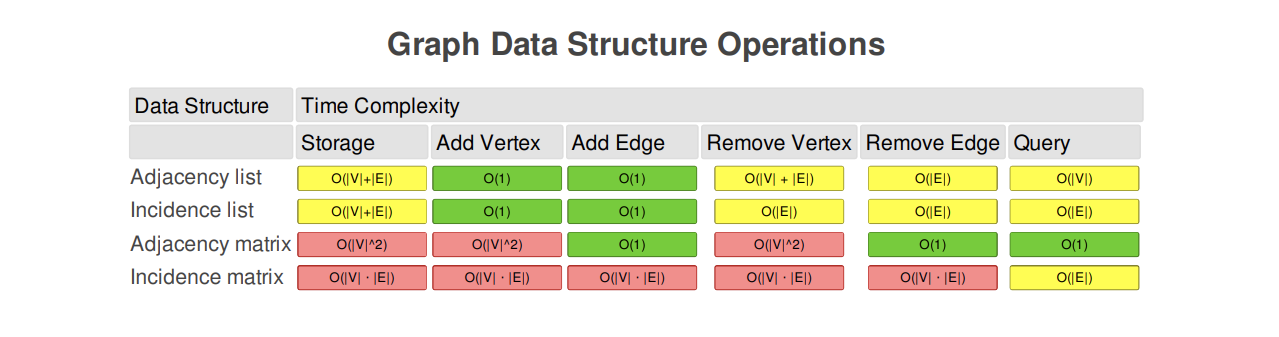
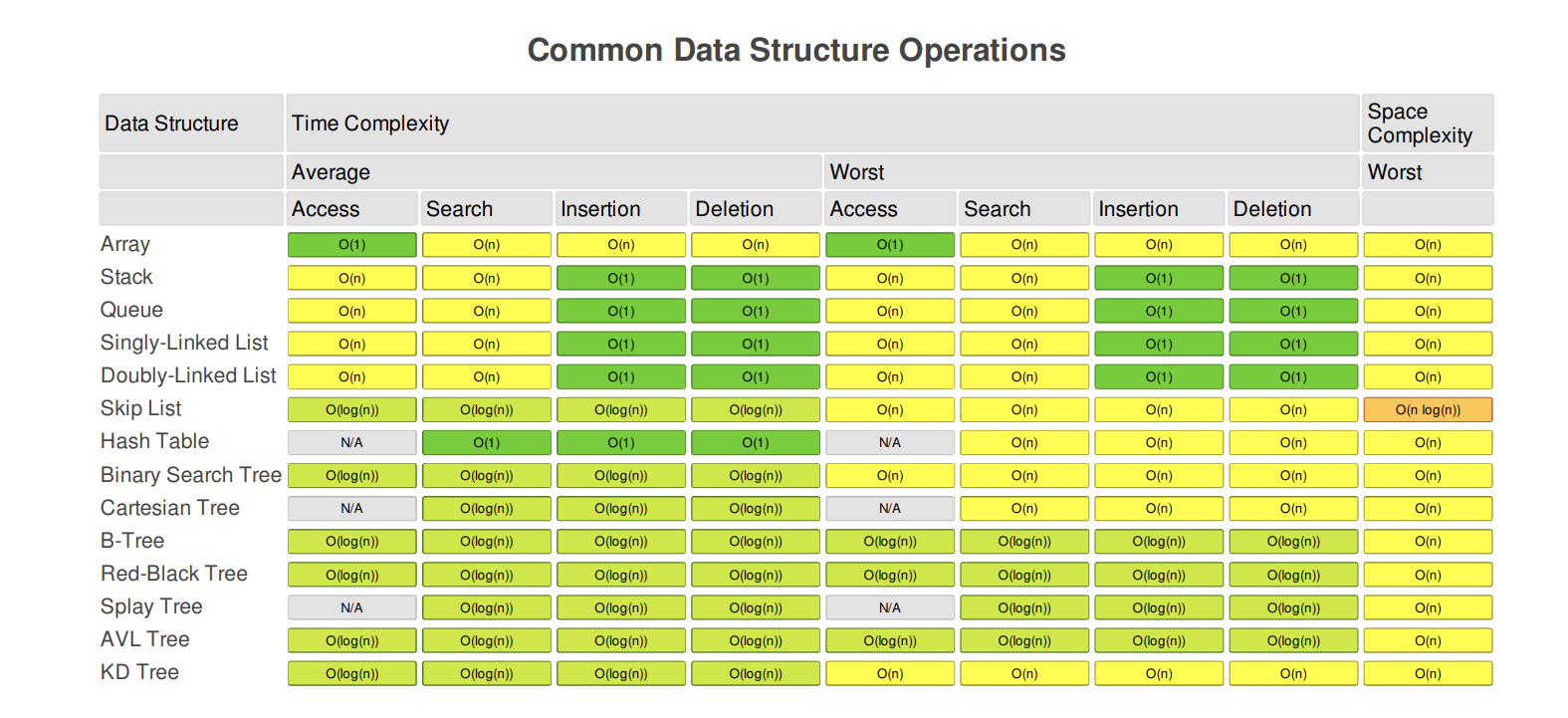
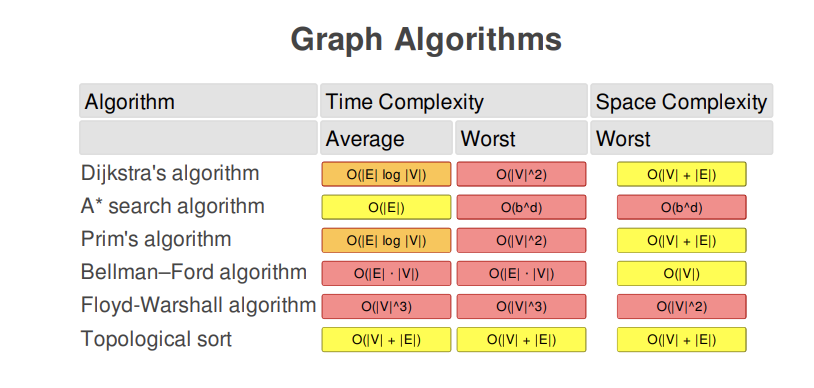
# Help Sheet

Time Complexities





Chart

Description automatically generated

Basic Concepts

Graph

* + Each graph element is called a *node*, or *vertex*.
  + Graph nodes are connected by *edges*.
  + Graphs can be *directed* or *undirected*.
  + Graphs can be *cyclic* or *acyclic*. A cyclic graph contains a path from at least one node back to itself.
  + Graphs can be *weighted* or *unweighted*. In a weighted graph, each edge has a certain numerical weight.
  + Graphs can be *traversed*. See important facts under *Tree* for an overview of traversal algorithms.
  + Data structures used to represent graphs:
    - *Edge list*: a list of all graph edges represented by pairs of nodes that these edges connect.
    - *Adjacency list*: a list or hash table where a key represents a node and its value represents the list of this node's neighbors.
    - *Adjacency matrix*: a matrix of binary values indicating whether any two nodes are connected.
* **Pros**:
  + Ideal for representing entities interconnected with links.
* **Cons**:
  + Low performance makes scaling hard.
* **Notable uses**:
  + Social media networks.
  + Recommendations in ecommerce websites.
  + Mapping services.
* **Time complexity** (worst case): varies depending on the choice of algorithm. O(n\*lg(n)) or slower for most graph algorithms.

Adjacency Matrix

**Assuming the graph has n vertices, the time complexity to build such a matrix is O(n^2). The space complexity is also O(n^2).** Given a graph, to build the adjacency matrix, we need to create a square n\*n matrix and fill its values with 0 and 1. It costs us O(n^2) space.

**The advantage of such representation is that we can check in O(1)  time if there exists edge e(ij) = (vi, vj) by simply checking the value at i row and j column of our matrix.**

Adjacency List

**If e is the number of edges in a graph, then the time complexity of building such a list is O(e). The space complexity is O(n+e).** But, in the worst case of a complete graph, which contains n/2 edges, the time and space complexities reduce to O(n^2).

As it was mentioned, complete graphs are rarely meet. **Thus, this representation is more efficient if space matters.** Moreover, we may notice, that the amount of edges doesn’t play any role in the space complexity of the adjacency matrix, which is fixed. But, the fewer edges we have in our graph the less space it takes to build an adjacency list.

**Therefore, the time complexity checking the presence of an edge in the adjacency list is O(n).** Let’s assume that an algorithm often requires checking the presence of an arbitrary edge in a graph. Also, time matters to us. Here, using an adjacency list would be inefficient.

Trees

* + Organizes values hierarchically.
  + A tree item is called a *node*, and every node is connected to 0 or more child nodes using *links*.
  + A tree is a kind of graph where between any two nodes, there is only one possible path.
  + The top node in a tree that has no parent nodes is called the *root*.
  + Nodes that have no children are called *leaves*.
  + The number of links from the root to a node is called that node's *depth*.
  + The height of a tree is the number of links from its root to the furthest leaf.
  + In a *binary tree*, nodes cannot have more than two children.
    - Any node can have one left and one right child node.
    - Used to make *binary search trees*.
    - In an unbalanced binary tree, there is a significant difference in height between subtrees.
    - An completely one-sided tree is called a *degenerate tree* and becomes equivalent to a linked list.
  + Trees (and graphs in general) can be *traversed* in several ways:
    - *Breadth first search* (BFS): nodes one link away from the root are visited first, then nodes two links away, etc. BFS finds the shortest path between the starting node and any other reachable node.
    - *Depth first search* (DFS): nodes are visited as deep as possible down the leftmost path, then by the next path to the right, etc. This method is less memory intensive than BFS. It comes in several flavors, including:
      * *Pre order traversal* (similar to DFS): after the current node, the left subtree is visited, then the right subtree.
      * *In order traversal*: the left subtree is visited first, then the current node, then the right subtree.
      * *Post order traversal*. the left subtree is visited first, then the right subtree, and finally the current node.
* **Pros**:
  + For most operations, the average time complexity is O(log(n)), which enables solid scalability. Worst time complexity varies between O(log(n)) and O(n).
* **Cons**:
  + Performance degrades as trees lose balance, and re-balancing requires effort.
  + No constant time operations: trees are *moderately* fast at everything they do.
* **Notable uses**:
  + File systems.
  + Database indexing.
  + Syntax trees.
  + Comment threads.
* **Time complexity**: varies for different kinds of trees.

Binary Search Tree

* + Nodes of a binary search tree (BST) are ordered in a specific way:
    - All nodes to the left of the current node are smaller (or sometimes smaller or equal) than the current node.
    - All nodes to the right of the current node are larger than the current node.
  + Duplicate nodes are usually not allowed.
* **Pros**:
  + Balanced BSTs are moderately performant for all operations.
  + Since BST is sorted, reading its nodes in sorted order can be done in O(n), and search for a node closest to a value can be done in O(log(n))
* **Cons**:
  + Same as trees in general: no constant time operations, performance degradation in unbalanced trees.
* **Time complexity** (worst case):
  + Access: O(n)
  + Search: O(n)
  + Insertion: O(n)
  + Deletion: O(n)
* **Time complexity** (average case):
  + Access: O(log(n))
  + Search: O(log(n))
  + Insertion: O(log(n))
  + Deletion: O(log(n))

Applications of Graph

Following  are some applications of graphs in data structures:

* Graphs are used in computer science to depict the flow of computation.
* Users on Facebook are referred to as vertices, and if they are friends, there is an edge connecting them. The Friend Suggestion system on Facebook is based on graph theory.
* You come across the Resource Allocation Graph in the Operating System, where each process and resource are regarded vertically. Edges are drawn from resources to assigned functions or from the requesting process to the desired resource. A stalemate will develop if this results in the establishment of a cycle.
* Web pages are referred to as vertices on the World Wide Web. Suppose there is a link from page A to page B that can represent an edge. This application is an illustration of a directed graph.
* Graph transformation systems manipulate graphs in memory using rules. Graph databases store and query graph-structured data in a transaction-safe, permanent manner.