Grokking Artificial intelligence Algorithms
by Rishal Hurbans

Section 2 Search Fundamentals

When we think about what makes us intelligent, the ability to plan before carrying out actions is a prominent attribute. Before embarking on a trip to a different country, before starting a new project, before writing functions in code, planning happens. Planning happens at different levels of detail in different contexts to strive for the best possible outcome when carrying out the tasks involved in accomplishing goals

Planning

Planning is not always going to work and one must be conscious of the fact that the journey to the goal may change when the environment changes. One must evaluate future states and evaluate the best course of action given the current environment.

Cost of Computation

Due to how traditional computers operate, all functions in computers take a certain amount of time to process. To describe the complexity of algorithms and functions we can utilize Big O notation which models the number of operations required as the input size increases. Any single operation that runs once is given the big O notation of O(1) while a function that iterates n times has a cost of O(n). Any function that compares all the items in a list with other items in the same list has a cost of O(n^2). The closer algorithms are to O(1) the more efficient the algorithm is. This is an important concept to know since any problem can be solved through brute force and a low cost function would benefit the speed of computation.

Frameworks to represent Spaces

Data Structures are very important for representing information in a way that an algorithm can easily understand it. This data structure can be organized and influenced by the context of the problem. One such data structure is the array which is simply a collection of values, usually represented as a linear string of values. A graph which consists of many states called nodes (or vertices) with connections between them known as edges. Edges may exist just as ways to connect nodes or may have weights associated with them to represent things such as distance or simply represent the cost of moving from one state to the other. Graphs can be represented using an array of arrays to indicate the relationship between nodes. Trees are another form of data structures used to simulate hierarchy of values or objects. Trees differs from graphs in that trees do not cycle and a node may only have an input edge and at most 2 output edges. Trees are made up of the root node that serves as the parent node with connections to all other nodes known as the child nodes. Note that subtrees exist within the main tree. Height refers to the total height of the tree while depth refers to the height at a certain level of the tree. Nodes with the same parent node are siblings, while nodes with no children are known as leaf nodes.

Types of searches

One type of search that algorithms are used for is uninformed search which has no other information apart from how its represented (usually a tree). This is where breadth first search (BFS) and depth first search(DFS) are commonly used. The BFS algorithm is used to explore all options at a specific depth before moving on to the other options at greater depths. DFS works explores a specific path from the start until it reaches its goal at the utmost depth. (DFS does not work great with weighted graphs since it doesn check all children of the root node first, results may differ depending on how the graph is formed(from personal experience in assignment 1))

Graph types

Undirected graphs essentially entail graphs where edges connecting nodes go both ways and the relationships are mutual. As for Directed graphs these are graphs where flow is in one direction between nodes. Where the child node cannot also be the parent node of its own parent as seen in undirected graphs. In Disconnected graphs in which one or more nodes is not connected by edges to any other nodes. Then there are Complete bipartite graph in which every node is connected to every other node of the other partition through an edge. And finally Weighted graphs where edges between nodes have weights and can influence how algorithms choose a path over another.

Important Figures:

Time Complexity

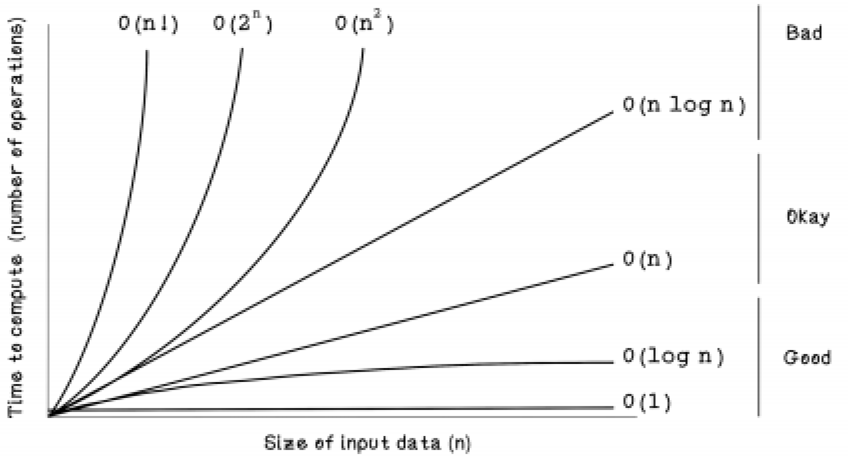


Figure 2.3 Big O complexity

Data vs. Information

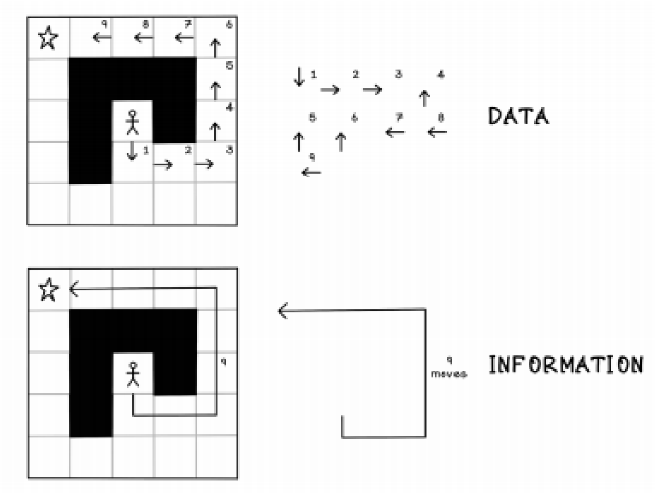


Figure 2.7 Data versus information

Data Structures

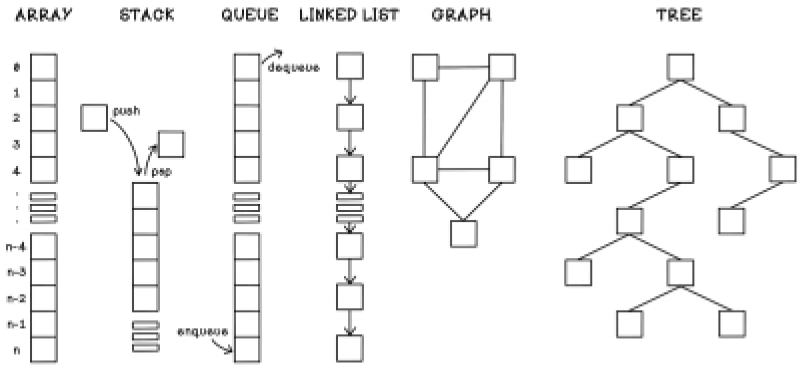


Figure 2.8 Data structures used with algorithms

Tree Attributes

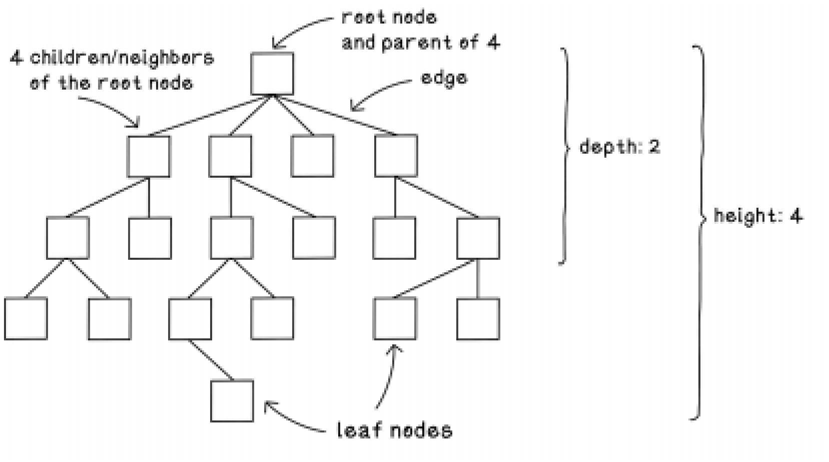
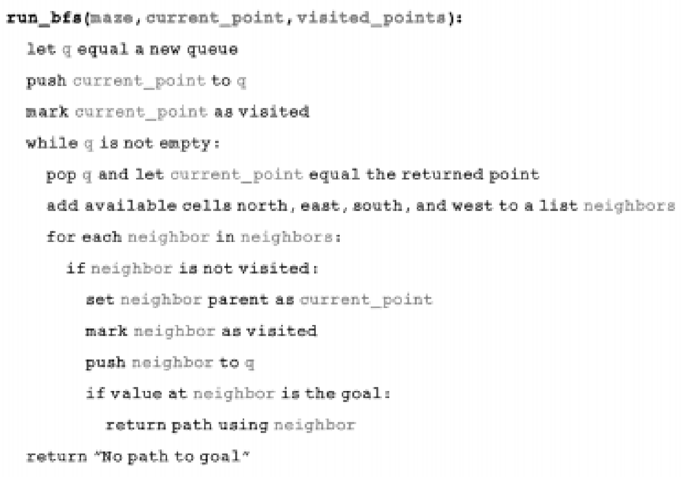


Figure 2.12 The main attributes of a tree

Breadth First Search pseudocode



```
run_bfs(maze, current_point, visited_points):
  let q equal a new queue
  push current_point to q
  mark current_point as visited
  while q is not empty:
    pop q and let current_point equal the returned point
    add available cells north, east, south, and west to a list neighbors
    for each neighbor in neighbors:
      if neighbor is not visited:
        set neighbor parent as current_point
        mark neighbor as visited
        push neighbor to q
        if value at neighbor is the goal:
          return path using neighbor
  return "No path to goal"
```

Breadth Search First Algorithm

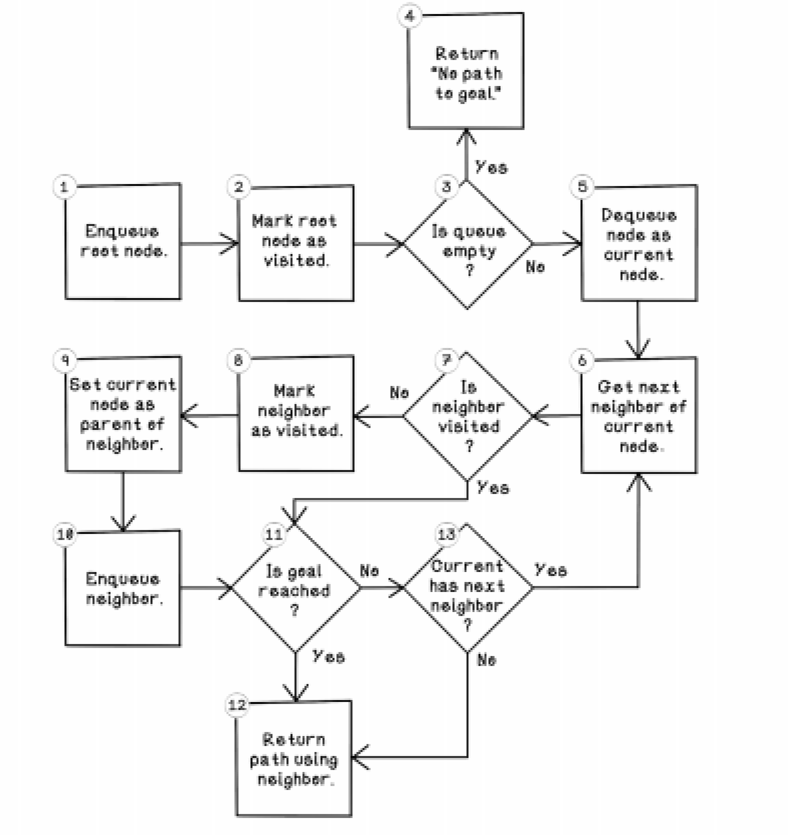
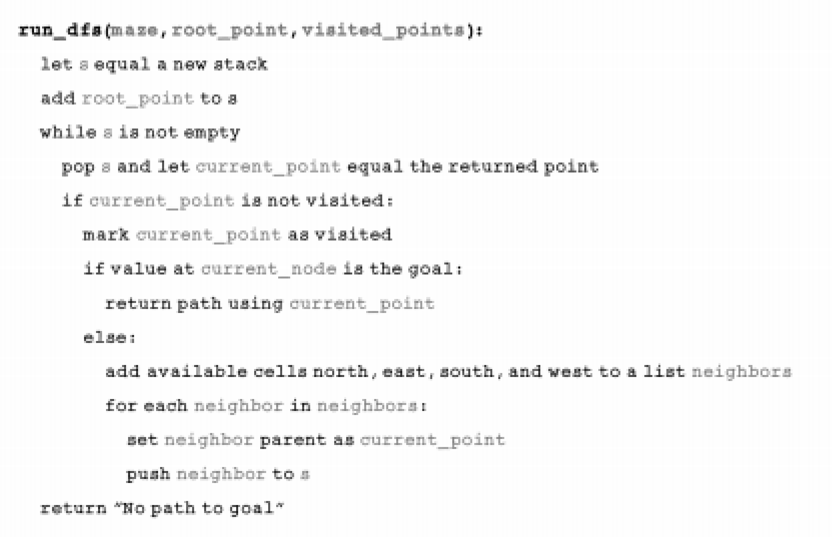

1
Enqueue root node.
2
Mark root node as visited.
3
Is queue empty ?
Yes
No
4
Return "No path to goal."
5
Dequeue node as current node.
6
Get next neighbor of current node.
7
Is neighbor visited ?
No
Yes
8
Mark neighbor as visited.
9
Set current node as parent of neighbor.
10
Enqueue neighbor.
11
Is goal reached ?
No
Yes
13
Current has next neighbor ?
Yes
No
12
Return path using neighbor.

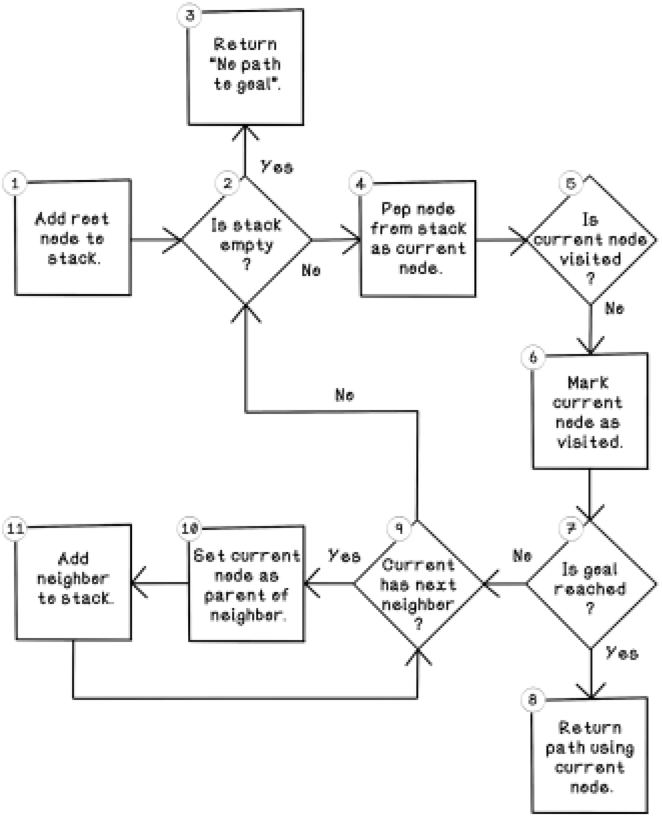
Figure 2.16 Flow of the breadth-first search algorithm

Depth First Search Pseudocode



```
run_dfs(maze, root_point, visited_points):
  let s equal a new stack
  add root_point to s
  while s is not empty
    pop s and let current_point equal the returned point
    if current_point is not visited:
      mark current_point as visited
      if value at current_node is the goal:
        return path using current_point
      else:
        add available cells north, east, south, and west to a list neighbors
        for each neighbor in neighbors:
          set neighbor parent as current_point
          push neighbor to s
  return "No path to goal"
```

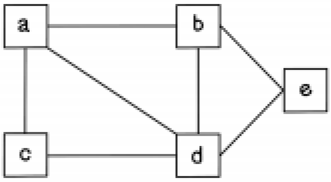
Depth Search First Algorithm


3
Return "No path to goal".
Yes
1
Add root node to stack.
2
Is stack empty ?
No
4
Pop node from stack as current node.
5
Is current node visited ?
No
6
Mark current node as visited.
No
11
Add neighbor to stack.
10
Set current node as parent of neighbor.
Yes
9
Current has next neighbor ?
No
7
Is goal reached ?
Yes
8
Return path using current node.

Types of Graphs

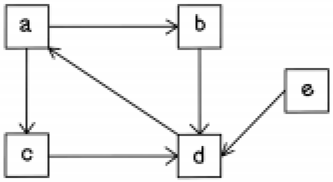
UNDIRECTED

No edges are directed.
Relationships between two
nodes are mutual.



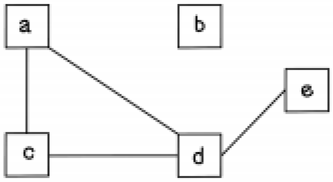
DIRECTED

Edges indicate direction.
Relationships between two
nodes are explicit.



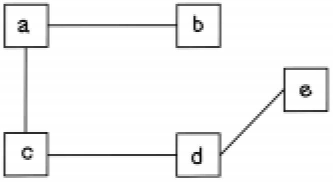
DISCONNECTED

One or more nodes are not
connected by any edges.



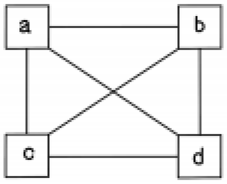
ACYCLIC

A graph that contains
no cycles.



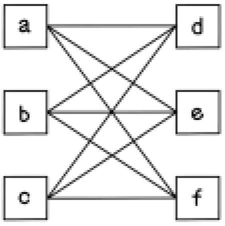
COMPLETE

Every node is connected
to every other node by an
edge.



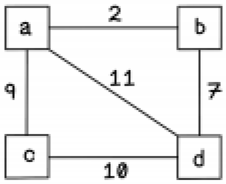
COMPLETE BIPARTITE

Every node from one partition is
connected to every node of the
other partition.



WEIGHTED

A graph where the
edges between nodes
have a weight.


a
2
b
11
9
7
c
10
d