States And Uninformed Search

Arnav Gupta

October 17, 2024

Contents

1 Introduction to Search 1

2 Depth-First Search 2
2.1 Properties of DFS 3

3 Breadth-First Search 4
3.1 Multi-Path Pruning 4
3.2 Properties of BFS 4

4 Iterative Deepening Search 5
4.1 Properties of IDS 6

5 Lowest-Cost-First Search 6
5.1 Properties of LCFS 6
5.2 Dijkstra's Algorithm 7

1 Introduction to Search

Solving some problems requires searching for a solution by searching a directed graph that is the **state space** of the problem. A graph consists of a set of nodes and a set of arcs/edges. Nodes represent **states** and edges represent legal **state transitions**.

A **search problem** is defined by:

- a set of states
- an initial state

- goal states or a goal test (function that tells whether a given state is a goal state)
- a successor (neighbour) function (action from one state to other states)
- optionally, a cost associated with each action

A solution to the search problem is a path from the start state to a goal state, possibly optimizing for cost.

Graph searching involves maintaining a **frontier** of paths from the start node that have been explored and expanding this into unexplored nodes until the goal node is encountered.

Search strategy: the way the frontier is expanded

Search algorithms can return multiple answers as well.

Types of search:

- uninformed (blind)
- heuristic
- sophisticated hacks

Properties of search algorithms:

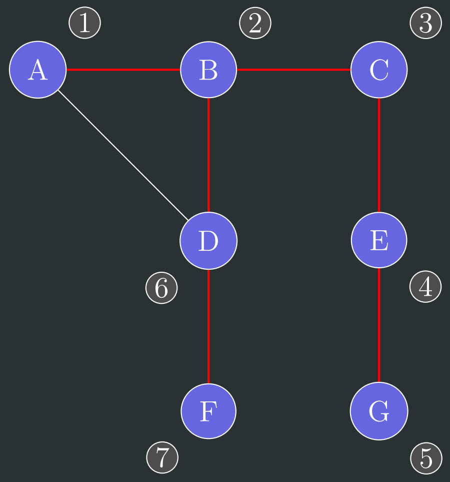
- *space complexity*: size of frontier in worst case
- *time complexity*: number of nodes visited in worst case
- *completeness*: if a solution exists, is it found?
- *optimality*: if a solution is found, is it the least cost solution?

Useful quantities:

- b: branching factor (max number of children of any node)
- m: maximum depth of the search tree
- d: depth of the shallowest goal node

2 Depth-First Search

Treat the frontier as a stack, select the last element added to the frontier to be expanded.



Cycle checking: prune a path that ends in a node already on the path

DFS can be implemented recursively, with cycle checking. With this implementation, the frontier is implicitly stored in the call stack.

2.1 Properties of DFS

Properties:

- *space complexity*: $O(bm)$
- *time complexity*: $O(b^m)$
- *completeness*: no, will get stuck in infinite path (may or may not be cycle)
- *optimality*: no, pays no attention to cost

DFS is useful when:

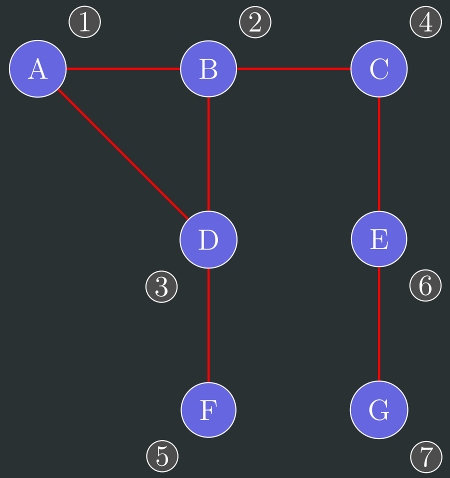
- space is restricted
- many solutions exist, possibly with long paths

DFS is poor when:

- infinite paths exist
- optimal solutions are shallow
- multiple paths to a node

3 Breadth-First Search

Treats the frontier as a queue, select the earliest element added to the frontier to be expanded.



3.1 Multi-Path Pruning

Prune a path to a given node than any path has been found to, since a path has already been found to it.

Pruning subsumes a cycle check, since the current path is a path to the node.

Requires storing all nodes paths have been found to, and must guarantee that this allows optimality.

3.2 Properties of BFS

Properties:

- *space complexity*: $O(b^d)$
- *time complexity*: $O(b^d)$
- *completeness*: yes, since it explores the tree level by level until it finds a goal
- *optimality*: no, it is guaranteed to find the shallowest goal node

BFS is useful when:

- space is no concern

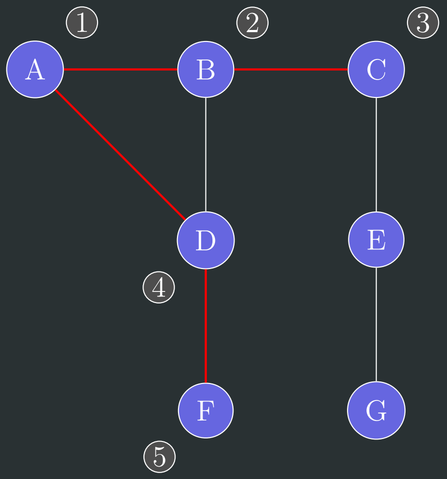
- a solution with the fewest arcs is desirable

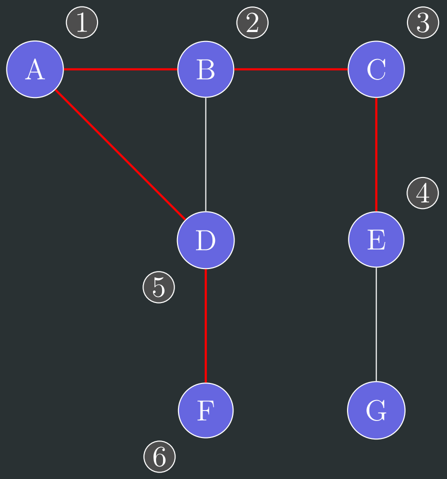
BFS is a poor method when:

- all solutions are deep in the tree
- problem is large and graph is dynamically generated

4 Iterative Deepening Search

For every depth limit, perform DFS until the depth limit is reached.





4.1 Properties of IDS

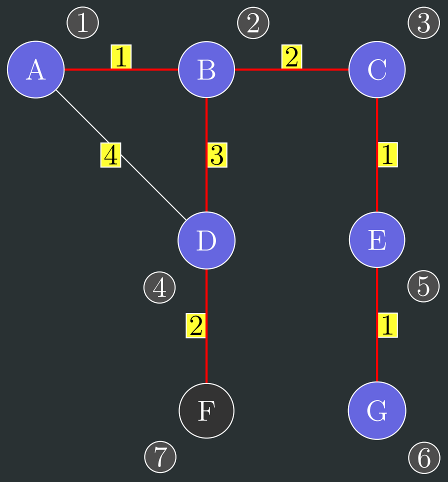
Properties:

- *space complexity*: $O(bd)$
- *time complexity*: $O(b^d)$
- *completeness*: yes, since it explores the tree level by level until it finds a goal
- *optimality*: no, it is guaranteed to find the shallowest goal node

5 Lowest-Cost-First Search

Selects a path on the frontier with lowest cost, where the frontier is a priority queue ordered by path cost.

It is an uninformed/blind search (does not take the goal into account).



5.1 Properties of LCFS

Properties:

- *space complexity*: exponential
- *time complexity*: exponential
- *completeness*: yes
- *optimality*: yes

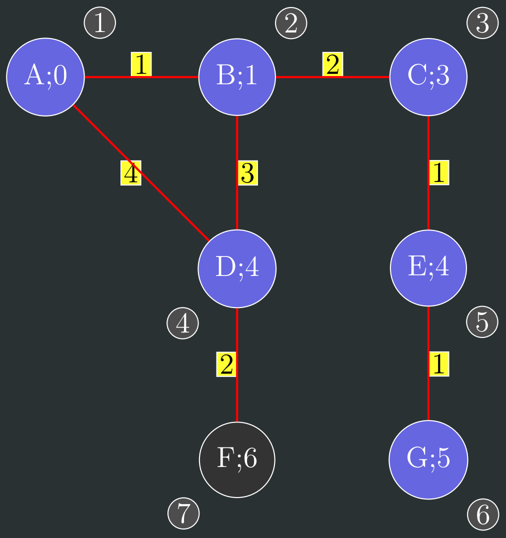
Completeness and optimality require that:

1. the branching factor is finite
2. the cost of every edge is strictly positive

5.2 Dijkstra's Algorithm

Variant of LCFS with multi-path pruning.

Frontier is a PQ sorted by cost, with each node keeping track



A;0
B;1
C;3
D;4
E;4
F;6
G;5
1
2
3
4
5
6
7
1
2
4
3
1
2
1