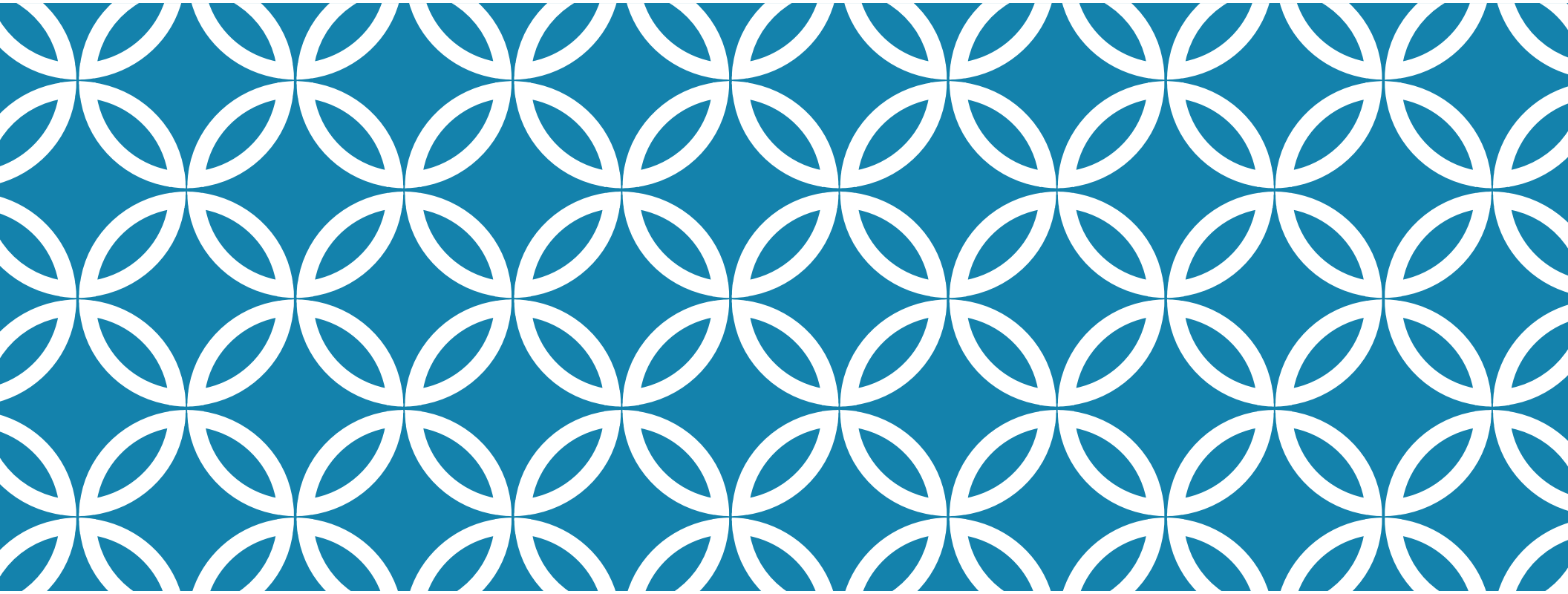


ARTIFICIAL INTELLIGENCE

Lecture 05

BREADTH FIRST SEARCH

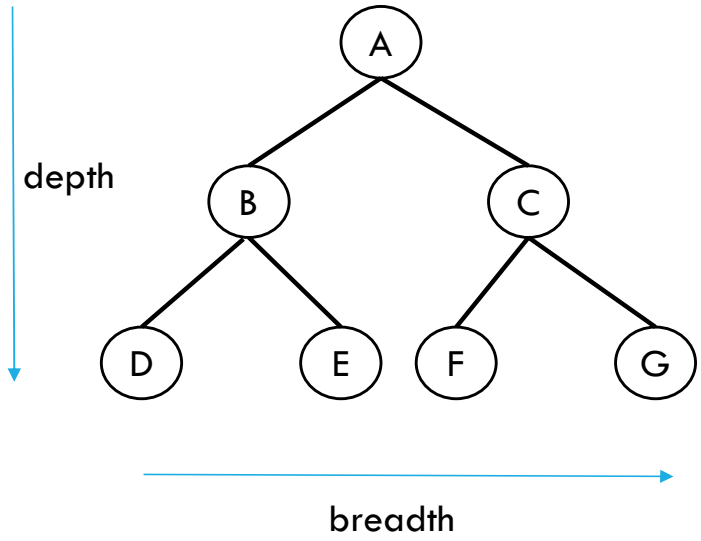
Expand Shallowest Node First

Frontier (or fringe): nodes in queue to be explored

Explored Set: Nodes that are already explored

Frontier is a first-in-first-out (FIFO) queue, i.e., new successors go at end of the queue.

Goal-Test when inserted



BREADTH FIRST SEARCH

Expand Shallowest Node First

Frontier (or fringe): nodes in queue to be explored

Explored Set: Nodes that are already explored

Frontier is a first-in-first-out (FIFO) queue, i.e., new successors go at end of the queue.

Goal-Test when inserted

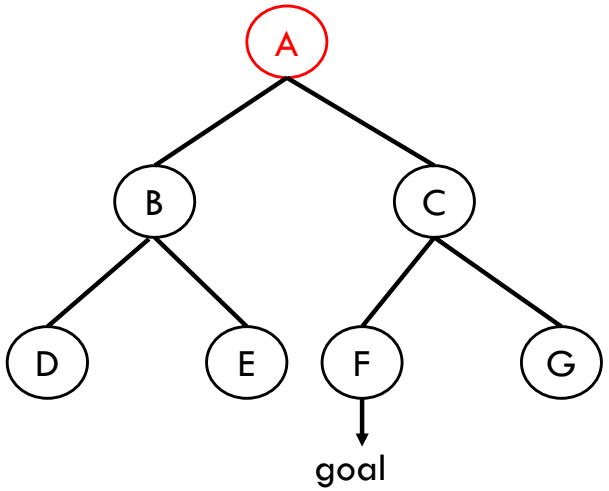
```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)   /* chooses the shallowest node in frontier */
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
              frontier ← INSERT(child, frontier)
```

BREADTH FIRST SEARCH

Is A a goal state?

Frontier = [A]

Explored = [ ]

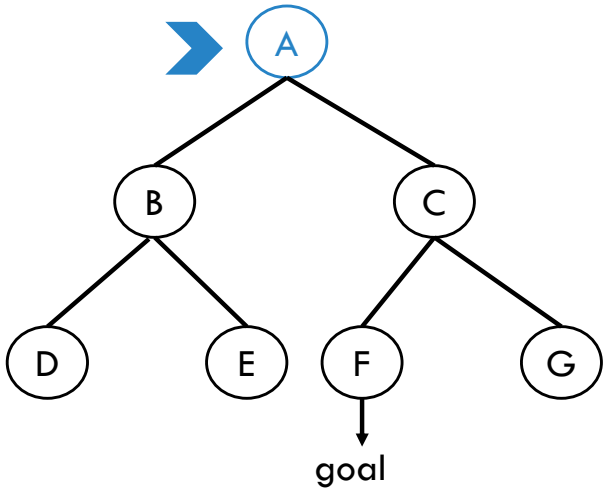



```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```

BREADTH FIRST SEARCH

Frontier = [A]
Explored = [ ]

Frontier = [ ]
Explored = [A]




```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)   /* chooses the shallowest node in frontier */
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
              frontier ← INSERT(child, frontier)
```

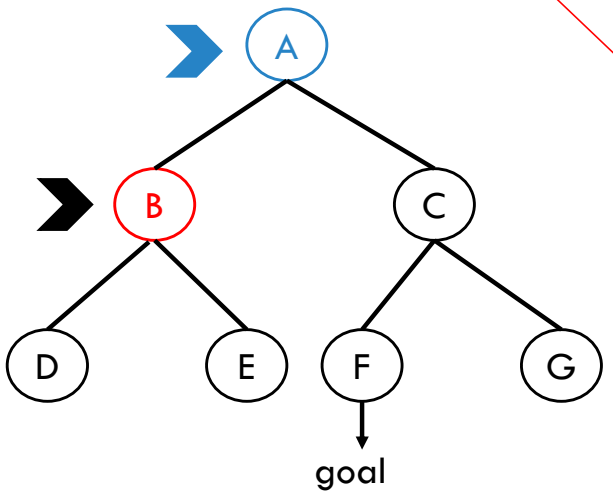
Node

Child

BREADTH FIRST SEARCH

Frontier = [ ]
Explored = [A]

Frontier = [B]
Explored = [A]




```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)   /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
Is B a goal state?  if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```

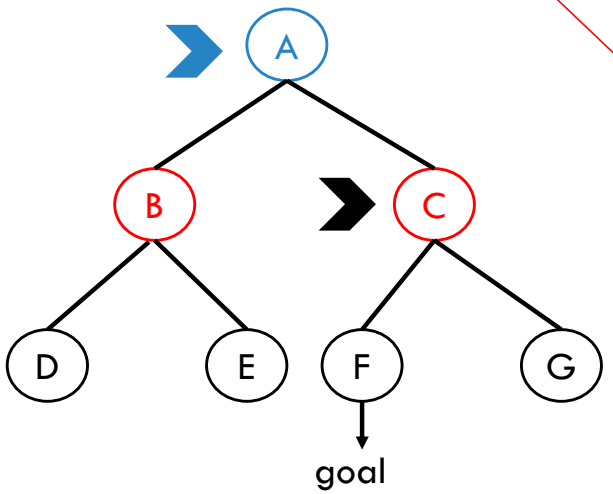
Node

Child

BREADTH FIRST SEARCH

Frontier = [B]
Explored = [A]

Frontier = [B, C]
Explored = [A]




```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)   /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
Is C a goal state?  if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
          frontier ← INSERT(child, frontier)
```

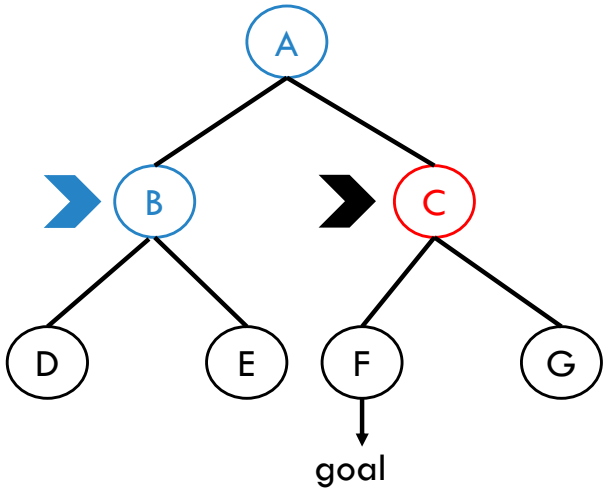
Node

Child

BREADTH FIRST SEARCH

Frontier = [B, C]
Explored = [A]

Frontier = [C]
Explored = [A, B]




```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)   /* chooses the shallowest node in frontier */
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
              frontier ← INSERT(child, frontier)
```

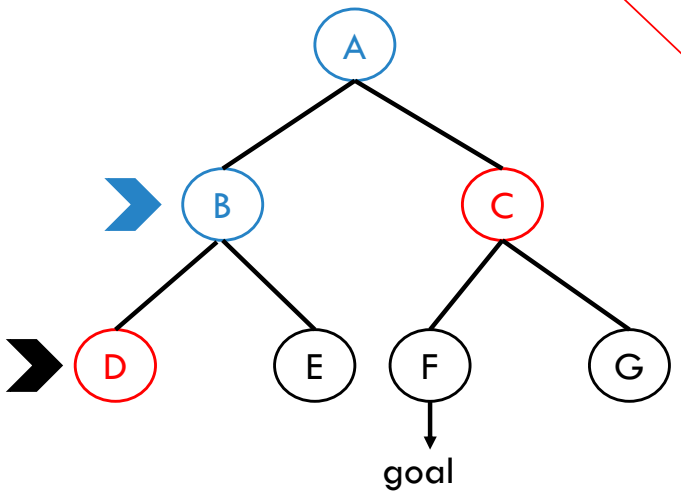
Node

Child

BREADTH FIRST SEARCH

Frontier = [C]
Explored = [A, B]

Frontier = [C, D]
Explored = [A, B]




```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)   /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
Is D a goal state?if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```

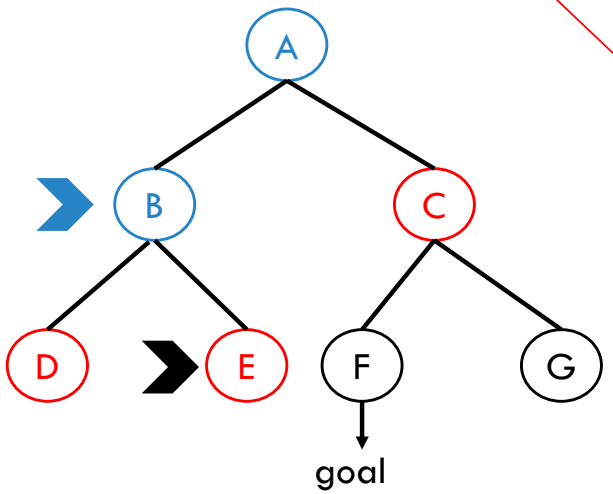
Node

Child

BREADTH FIRST SEARCH

Frontier = [C, D]
Explored = [A, B]

Frontier = [C, D, E]
Explored = [A, B]




```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)   /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
Is E a goal state?  if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
          frontier ← INSERT(child, frontier)
```

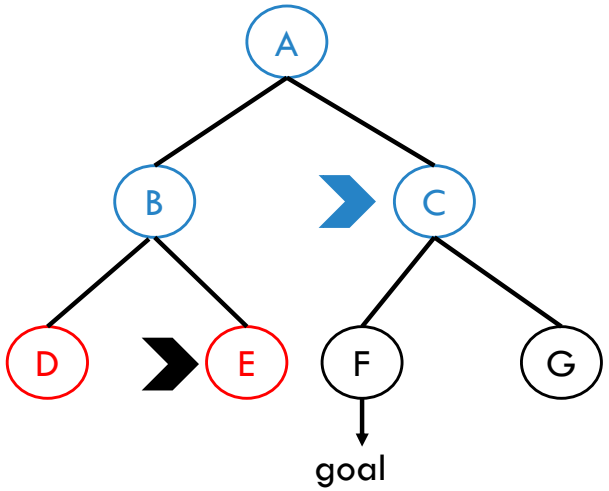
Node

Child

BREADTH FIRST SEARCH

Frontier = [C, D, E]
Explored = [A, B]

Frontier = [D, E]
Explored = [A, B, C]




```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)   /* chooses the shallowest node in frontier */
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
              frontier ← INSERT(child, frontier)
```

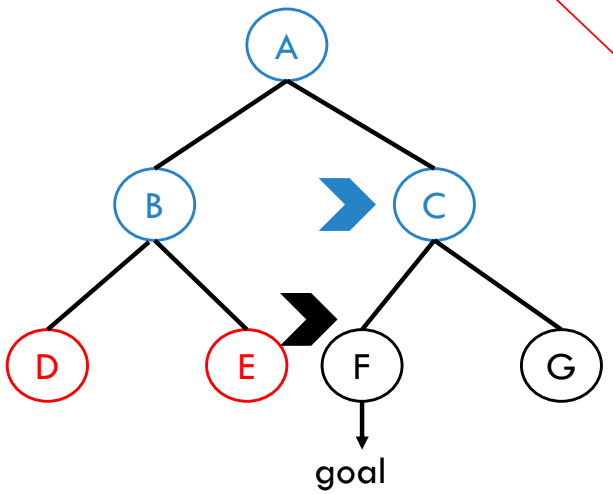
Node

Child

BREADTH FIRST SEARCH

Frontier = [D, E]
Explored = [A, B, C]

Frontier = [D, E]
Explored = [A, B, C]



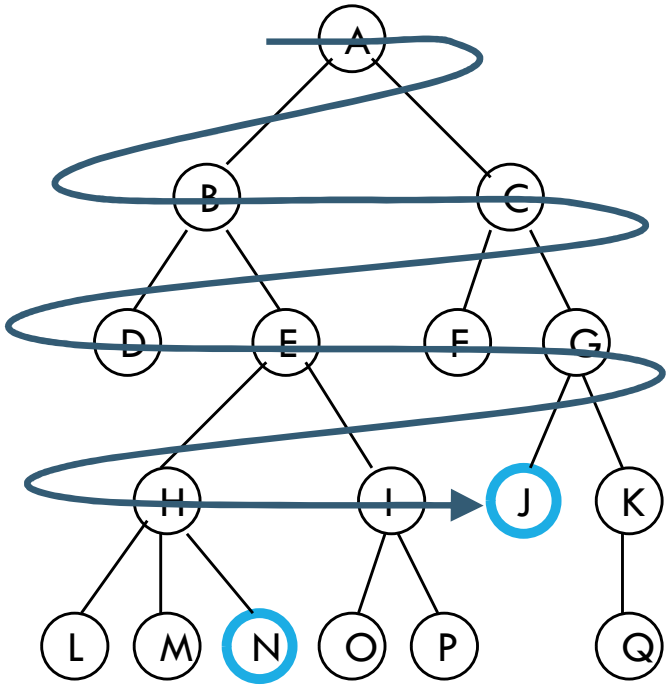

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)   /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```

Is F a goal state?

Node

Child

BFS: SUMMARY



BFS: COMPLETENESS

Completeness: Yes (guaranteed to find a solution if there exists one)

if the shallowest goal node is at some finite depth d, breadth-first search will eventually find it after generating all shallower nodes (provided the branching factor b is finite) .

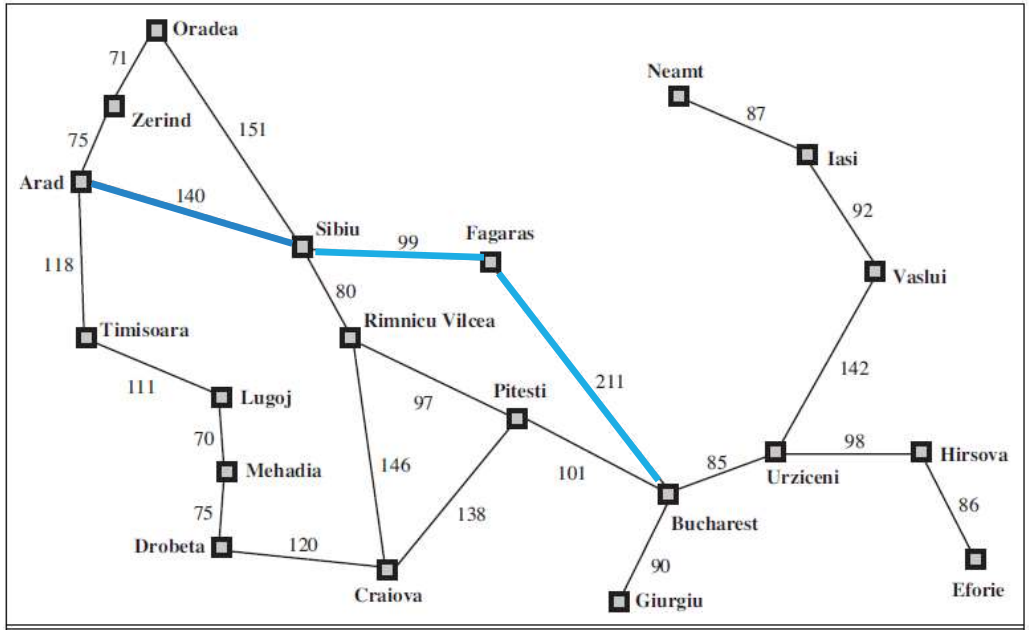


Figure 3.2 A simplified road map of part of Romania.

BFS goal path

BFS: OPTIMAL?

Not necessarily optimal

Only optimal if every action has same cost.

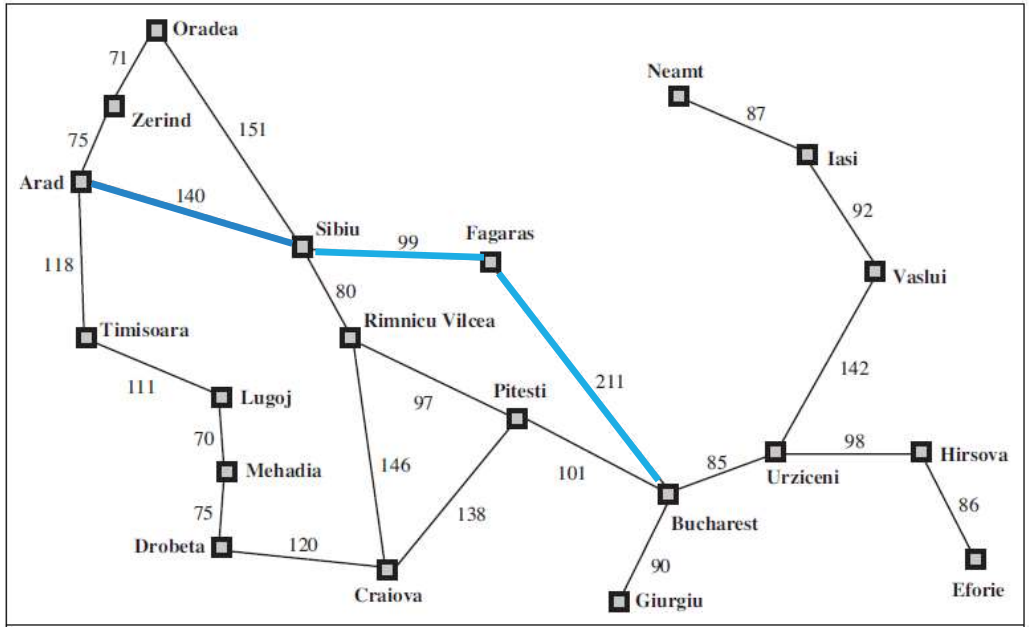


Figure 3.2 A simplified road map of part of Romania.

BFS goal path

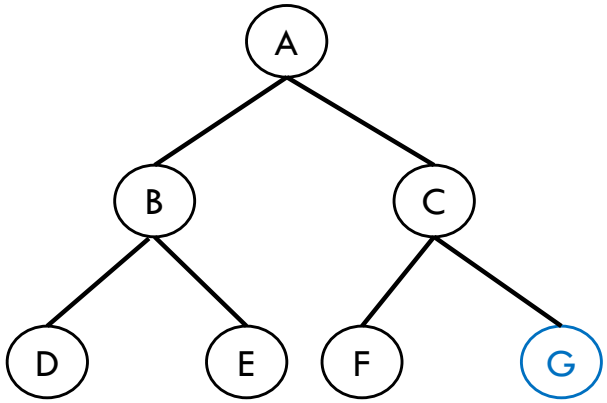
BFS: TIME COMPLEXITY?

Worst complexity is when G is a goal state.

In this case, total number of nodes generated is

$b+b^2+b^3 + \ldots + b^d = O(b^d)$ (The d^{th} layer contains nodes much larger than all the nodes in previous layers combined!)

So the time complexity is $O(b^d)$



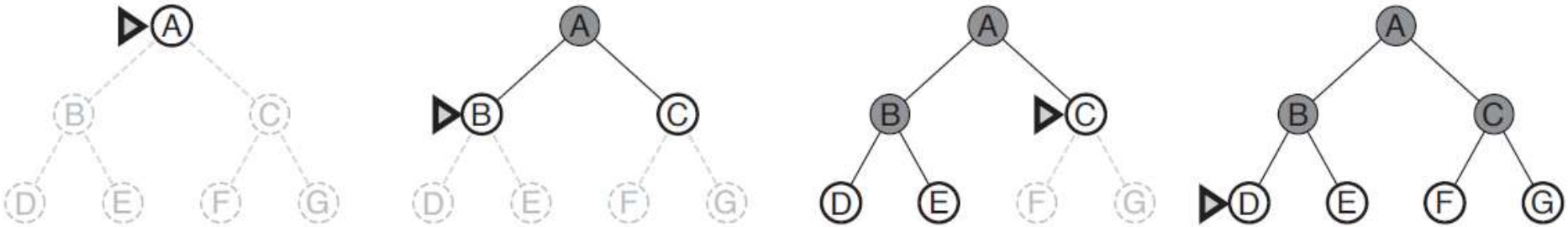
A binary tree, b=2, d=2

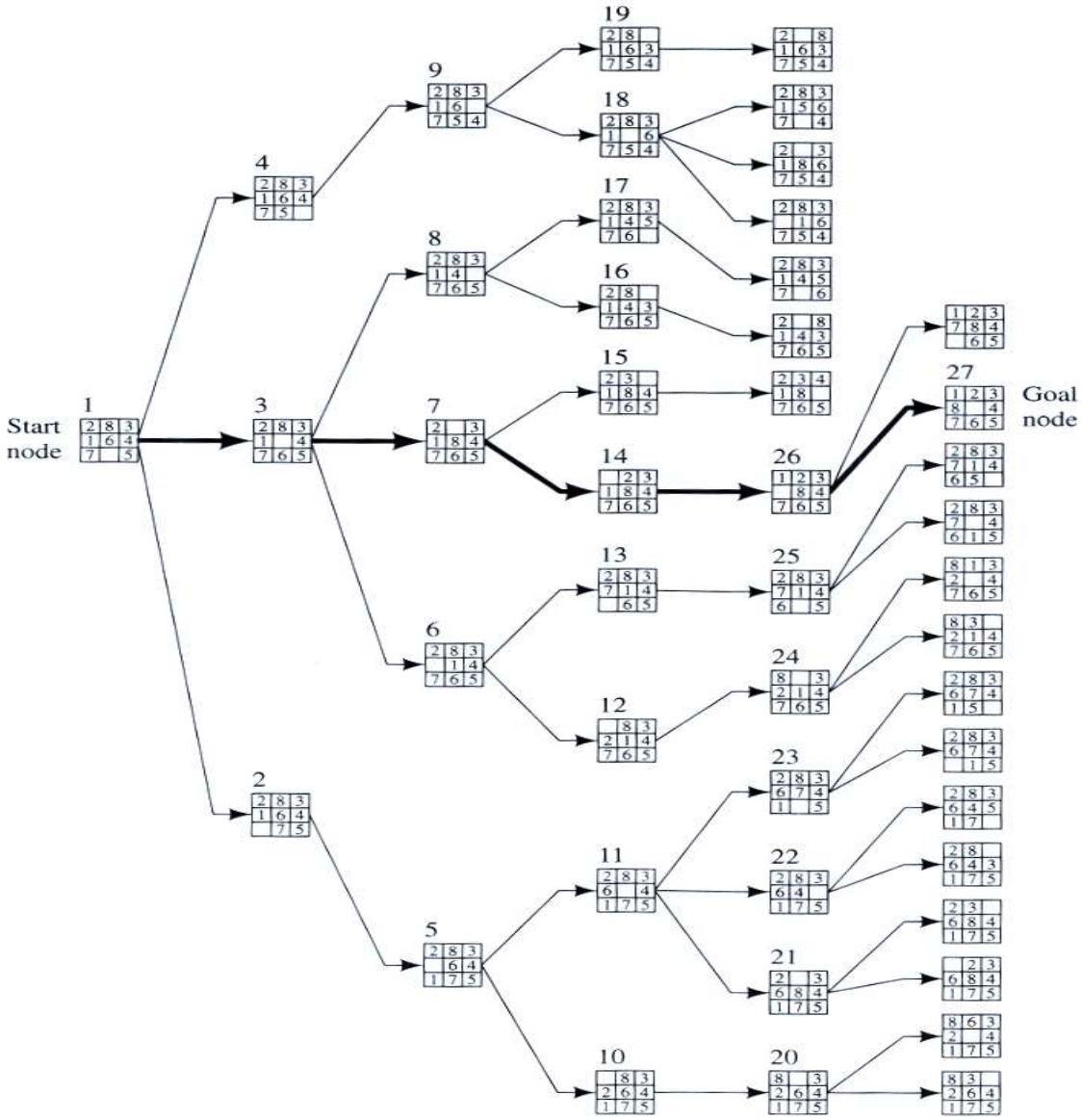
BFS: SPACE COMPLEXITY?

There will be $O(b^{d-1})$ nodes in the explored set and $O(b^d)$ nodes in the frontier.

So the space complexity is $O(b^d)$, i.e., it is dominated by the size of the frontier.

Exponential time complexity can be accepted but exponential space complexity is BAD !




Start node
Goal node
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27