UNINFORMED SEARCH STRATEGIES

Breadth-first search

UNINFORMED SEARCH STRATEGIES

- Breadth-first search
- Dijkstra's algorithm or uniform-cost search
- Depth-first search
- Depth-limited and iterative deepening search
- Bidirectional search

BREADTH-FIRST SEARCH

- Breadth-first search = Best-first search where evaluation function f(n) is the depth of the node.
- Breadth-first search uses **First in first out** queue whereas Best-first search uses **Priority** queue.
- Early goal test (Breadth-first search) rather late goal test (Best-first search).
- Spreads out in wave of uniform depth

BREADTH-FIRST SEARCH

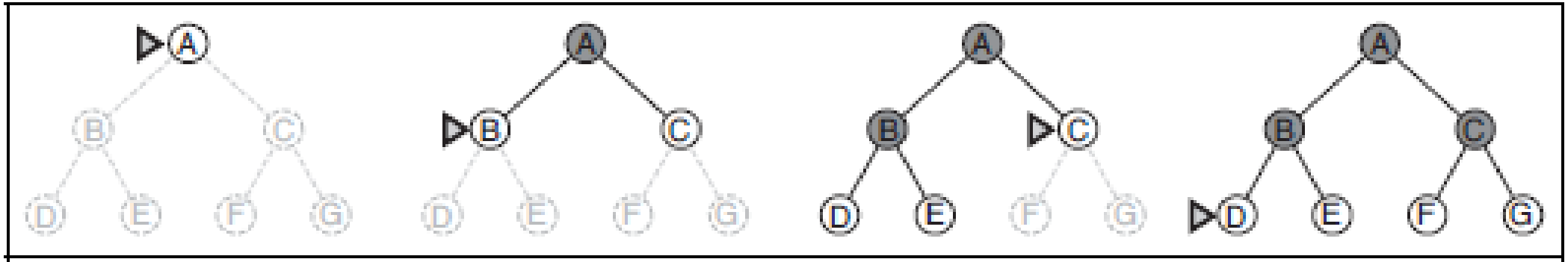

Figure 3.12 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

BREADTH-FIRST SEARCH

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure

function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s' ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s')
    yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

Iteration number	Frontier (Queue)	Reached (Look-up table)
0	Arad	Arad, Nil, Nil, 0
1	Sibiu, Zerind, Timisoara	Arad, Nil, Nil, 0 Sibiu, Arad, Go[Sibiu], 140 Zerind, Arad, Go[Zerind], 75 Timisoara, Arad, Go[Timisoara], 118
2	Zerind, Timisoara, Fagaras, Rimnicu Vilcea	Arad, Nil, Nil, 0 Sibiu, Arad, Go[Sibiu], 140 Zerind, Arad, Go[Zerind], 75 Timisoara, Arad, Go[Timisoara], 118 Fagaras, Sibiu, Go[Fagaras], 239 Rimnicu Vilcea, Sibiu, Go[R.V.], 220
3	Timisoara, Fagaras, Rimnicu Vilcea, Oradia	Arad, Nil, Nil, 0 Sibiu, Arad, Go[Sibiu], 140 Zerind, Arad, Go[Zerind], 75 Timisoara, Arad, Go[Timisoara], 118 Fagaras, Sibiu, Go[Fagaras], 239 Rimnicu Vilcea, Sibiu, Go[R.V.], 220 Oradia, Zerind, Go[Oradia], 146

Iteration number	Frontier (Queue)	Reached (Look-up table)
4	Fagaras, Rimnicu Vilcea, Oradia, Lugoj	Arad, Nil, Nil, 0 Sibiu, Arad, Go[Sibiu], 140 Zerind, Arad, Go[Zerind], 75 Timisoara, Arad, Go[Timisoara], 118 Fagaras, Sibiu, Go[Fagaras], 239 Rimnicu Vilcea, Sibiu, Go[R.V.], 220 Oradia, Zerind, Go[Oradia], 146 Lugoj, Timisoara, Go[Lugoj], 229
5	Rimnicu Vilcea, Oradia, Lugoj	Arad, Nil, Nil, 0 Sibiu, Arad, Go[Sibiu], 140 Zerind, Arad, Go[Zerind], 75 Timisoara, Arad, Go[Timisoara], 118 Fagaras, Sibiu, Go[Fagaras], 239 Rimnicu Vilcea, Sibiu, Go[R.V.], 220 Oradia, Zerind, Go[Oradia], 146 Lugoj, Timisoara, Go[Lugoj], 229

BREADTH-FIRST SEARCH (FINAL SOLUTION)

- Bucharest, Fagaras, Go[Bucharest], 450
- Fagaras, Sibiu, Go[Fagaras], 239
- Sibiu, Arad, Go[Sibiu], 140
- Arad, Nil, Nil, 0

- Arad -> Sibiu -> Fagaras -> Bucharest

BREADTH-FIRST SEARCH

- Every node generates b successors
- Solution is at depth d
- Total number of nodes generated = $1 + b + b^2 + b^3 + \ldots + b^d$
- Time and Space complexity = $O(b^d)$

- Challenges:

1. Memory requirement
2. Execution time

BREADTH-FIRST SEARCH

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.