# Hieroglyphs

¬: NOT, Negation, !, det modsatte. SymPy: Not(p)

→: Implies, “giver”. SymPy: Implies(p,q)

↔: If and only if, iff,equivalence, biconditional. SymPy: Equivalent(p,q)

**∨**: OR, |, Disjunction. SymPy: Or(p,q)

**∧**: AND, Conjunction. SymPy: And(p,q)

⊥: For any set A and sentence p the remainder set A⊥p (“A remainder p”) is the set of inclusion-maximal subsets of A that do not imply p. In other words, a set B is an element of A⊥p if and only if it it is a subset of A that does not imply p, and there is no set B’ not implying p such that B⊂ B’⊆A. The elements of A⊥p are called “remainders”.

∈: Tilhører, belongs to

⊂: Subset of, f.eks. set A is a subset of a set B if all elements of A are also elements of B

Cn: For any set B of sentences, Cn(B) is the set of logical consequences of B

### Non-deterministic search

Π: “a function that maps each state s ∈ S to a non-empty set of actions denoted by Π(s) “

# Search Algorithms

## Tree search. Tree vs Graph

So, the basic **differences** are

1. In the case of a graph search, we use a list, called the **closed list** (also called **explored set**), to keep track of the nodes that have already been visited and expanded, so that they are not visited and expanded again.
2. In the case of a tree search, we do **not** keep this closed list. Consequently, the same node can be visited multiple (or even infinitely many) times, which means that the produced tree (by the tree search) may contain the same node multiple times.

## Uninformed Search Algorithms

1. **Depth-first search(DFS)** : The algorithm starts at the source node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.
2. **Breadth-first search(BFS)** : It starts at the source node and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level. It uses the opposite strategy of depth-first search, which instead explores the node branch as far as possible before being forced to backtrack and expand other nodes.
3. **Bidirectional BFS**: It runs two simultaneous Breadth-first searches, one from source node towards destination node and other from destination node towrds the source node.

## Informed Search Algorithms

1. **A\* search:** A\* search evaluates nodes by combining the cost to reach the node, and the cost to get from the node to the goal. It always provides the shortest path between two nodes.
2. **Greedy Best-first search:** Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Unlike A\* algorithms it doesn't always provides optimal path.

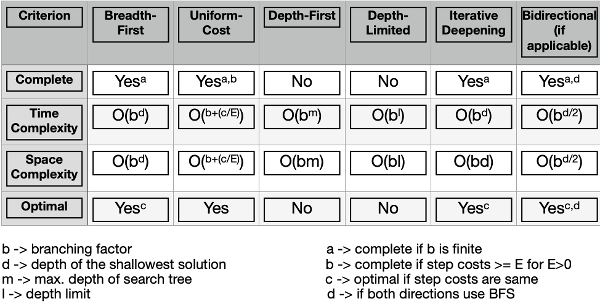
## Properties

**Completeness:** A search algorithm is said to be complete if it guarantees to return a solution if at least any solution exists for any random input.

**Optimality:** If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.

**Time Complexity:** Time complexity is a measure of time for an algorithm to complete its task.

**Space Complexity:** It is the maximum storage space required at any point during the search, as the complexity of the problem.



|  |  |  |
| --- | --- | --- |
|  | **Greedy Best First Search** | **A\*** |
| **Complete** | No | Yes |
| **Optimal** | No | Yes\* |

A\* Tree Search is optimal if heuristic is admissible.

A\* Graph Search is optimal if heuristic is admissible & consistent.

## Heuristics

An admissible heuristic is used to estimate the cost of reaching the goal state in an informed search algorithm. **In order for a heuristic to be admissible to the search problem, the estimated cost must always be lower than or equal to the actual cost of reaching the goal state**.

1. **Admissible heuristic**: When for each node *n* in the graph, *h(n)* **never overestimates** the cost of reaching the goal.
2. **Consistent heuristic**: When for each node *n* in the graph and each node *m* of its successors, *h(n) <= h(m) + c(n,m)*, where *c(n,m)* is the cost of going from *n* to *m*.   
   For every node *N* and each succesor *P* of *N*, the estimated cost of reaching the goal from *N* is no greater than the step cost of getting to *P* plus the estimated cost of reaching the goal from *P*.

Consistency implies admissibility. In other words, if a heuristic is consistent, it is also admissible. However, admissibility does not imply consistency.

A picture containing circle, clipart, illustration

Description automatically generated

H(c) = 1 : Not consistent

H(c) = 3 : Consistent

# Clauses

**Horn clause** is a type of logical formula used in mathematical logic and logic programming. It is a clause (a disjunction of literals) with at most one positive (i.e., unnegated) literal

**Definite clause** is a type of Horn clause that has exactly one positive (i.e., unnegated) literal. In other words, it is a disjunction of literals with exactly one positive literal and zero or more negative literals. For example: (¬p ∨ ¬s ∨ r) is a deﬁnite clause, while (p ∨ s ∨ ¬r) is not.

**Goal clause** is a type of Horn clause that has no positive literals. In other words, it is a disjunction of negated literals. Note that the empty clause, which consists of no literals and is equivalent to false, is also considered a goal clause

# Plausibility Orders

Practice Exam 2023

1. Initial state is the most plausible state = y

2. Revision with not p is the most plausible state that does not imply p = z

3. Contraction with p is the union of those two = y and z