basic-search
The algorithm in Figure 3.7, page 77 (TREE-SEARCH is the most basic version wich does not check for repeated nodes. GRAPH-SEARCH does the same as TREE-SEARCH but checking for repeated nodes). The version without searching for repeated nodes should be something like:

```
“thegame is the game to solve, queuing-fn is the queuing function"
 (defun basic-search (thegame queuing-fn)
   "First create an initial queue (aka as frontier in the book, aka as nodes in the aima code) with the initial game (thegame) on it"
   "Then loop taking the first node from the queue and adding its successors to the queue if it is not a goal state"
   (loop
          "if the queue is empty return NIL"
          "get the first element (node) from the queue (and remove it from the queue)"
          "if the node is the goal state which you check with (goalp (game-board node)) or (goalp node) depending whether
                                                                          your goalp takes a board or a game as argument"
                  "return the node"
          (funcall queuing-fn nodes (expand node))   "Expands the node using EXPAND. Puts the successors (results from expand) on
                                                        the queue (nodes) using QUEUING-FN"
  ))
```

The above basic-search function will behave differently according to the queuing-fn passed.
It can behave as DFS, BFS, BestFirst, etc.
So, our specialized search algorithms will call this function and will pass the corresponding queuing-fn.
For example, depth-first-search should pass a function that inserts the nodes at the front of the queue as follows:

```
(defun depth-first-search (thegame)
  (basic-search thegame #'enqueue-at-front))
```

That's it. That is the depth first search! The other two searches look the same. They just pass the appropriate enqueuing function to get the desired behavior from basic-search.

The EXPAND function used by basic-search should return all the legal games that would result from making ONE change to the board of the current game (passed as an argument to EXPAND)"

```
(defun expand (currentgame)
        "Return all the legal successors of the current game"
        "All generated successor games should have:
                a board that is legal and that results from making ONE change to the board of currentgame
                currentgame as parent
                depth = depth of currentgame + 1
        )
```