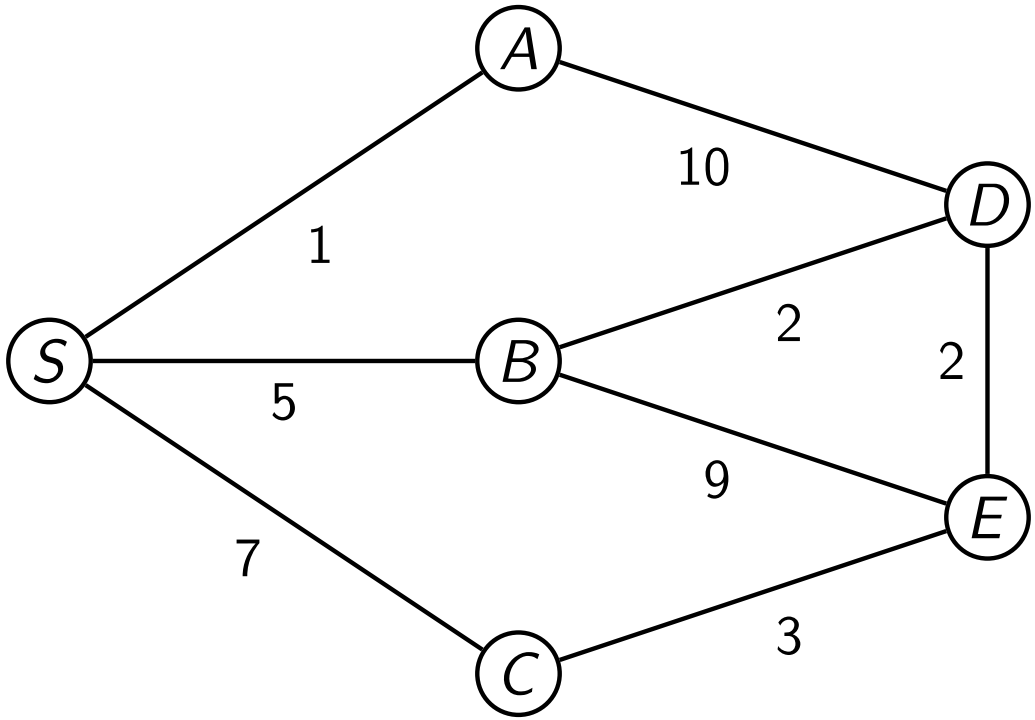
- Generalization of breadth-first search
- Working set (fringe) is a partial order arranged in ascending order of cost (minheap)
- State with the lowest path cost in the fringe (which is the first element in the partial order) is selected

Uniform-Cost Search

- Generalization of breadth-first search
- Working set (fringe) is a partial order arranged in ascending order of cost (minheap)
- State with the lowest path cost in the fringe (which is the first element in the partial order) is selected
- Does this remind you of any graph alsorithm that you have learnt?

UCS: Illustration

- How does it work on our toy example?



Find a route from S to E

UCS: Illustration

- Node examination order: $S(0), A(1), B(5), C(7), D(7), E(9)$

UCS: Illustration

- Node examination order: $S(0), A(1), B(5), C(7), D(7), E(9)$
- Solution found: $S \rightarrow B \rightarrow D \rightarrow E$ with a path cost of 9

UCS: Illustration

- Node examination order: $S(0), A(1), B(5), C(7), D(7), E(9)$
- Solution found: $S \rightarrow B \rightarrow D \rightarrow E$ with a path cost of 9
- Selectively switches between the paths!

UCS: Complexities

- Is UCS complete? — that is, will it find a solution if one exists?

UCS: Complexities

- Is UCS complete? — that is, will it find a solution if one exists? — Yes!

UCS: Complexities

- Is UCS complete? — that is, will it find a solution if one exists? — Yes! — but there should not be any negative cost!

UCS: Complexities

- Is UCS complete? — that is, will it find a solution if one exists? — Yes! — but there should not be any negative cost!
- Is it optimal?

UCS: Complexities

- Is UCS complete? — that is, will it find a solution if one exists? — Yes! — but there should not be any negative cost!
- Is it optimal? — Yes, if the cost function is monotonic

UCS: Complexities

- Is UCS complete? — that is, will it find a solution if one exists? — Yes! — but there should not be any negative cost!
- Is it optimal? — Yes, if the cost function is monotonic
- Time complexity?

UCS: Complexities

- Is UCS complete? — that is, will it find a solution if one exists? — Yes! — but there should not be any negative cost!
- Is it optimal? — Yes, if the cost function is monotonic
- Time complexity? — similar to that of breadth-first —

UCS: Complexities

- Is UCS complete? — that is, will it find a solution if one exists? — Yes! — but there should not be any negative cost!
- Is it optimal? — Yes, if the cost function is monotonic
- Time complexity? — similar to that of breadth-first — $O(b^{\lceil C^*/e \rceil})$

UCS: Complexities

- Is UCS complete? — that is, will it find a solution if one exists? — Yes! — but there should not be any negative cost!
- Is it optimal? — Yes, if the cost function is monotonic
- Time complexity? — similar to that of breadth-first — $O(b^{\lceil C^*/e \rceil})$
- Space complexity?

UCS: Complexities

- Is UCS complete? — that is, will it find a solution if one exists? — Yes! — but there should not be any negative cost!
- Is it optimal? — Yes, if the cost function is monotonic
- Time complexity? — similar to that of breadth-first — $O(b^{\lceil C^*/e \rceil})$
- Space complexity? — same as that of time complexity

UCS: Complexities

- Is UCS complete? — that is, will it find a solution if one exists? — Yes! — but there should not be any negative cost!
- Is it optimal? — Yes, if the cost function is monotonic
- Time complexity? — similar to that of breadth-first — $O(b^{\lceil C^*/e \rceil})$
- Space complexity? — same as that of time complexity — $O(b^{\lceil C^*/e \rceil})$

Questions?

Depth-First Search

- Commit to a path

Depth-First Search

- Commit to a path — path switch occurs only when "dead-leafs" are encountered

Depth-First Search

- Commit to a path — path switch occurs only when “dead-leafs” are encountered
- Working set is a LIFO Stack

Depth-First Search

- Commit to a path — path switch occurs only when "dead-leafs" are encountered
- Working set is a LIFO Stack — newly generated nodes are added at the top of the stack

Depth-First Search

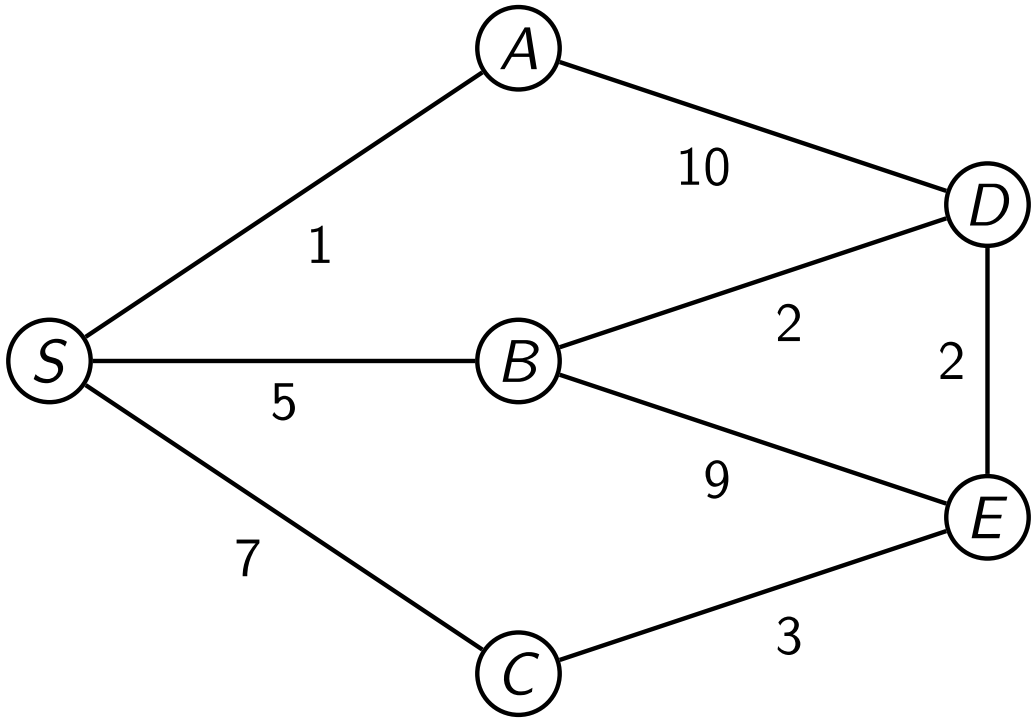
- Commit to a path — path switch occurs only when "dead-leafs" are encountered
- Working set is a LIFO Stack — newly generated nodes are added at the top of the stack — and the node at the top is always selected

Depth-First Search

- Commit to a path — path switch occurs only when "dead-leafs" are encountered
- Working set is a LIFO Stack — newly generated nodes are added at the top of the stack — and the node at the top is always selected
- Preference is given to newly generated nodes

DFS: Illustration

- How does it work on our toy example?



Find a route from S to E

DFS: Illustration

- Node examination order: S, A, D, B, E

DFS: Illustration

- Node examination order: S, A, D, B, E
- Solution found: $S \rightarrow A \rightarrow D \rightarrow B \rightarrow E$ with path cost of 22

DFS: Complexities

- Is DFS complete?

DFS: Complexities

- Is DFS complete? — No! — not in general

DFS: Complexities

- Is DFS complete? — No! — not in general
- Is it optimal?

DFS: Complexities

- Is DFS complete? — No! — not in general
- Is it optimal? — No!

DFS: Complexities

- Is DFS complete? — No! — not in general
- Is it optimal? — No!
- Time complexity?

DFS: Complexities

- Is DFS complete? — No! — not in general
- Is it optimal? — No!
- Time complexity? — $O(b^m)$

DFS: Complexities

- Is DFS complete? — No! — not in general
- Is it optimal? — No!
- Time complexity? — $O(b^m)$
- Space complexity?

DFS: Complexities

- Is DFS complete? — No! — not in general
- Is it optimal? — No!
- Time complexity? — $O(b^m)$
- Space complexity? — $O(bm)$

Backtracking

- Backtracking is a variation of depth-first search where only one successor is generated at a time

Backtracking

- Backtracking is a variation of depth-first search where only one successor is generated at a time
- When a "dead-leaf" is encountered, or when search fails for all the successors, the algorithm backtracks to the parent node to generate the next successor

Backtracking

- Backtracking is a variation of depth-first search where only one successor is generated at a time
- When a "dead-leaf" is encountered, or when search fails for all the successors, the algorithm backtracks to the parent node to generate the next successor
- It is easy to implement using recursion — usage of stack is implicit!

Backtracking

- Backtracking is a variation of depth-first search where only one successor is generated at a time
- When a "dead-leaf" is encountered, or when search fails for all the successors, the algorithm backtracks to the parent node to generate the next successor
- It is easy to implement using recursion — usage of stack is implicit!
- Space complexity?

Backtracking

- Backtracking is a variation of depth-first search where only one successor is generated at a time
- When a "dead-leaf" is encountered, or when search fails for all the successors, the algorithm backtracks to the parent node to generate the next successor
- It is easy to implement using recursion — usage of stack is implicit!
- Space complexity? — $O(m)$

Questions?

Can we do better?

- Is it possible to do better?

Can we do better?

- Is it possible to do better?
- May be not wrt time complexity

Can we do better?

- Is it possible to do better?
- May be not wrt time complexity
- Can we design an algorithm that is complete (like BFS) and with linear space complexity (like DFS)?

Can we do better?

- Is it possible to do better?
- May be not wrt time complexity
- Can we design an algorithm that is complete (like BFS) and with linear space complexity (like DFS)?
- Can you suggest some modification to DFS to make it complete?