# FIT1045 – Glossary (S1/19)

## Algorithms

**Algorithm** An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.

**Assertion** An assertion is a logical statement on a program (execution) state.

**Big-O Notation** A function $f(n)$ is in $O(g(n))$ (read "$f$ is in big oh of $g$") if there are positive numbers $c$ and $n_0$ such that for all $n \geq n_0$ it holds that $t(n) \leq cg(n)$.

**Computational complexity** The (best-case/worst-case) computational (time) complexity of an algorithm is the number of elementary steps $T(n)$ needed for computing its output for an input of a size $n$ (in the best/worst case).

**Decision Problem** A computational problem is called a decision problem if the required output for each input is Boolean (yes or no). Inputs for which output is yes (True) are a called yes-input. Inputs for which output is no (False) are a called no-input.

**Invariant** An assertion INV is a loop invariant for a loop if the loop initialisation will yield a state in which INV holds and every execution of the loop body yields a state in which INV holds again (if run in any state in which INV holds and the loop exit condition does not hold).

**Lexicographic order** Let $x$ and $y$ be two sequences containing mutually comparable elements. Then we call $x$ lexicographically smaller than $y$ if either a) up to some position $i$ (exclusive) all elements of $x$ and $y$ are equal and $x[i] < y[i]$ or b) if $x$ is a proper prefix of $y$, i.e., $x$ is shorter than $y$ and the leading positions of $y$ contain exactly the same element as $x$.

**NP, complexity class** The class NP is the class of decision problems $D$ that have a polynomial time verification algorithm $A$. That is, $A$ accepts as inputs pairs of $D$-inputs $x$ and short certificates $c$ (the length of $c$ must be polynomially bounded in the length of $x$), and for each $D$-input $x$ there is a certificate $c$ such that $A(x, c) =$ True if and only if $x$ is a yes-input of $D$.

**NP-complete** A decision problem $D$ is NP-complete if it is in NP and all problems in NP are polynomially reducible to $D$.

**Permutation** A sequence $a$ is a permutation of a sequence $b$ if $a$ and $b$ contain the same elements at the same number of positions.

**P, complexity class** The class P is the class of decision problems that can be solved by a polynomial time algorithm.

**Polynomial time** An algorithm is called polynomial time if its worst-case time complexity $T(n)$ is in $O(p(n))$ for some polynomial $p$ (i.e., $p(n) = n$, $p(n) = n^2$, $p(n) = n^3$, etc.).

**Polynomially reducible** A decision problem $D_1$ is polynomially reducible to a decision problem $D_2$ if there is a function $t$ that transforms inputs of $D_1$ to inputs of $D_2$ such that $t$ is computable by a polynomial time algorithm and maps yes-inputs of $D_1$ to yes-inputs of $D_2$ and no-inputs of $D_1$ to no-inputs of $D_2$.

# Data Structures

**Adjacency Matrix** An adjacency matrix is an $n \times n$ table with 0/1 entries that represent a graph $G = (V, E)$ of $n$ vertices by storing in row $i$ and column $j$ whether there is an edge between vertices $i$ and $j$ (0 means no edge, 1 means edge present).

**Adjacency List** An adjacency list is a list containing $n$ lists that represents a graph of $n$ vertices by storing in the $i$-th list the indices of the vertices adjacent to vertex $i$ in order.

**Binary Search Tree** A binary search tree is a binary tree where all vertices are labelled with mutually comparable keys and that satisfies the following order property: for each vertex $v$ with label $k$ the labels of all vertices in the left subtree of $v$ are less or equal to $l$ and the labels of all vertices in the right subtree of $v$ or greater or equal to $k$.

**Bit List** A bit list is a list of 0/1 entries that represents a sub-selection of elements of some base list of options of the same length through the following convention: the $i$-th option is present in the sub-selection if the $i$-th element of the bit list is 1 (and 0 otherwise).

**Heap** A (min) heap is a binary tree where all vertices are labelled with mutually comparable keys that satisfies the following two properties: 1. The tree is essentially complete, i.e., only some right-most vertices on the last level can be missing. 2. The tree is partially ordered, i.e., for each vertex $v$ with label $k$, the labels of each of its children are greater or equal to $k$.

**Pair-of-children List** A pair-of-children list represents a binary tree as a list of length equal to the number of vertices where index $i$ contains a pair $(l, r)$ referring to the indices of the left and the right child of $i$, respectively (or a special value if a child is missing; in Python we use `None` for this case).

**Parent List** A parent list represents a rooted tree as a list of length equal to the number of vertices where the parent of vertex $i$ is stored at index $i$ of the list.

**Queue** A queue is a linear data structure in which elements are added only to the back (enqueue) and removed only from the front (dequeue).

**Stack** A stack is a linear data structure in which elements are added (push) and removed (pop) only from the back.

**Table** A table is a data structure that organises a set of values along the two dimensions of rows and columns.

## Graphs

**Clique** A clique is a subset of vertices $C$ of a graph such that for every two distinct vertices $v, w$ from $C$ there is an edge between $v$ and $w$ (cf. independent set).

**Connected** A graph is called connected if it contains a path between any two of its vertices.

**Cycle** A cycle is a path $P = [v_1, \ldots, v_k]$ in a graph such that $v_1 = v_k$.

**Independent Set** An independent set is a subset of vertices $C$ of a graph such that for every two distinct vertices $v, w$ from $C$ there is *no* edge between $v$ and $w$ (cf. clique).

**Hamiltonian Cycle** A Hamiltonian Cycle of a graph is a cycle that contains all vertices of that graph.

**Path** A path is a non-self-intersecting sequence $P = [v_1, \ldots, v_k]$ of successively connected vertices in a graph $G = (V, E)$. That is, for all $i$ from 1 to $k$ we have that $(v_i, v_{i+1})$ is in $E$ and additionally for all $j$ from $i + 1$ to $k - 1$ we have that $v_i \neq v_j$ (that means a vertex is only allowed to show up twice in the path if it is the start and end vertex, in which case we refer to the path as a *cycle*).

**Path Length** The length of a path $P = [v_1, \ldots, v_k]$ in an edge-weighted graph $G = (V, E, w)$ is the sum of all edge weights along the path, i.e., $w(v_1, v_2) + \ldots + w(v_{k-1}, v_k)$. In an unweighted graph the length $P$ is simply the number of contained edges $k - 1$, which is the same as assuming that all edge weights are equal to 1.

**Spanning Tree** A spanning tree of a graph $G = (V, E)$ is a subgraph $T = (V, F)$ of $G$ that contains all vertices of $G$ and that is a tree.

**Simple** A graph is called simple if it does not contain loops or parallel edges. In this unit we usually assume graphs to be simple unless we explicitly mention otherwise.

**Tree** A tree is a graph that is connected and acyclic (does not contain any cycles).

**Vertex Cover** A vertex cover is a subset of vertices $C$ of a graph such that for every edge of the graph either of the two end points is in $C$.

## Rooted Trees

**Binary Tree** A binary tree is a rooted tree where each vertex has at most two children, a left child and a right child.

**Child** In a tree with root $r$, a vertex $v$ is called the child of another vertex $p$ if $p$ is the predecessor of $v$ on the unique path from $v$ to the root $r$.

**Complete** A rooted tree is called complete (or essentially complete) if only some right-most vertices on the last level are missing (when compared to a perfect tree).

**Height** The height of a rooted tree is the maximum length of a path from the root to a leaf (or $-1$ if the tree is empty).

**Inner Vertex** A vertex in a rooted tree is called an inner vertex if it has at least one child.

**Leaf** A vertex in a rooted tree is called a leaf it it does not have any children.

**Level** The level of a vertex $v$ in a tree with root $r$ is the length of the unique path from $v$ to $r$. This implies that the level of $r$ is 0.

**Node** The term *node* is a synonym for the term *vertex* which is often used in the context of rooted trees. To avoid confusion, we avoid using this synonym in this unit.

**Parent** The parent of a vertex $v$ in a tree with root $r$ is the predecessor of $v$ on the unique path from $v$ to the root $r$ (the root itself does not have a parent).

**Perfect** A binary tree is called perfect if all of its inner vertices have exactly two children and all leafs are on the same level.