GENERIC GRAPH SEARCH

A graph search searches a **search space** (state space) for **A PATH** from the **initial state** to a **goal state** (of which there may be many).

The search space can be
- an **explicit graph** (e.g., maze or Michigan cities/roads)

or
- an **implicit graph** (i.e., a production system) (e.g., missionaries & cannibals or Robbie building a block tower using A2 moves).

The path can be: SOME path or the MINIMUM COST (shortest) path

---

Categories of nodes: All nodes in the graph are either:

1) OPENED – generated, but not yet evaluated
2) CLOSED – generated, evaluated, and finished with
3) NOT YET GENERATED (for an implicit graph) or NOT YET CONSIDERED (for an explicit graph)

Only OPEN and CLOSED nodes are stored.

In the algorithms below, these are stored as 2 separate SETS of nodes: OPENED and CLOSED.

In the Prolog A code, each opened or closed node is asserted as a fact, either an* `opened(...)` *fact or a* `closed(...)` *fact.*

The Selection Criteria

- are you searching a tree or a graph?
  (for graphs, you have to check for cycles → infinite loops)
- each time around the loop choose the *"BEST"* node (from OPENED)
  - but **HOW DO YOU MEASURE "BEST"?**
  - what information to you have about the problem (search space)?
    - any DATA? (known "truths") (edge weights or cost of doing a "move")
    - any KNOWLEDGE? (estimating capabilities) (evaluation function for ESTIMATING the path-cost (or sum of "move" costs) from THIS node to the GOAL)

Some Costs

FinalCost (from initial state to goal state via *this node*) =
the actual cost you've **GoneThusFar** (from the initial state to *this node*)
PLUS the cost of **HowFarYetToGo** (from *this node* to the goal state)

`f(of some node) = g(of that node) + h(of that node)`

`g(of a node)` is knowable DATA.

But `h(of a node)` can only be ESTIMATED based on KNOWLEDGE
so use we use `h^(of a node)` as the estimate of `h(of a node)`.
So you only can get `f^(of a node)` as the estimate of `f(of a node)`.

So: **`f^(node) = g(node) + h^(node)`**

That is:

the ***ESTIMATED*** **FinalCost** (from initial state to goal state via *this node*) =
the ***ACTUAL*** cost you've **GoneThusFar** (from initial state to *this node*)
PLUS the ***ESTIMATED*** cost of **HowFarYetToGo** (from *this node* to the goal state)

What needs to be stored for each node in either set (in the algorithms below)?

For $\mathbf{A^T}$	nodeName, its g value,	its parentPtr
For $\mathbf{A^{KT}}$	nodeName, its g value, its f value,	its parentPtr
For **A***	nodeName, its g value, its f^ value,	its predecessorPtr

In the Prolog A code, the stored facts have the format:*
opened(StateName, PathList, G, FHat).
where PathList is a list of states from THIS state back to the INITIAL state: [StateName, Predecessor, ..., StartState]
closed(StateName, G).

$\mathbf{A^T}$ (Algorithm for searching a TREE, using only DATA)

```
CLOSED = { }
OPENED = { root }
        g(root) = 0
        parentPtr(root) = null

while OPENED still has nodes and you've not yet hit a GOAL
{       let Target = node in OPENED with smallest g value
                (if ties. . . is 1 of them the GOAL, then choose it, else its arbitrary)
        if  Target = GOAL
                exit successfully     // the minimum-cost path is g(Target)
        else // expand Target node
        {       move Target from OPENED to CLOSED
                generate all successors of Target [apply all legal moves to Target]
                        & for each one:
                            put that Successor node in OPENED
                            g(Successor) = g(Target) + cost (Target to Successor)
                            parentPtr(Successor) = Target
        }
}
```

This fails if you run out of nodes in OPENED before reaching a GOAL i.e., there is no path from the root to a goal node.

A^{KT} (Algorithm for searching a TREE, using KNOWLEDGE)

```
CLOSED = { }
OPENED = { root }
            g(root) = 0
            calculate (or lookup)  h^(root)
            f^(root) = g(root) + h^(root)
            parentPtr(root) = null

while OPENED still has nodes and you've not yet hit a GOAL
{           let Target = node in OPENED with smallest f^ value
                        (if ties. . .)
            if  Target = a GOAL
                        exit successfully      //  the minimum-cost path is g(Target)
            else  // expand Target node
            {           move Target from OPENED to CLOSED
                        generate all successors of Target [ apply all legal moves to Target]
                              & for each one:
                                    put it in OPENED
                                    g(Successor) = g(Target) + cost (Target to Successor)
                                    calculate (or lookup)  h^(Successor)
                                    f^(Successor) = g(Successor) + h^(Successor)
                                    parentPtr(Successor) = Target
            }
}
```

This fails if you run out of nodes in OPENED before reaching a GOAL
i.e., there is no path from the root to a goal node.

A* (The famous A STAR Algorithm for searching a GRAPH (or a TREE) using KNOWLEDGE (and DATA)

```
CLOSED = { }
OPENED = { initialState }
            g(root) = 0
            calculate (or lookup)  h^(initialState)
            f^(initialState) = g(initialState) + h^(initialState)
            predecessorPtr(initialState) = null

while OPENED still has nodes and you've not yet hit a  GOAL
{           let Target  = node in OPENED with smallest f^ value
                        (if ties. . .)
            if  Target = GOAL
                        exit successfully      //  the minimum-cost path is g(Target)
            else  // expand Target node
            {           move Target from OPENED to CLOSED
                        generate all successors of Target [ apply all legal moves to Target]
                              & for each one:
                              {       calculate gPrime(Successor)
                                               = g(Target) + cost(Target to Successor)
                                      if Successor NOT already in OPENED or CLOSED
                                      {          put it in OPENED
                                                 g(Successor) = gPrime(Successor)
                                                 calculate (or lookup) h^(Successor)
                                                 f^(Successor) = g(Successor) + h^(Successor)
                                                 predecessorPtr(Successor) = Target
                                      }
                                      else  // it's previously been OPENED or CLOSED
                                      {      if gPrime(Successor) <  g(Successor)
                                             {     g(Successor) = gPrime(Successor)
                                                   change  predecessorPtr(Successor) to Target
                                                   calculate (or lookup)  h^(Successor)
                                                   calculate NEW f^(Successor)
                                                   if Successor was in CLOSED
                                                            do fixDescendents procedure
                                             }
                                             else   // gPrime >= g
                                                   do nothing more with this Successor
                                      }
                              }
            }
}
```

This fails if you run out of nodes in OPENED before reaching a GOAL
i.e., there is no path from the initial state to a goal node.