COMP 3200
Artificial Intelligence

Lecture 4

Search Strategies

Search Strategies

- Uninformed (Blind) Search
  - No info about states beyond problem desc.
  - Limited to successor generation, goal test

- Informed (Heuristic) Search
  - Can guess which states are 'more promising'
  - Hopefully leads to faster search episodes

Breadth-First Search (BFS)

- Root node expanded first
- Root successors expanded next
- Their successors next... etc
- BFS in general
  - All nodes at a given depth are expanded before any nodes in the next level
- Use a Queue for fringe

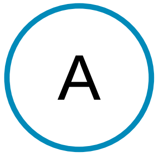
General Uninformed Tree Search

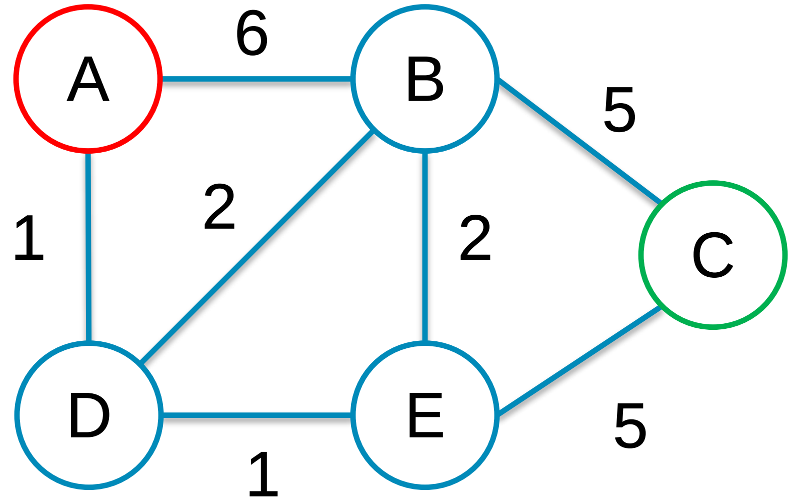
```
Function Tree-Search(problem, strategy)
    fringe = {Node(problem.initial_state)}
    while (true)
        if (fringe.empty) return fail
        node = strategy.select_node(fringe)
        if (node.state is goal) return solution
        else fringe.add(Expand(node, problem))
```

Breadth-First Search

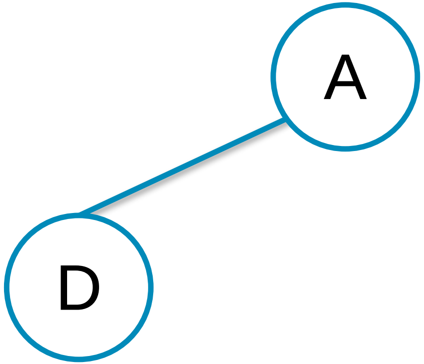
```
Function Tree-Search(problem, BFS)
    fringe = Queue{Node(problem.initial_state)}
    while (true)
        if (fringe.empty) return fail
        node = fringe.pop()
        if (node.state is goal) return solution
        else fringe.push(Expand(node, problem))
```

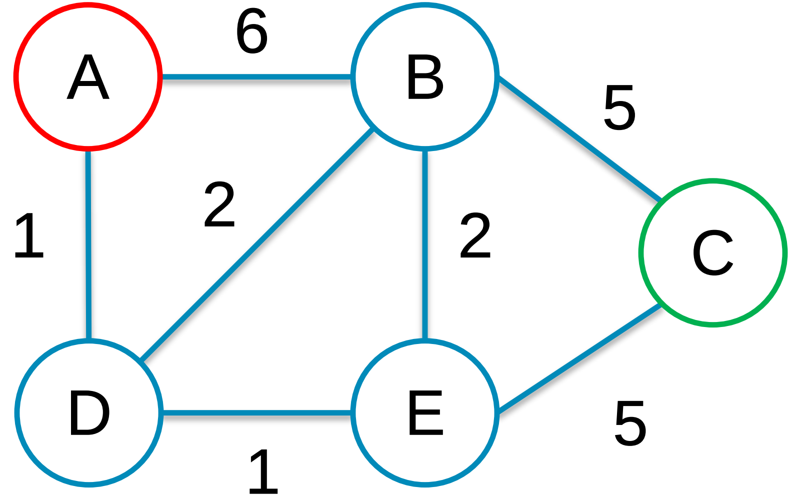
BFS



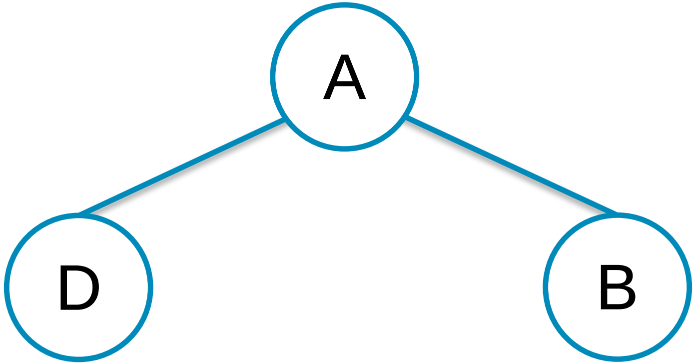


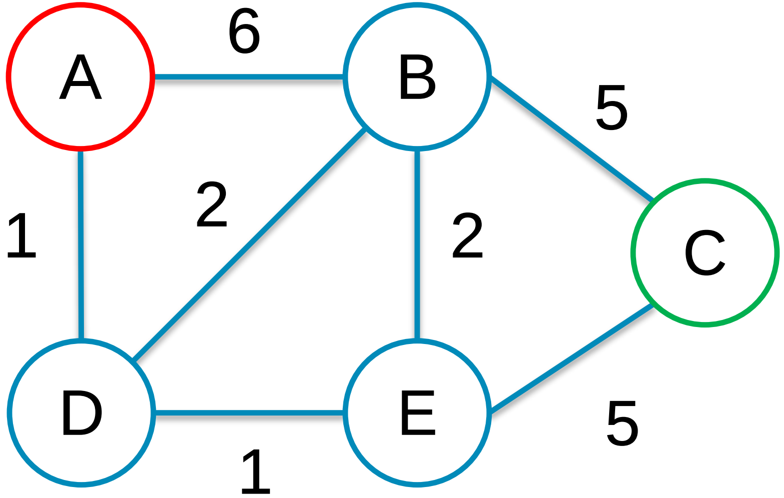
BFS



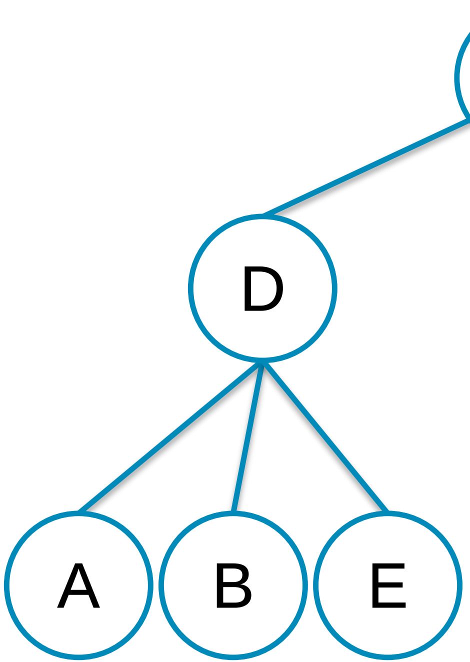


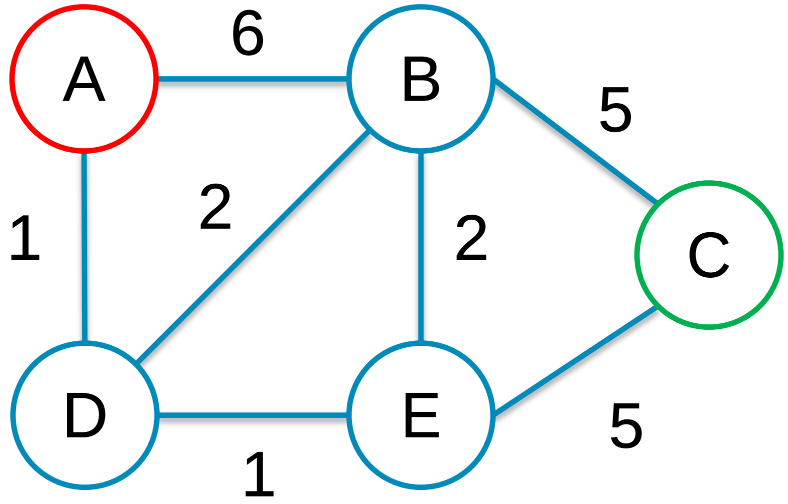
BFS



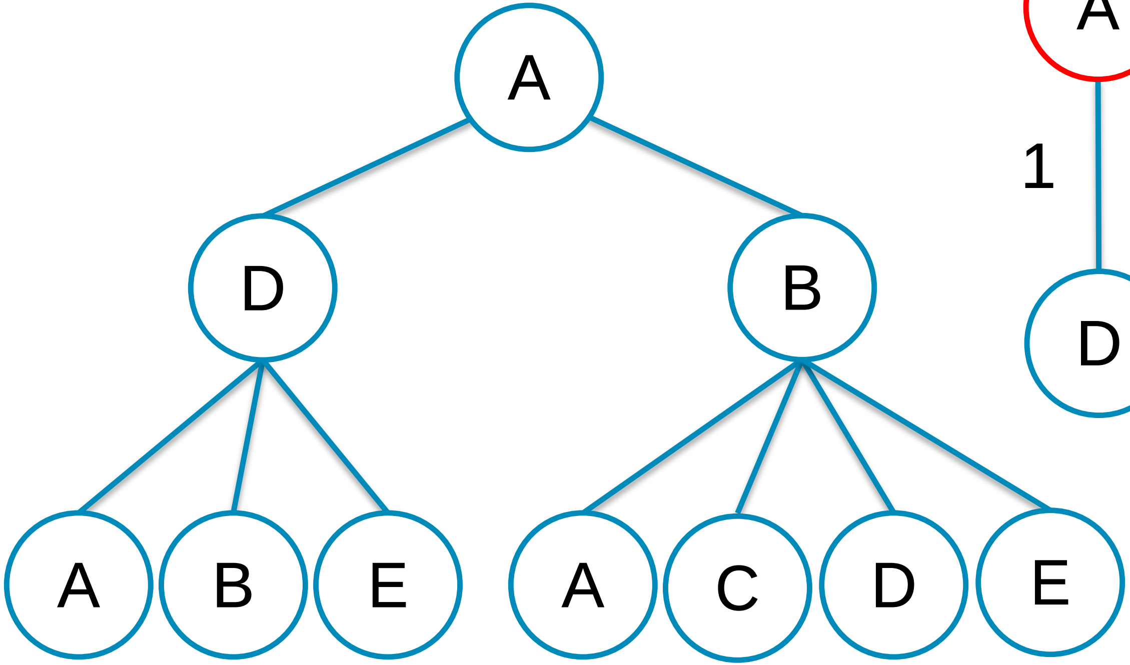


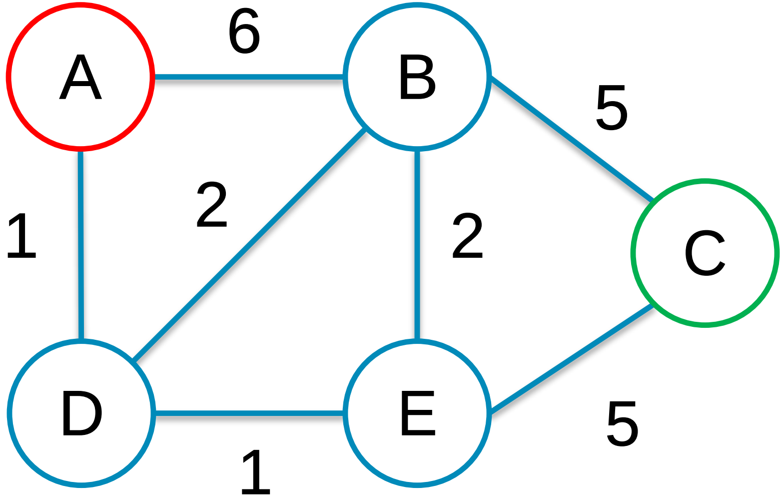
BFS

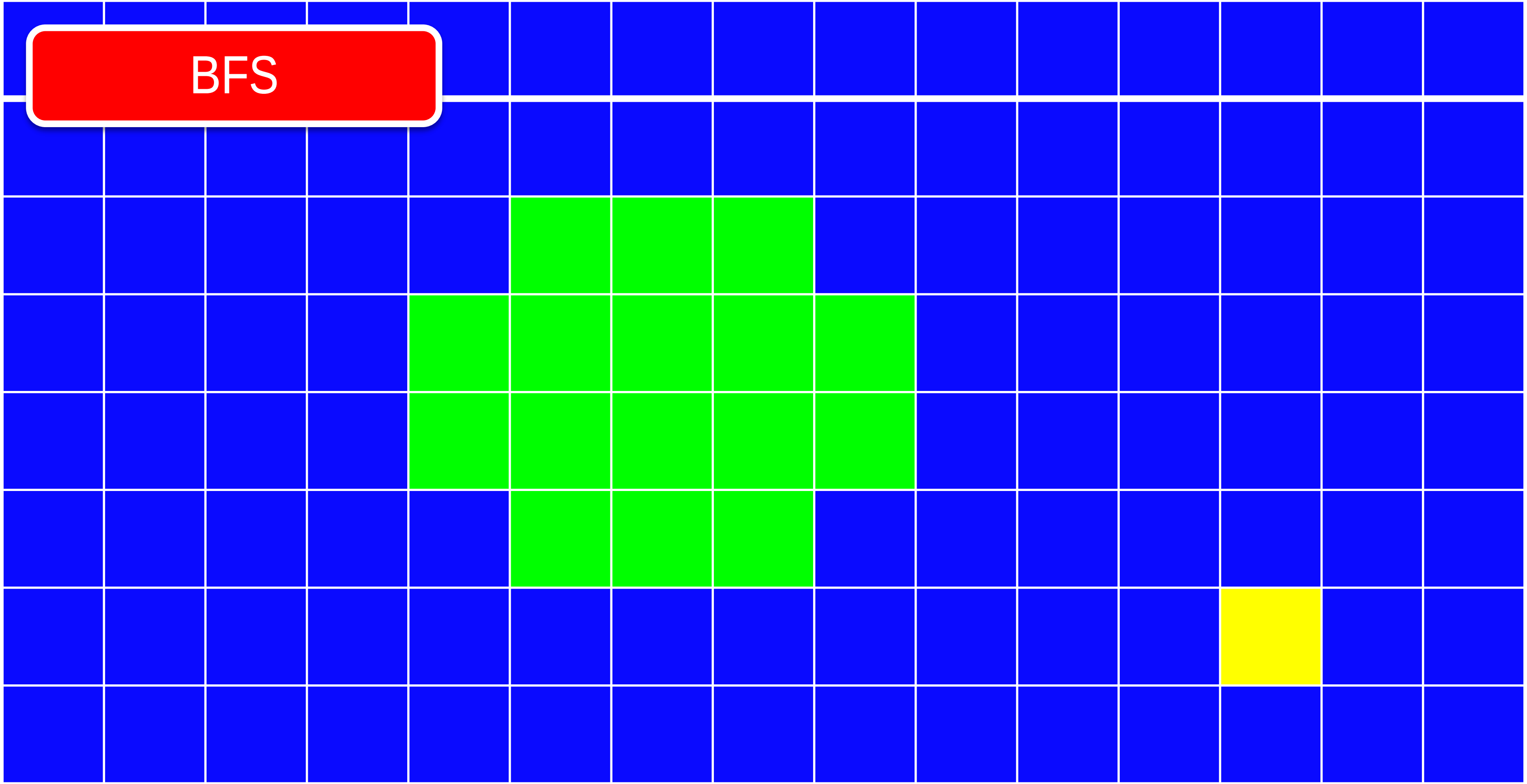


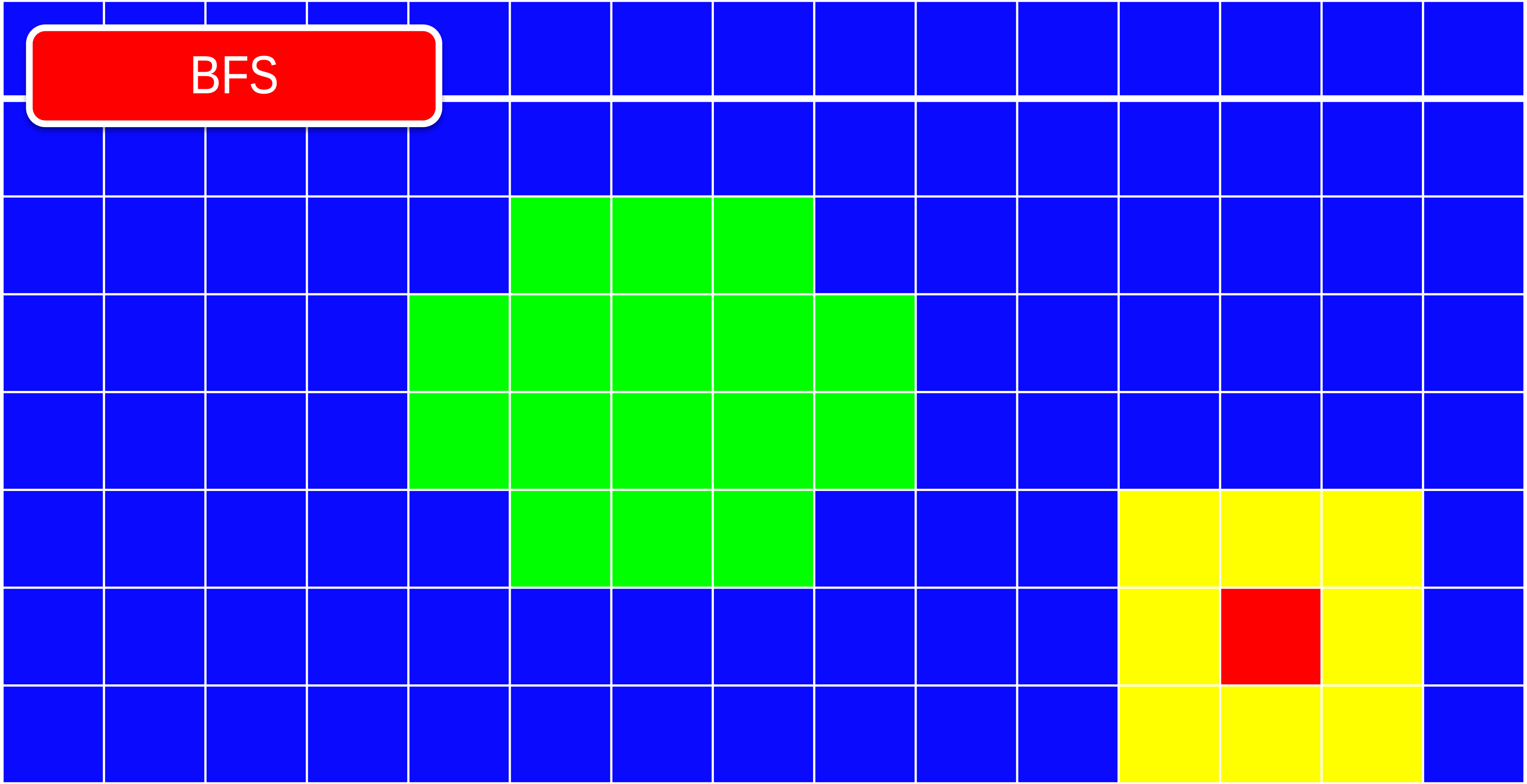


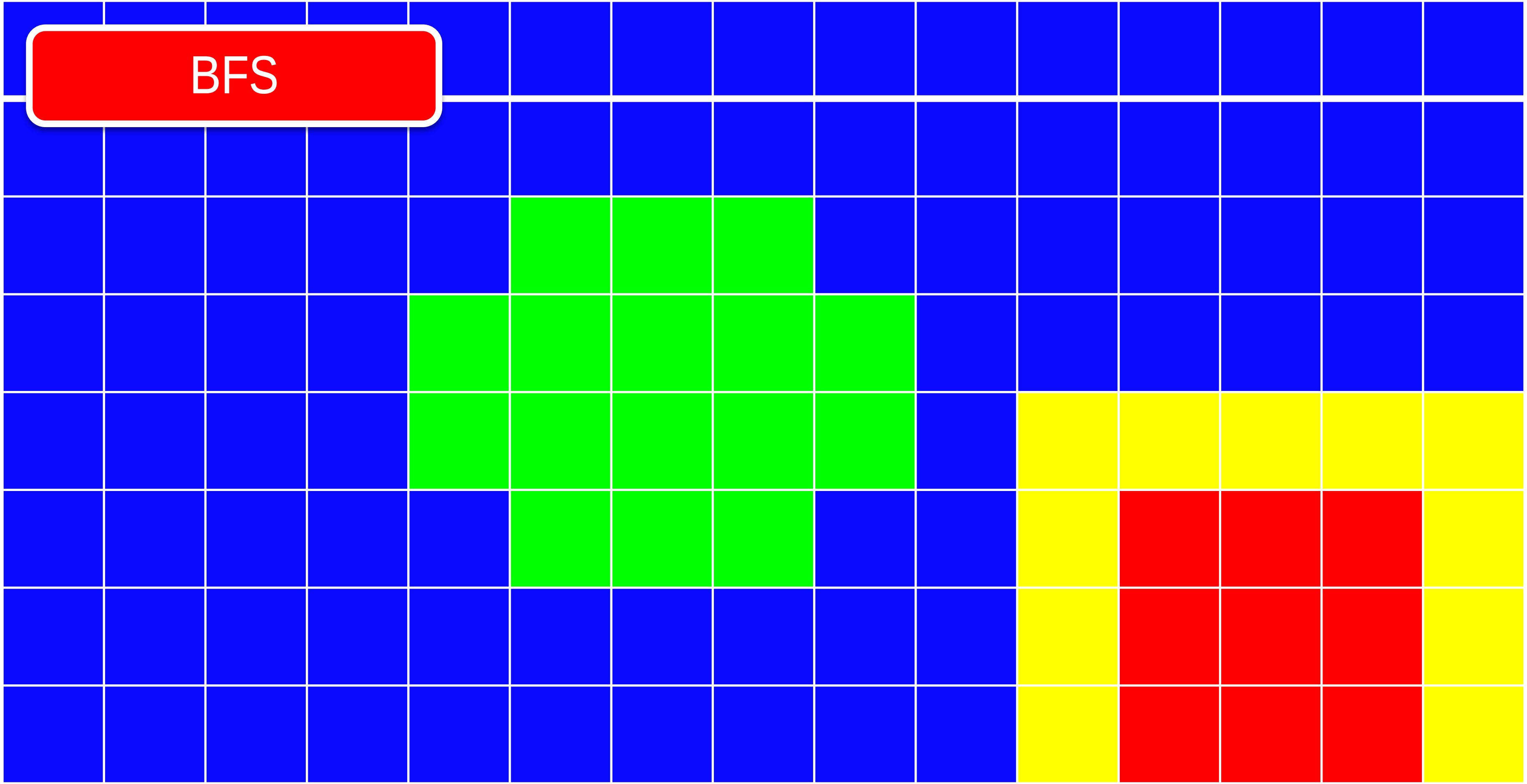
BFS

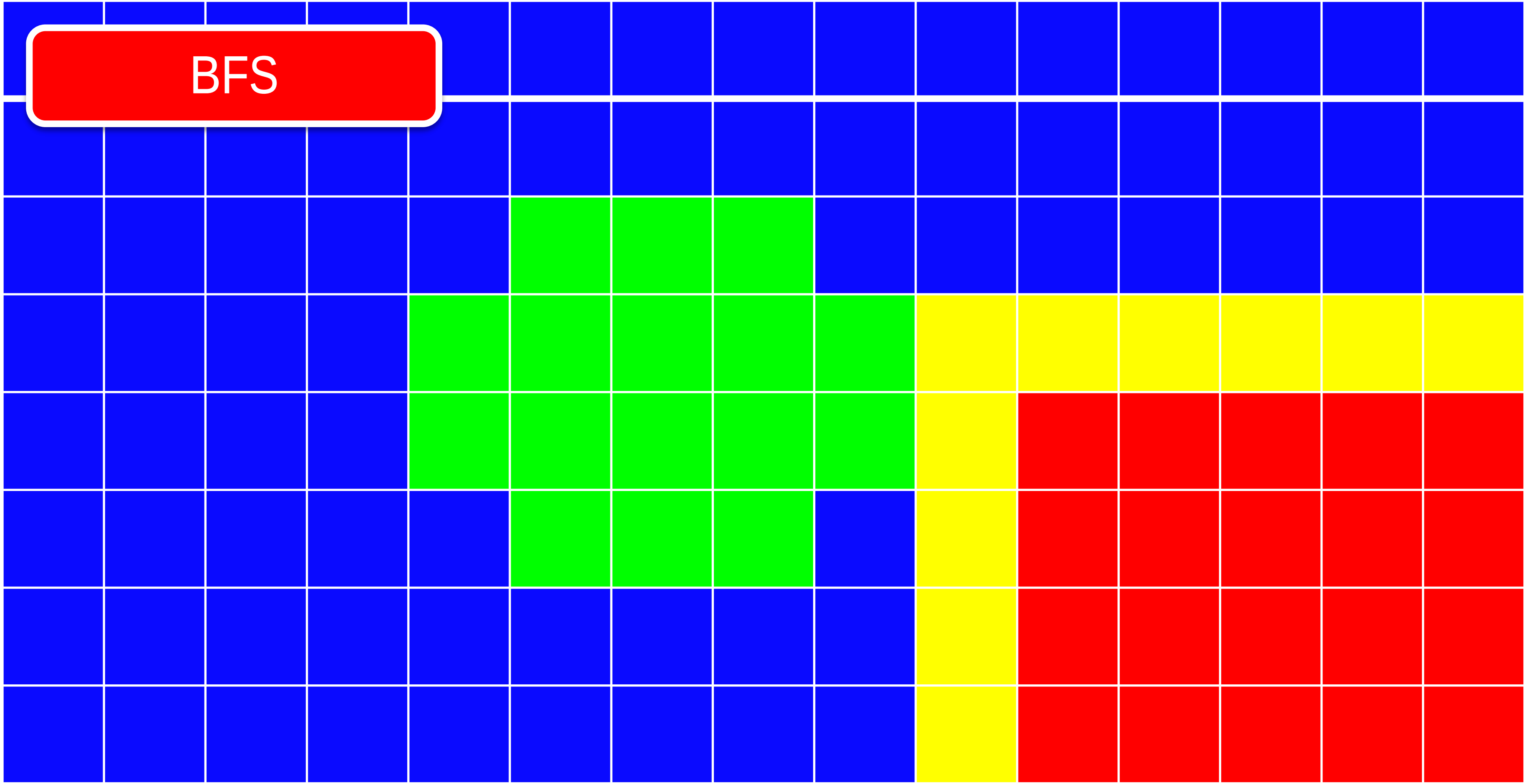


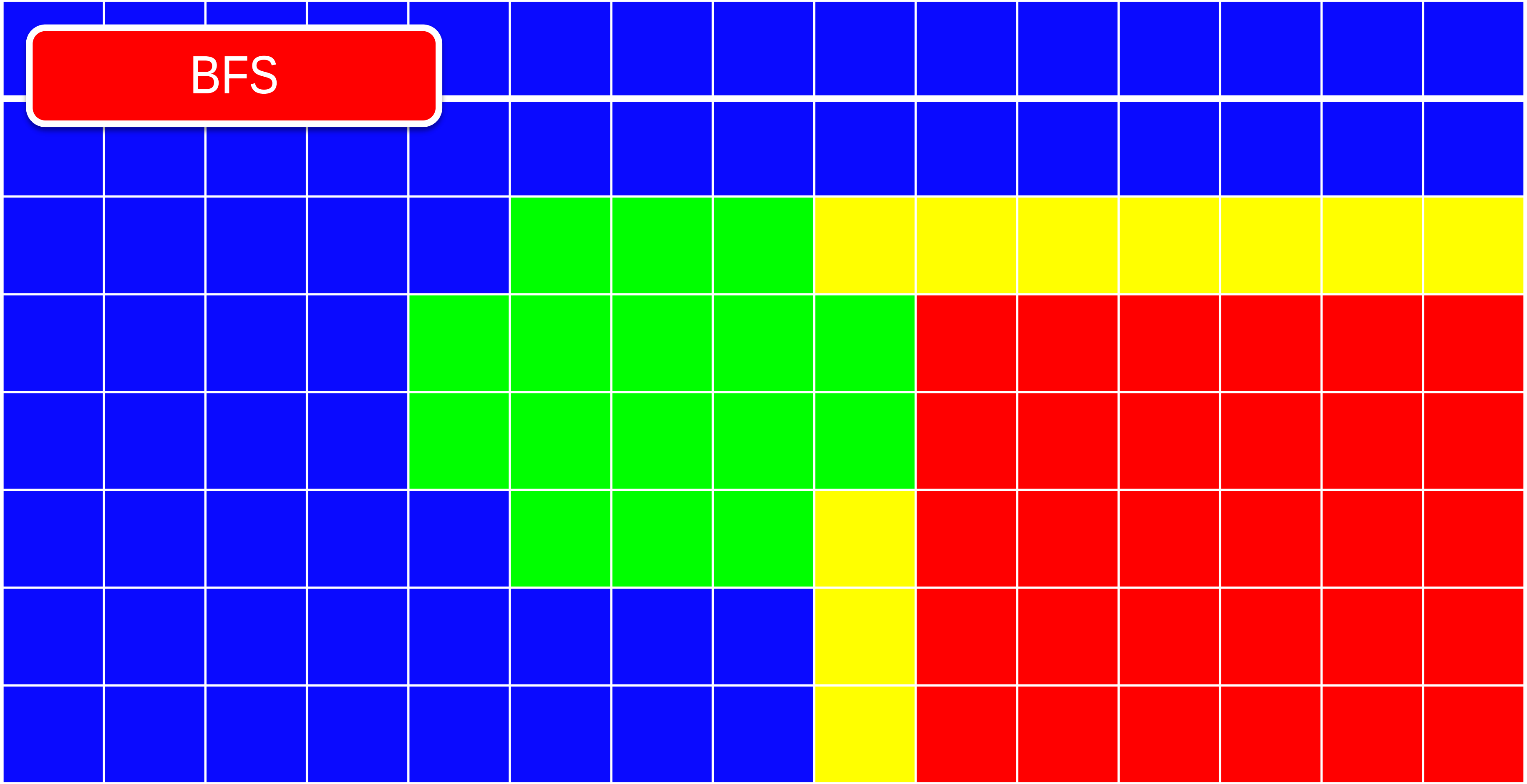


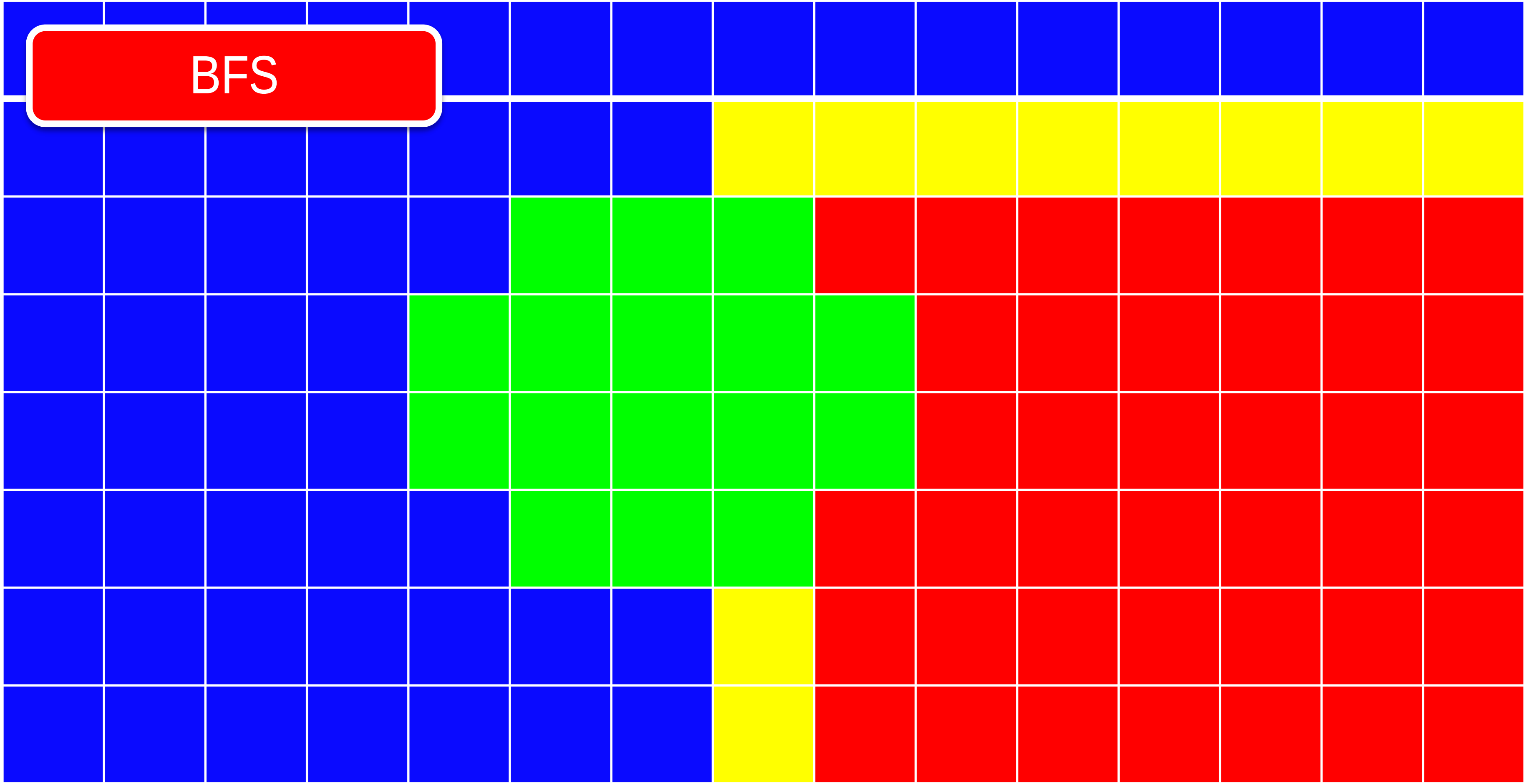

BFS


BFS


BFS


BFS


BFS


BFS

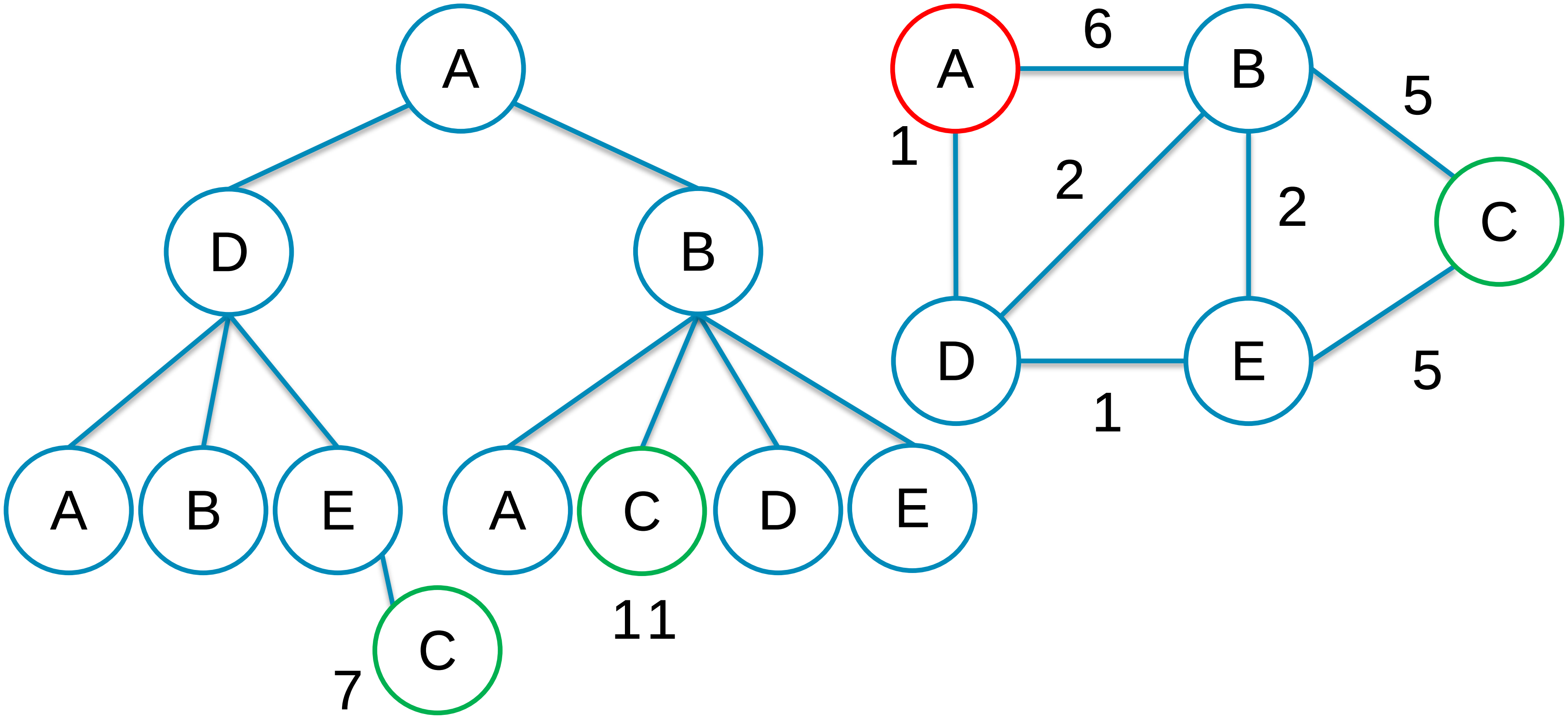
Problem Solving Performance

- Completeness
  - Is it guaranteed to find a solution if it exists
- Optimality
  - Does it find the optimal solution
- Time Complexity
  - How many nodes must it generate to find a soluton
- Space Complexity
  - How much memory is needed to run the search

BFS Performance

- BFS is Complete
  - BFS explores entire tree, so it will find a goal node
  - BFS will find the shallowest goal node, since it explores all nodes at a given depth before the next depth
- BFS is not Optimal
  - In general BFS does not produce optimal solutions, since the shallowest node is not necessarily optimal
  - BFS optimal if path cost is non-decreasing function on depth
  - BFS is optimal if action costs are all the same

Example: BFS Not Optimal



BFS Performance

- Time Complexity $O(b^d)$
  - Each state has b children
  - Consider goal at depth d
  - b * b * b.... = $O(b^d)$ nodes generated
- Space Complexity $O(b^d)$
  - Must store entire search tree in memory
  - Space = nodes generated

BFS Performance Table

Depth	Nodes	Time	Memory
2	1100	.11 seconds	1 MB
4	111,100	11 seconds	106 MB
6	10^7	19 minutes	10 GB
8	10^9	31 hours	1 TB
10	10^{11}	129 days	101 TB
12	10^{13}	35 years	10 PB
14	10^{15}	3523 years	1 EB

b = 10, 10000 nodes/s, 1000bytes / node

Uniform-Cost Search

- BFS optimal when all action costs equal because it expands the shallowest node
- Uniform cost search optimal for any cost
- UCS expands node with lowest path cost
- If all costs are equal, equivalent to BFS
- Works only if all costs > 0
  - Can infinite loop if 0 or negative costs
- UCS is Dijsktra’s Algorithm for a single goal

Uniform Cost Search

```
Function Tree-Search(problem, UCS)
    fringe = {Node(problem.initial_state)}
    while (true)
        if (fringe.empty) return fail
        node = min_g_value(fringe)
        if (node.state is goal) return solution
        else fringe.push(Expand(node, problem))
```

Uniform Cost Search Complexity

- UCS is Complete if b is finite
- UCS is Optimal if action costs > 0 (ε)
- Time Complexity
  - Measured by path costs, not depth
  - Can’t use b and d for complexity
  - Let C* be the cost of optimal solution
  - Complexity = $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$
  - Worst case this can be much bigger than b^d

Depth-First Search (DFS)

- Expands the deepest node on fringe
- Search goes immediately to the deepest level of the search tree to a leaf node
  - Leaf = Node has no successors
- If a leaf is reached, it is discarded and the search 'backs up' to previous depth
- Implementation
  - Use a Stack for fringe
  - More often implemented as recursive function

Depth-First Search

```
Function Tree-Search(problem, DFS)
    fringe = Stack{Node(problem.initial_state)}
    while (true)
        if (fringe.empty) return fail
        node = pop(fringe)
        if (node.state is goal) return solution
        else fringe.push(Expand(node, problem))
```

Recall: BFS

```
Function Tree-Search(problem, BFS)
    fringe = Queue{Node(problem.initial_state)}
    while (true)
        if (fringe.empty) return fail
        node = pop(fringe)
        if (node.state is goal) return solution
        else fringe.push(Expand(node, problem))
```

Recursive DFS Implementation

```
Function DFS(node, problem)
    if (node.state is goal) return solution
    successors = Expand(node, problem)
    for s in successors:
        DFS(s, problem)
```

Call DFS(root_node, problem)

DFS Performance

- DFS is not Complete
  - In general, DFS is not complete
  - For example, can enter infinite loop
  - (Enhancements can make it sort of complete)

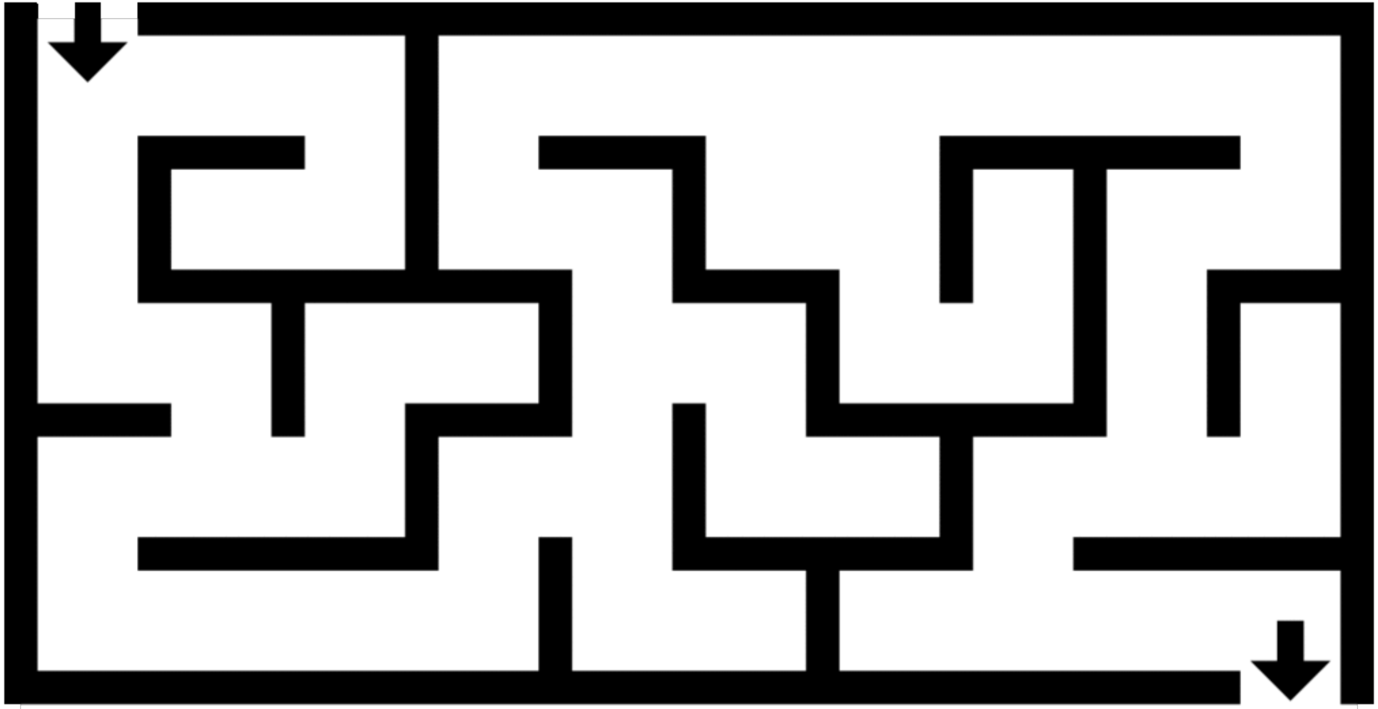
- DFS is not Optimal
  - In general, not optimal
  - Returns the first goal found, could be anywhere in the tree, not guaranteed to be optimal

DFS Performance

- Time Complexity $O(b^m)$
  - May go down long possibly infinite paths
  - May have to generate the entire tree
  - Generates b^m nodes where m is max depth of a node

- Space Complexity $O(bm)$
  - Only has to store the current path at any time
  - Uses $O(bm)$ memory where m is max node depth
  - Stores successors of nodes in the path (b factor)

Depth-First Search



Depth-Limited Search (DLS)

- How to prevent DFS from ‘getting lost’
- Apply a depth limit to the search
  - Choose a depth limit L
  - Enforce nodes at depth L have no successors
- Solves the infinite path problem
  - Will search entire tree up to depth L
- Introduces a new problem
  - Incomplete if solution depth $d > L$
- Time Complexity $O(b^L)$
- Space Complexity $O(bL)$

Recursive DLS

```
Function DLS(node, L, problem)
    if (node.depth > L) return
    if (node.state is goal) return solution
    successors = Expand(node, problem)
    for s in successors:
        DFS(s, problem)
Call DFS(root_node, L, problem)
```

Choosing a Maximum Depth

- Analysis of maximum path length
- Known ‘diameter’ of a search space
- Number of states in problem

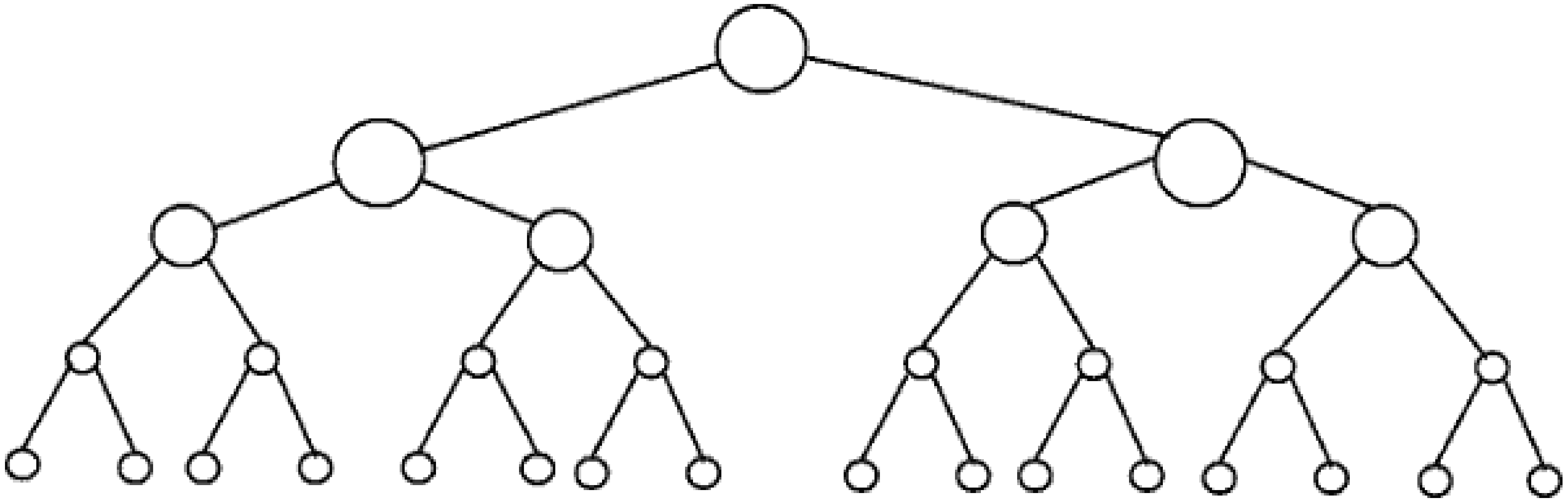
- Let the search figure it out!

Iterative Deepening DFS

- Main Idea: Gradually increase depth limit
  - Try max depth 1, 2, 3, ... M
- Goal will be found at shallowest depth d
  - No solution found at 1... d-1, d = shallowest
- Completeness + Optimality of BFS
  - Guaranteed to find a solution
  - Optimal iff action costs are the same
- Space Complexity of DFS O(bd)

ID-DFS Time Complexity

- May seem wasteful, some re-computation



ID-DFS Time Complexity

- Number of nodes searched
  - $(d)b + (d-1)b^2 + \ldots + 1(b^d)$
  - Time Complexity $O(b^d)$
- Same time complexity as BFS
- Same space complexity as DFS O(bd)
- In practice, generates ~ twice the nodes
- In general, ID-DFS is preferable to BFS

Recall: Recursive DLS

```
Function DLS(node, L, problem)
    cutoff_occurred = false
    if (node.state is goal) return solution
    if (node.depth >= L) return cutoff
    for s in Expand(node, problem):
        result = DLS(s, L, problem)
        if (result == cutoff) cutoff_occurred = true
        else if (result != fail) return result
    return (cutoff_occurred ? cutoff : failure)
```

Call DLS(root_node, L, problem)

Iterative Deepening DFS

```
Function ID-DFS(node, problem)
    for d in (1, infty)
        result = DLS(node, d, problem)
        if (result != cutoff) return result
```

Comparing Search Strategies

Criterion	BFS	UCS	DFS	DLS	ID-DFS
Complete	YES [a]	YES [ab]	NO	NO	YES [a]
Optimal	YES [c]	YES	NO	NO	YES [c]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(b^m)$	$O(b^L)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(bm)$	$O(bL)$	$O(bd)$

(a) if b is finite (b) if action costs > 0 (c) if all costs equal

Comparing Search Strategies

Criterion	BFS	UCS	DFS	DLS	ID-DFS
Complete	YES [a]	YES [ab]	NO	NO	YES [a]
Optimal	YES [c]	YES	NO	NO	YES [c]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(b^m)$	$O(b^L)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(bm)$	$O(bL)$	$O(bd)$

(a) if b is finite (b) if action costs > 0 (c) if all costs equal