Iterative Deepening A* (IDA*) Search: Rough Algorithm from LISP

```
function IDA
        root <- start node
        f-limit <- fcost(root)
        solution <- NIL
        loop
                DFS_Contour(root, f-limit)
                if not null(solution) then return(solution)
                if f-limit = infinity then return(NIL)
        end loop
end IDA

function DFS_Contour(node, f-limit) returns (solution, f-limit)
        next-f <- infinity
        if fcost(node) > f-limit then mreturn(NIL, fcost(node))
        else if goal(node) then mreturn(node, f-limit)
        else for each successor s of node
                new-f <- fcost(s)
                DFS-Contour(s, f-limit)
                if not null(solution) then mreturn(solution, f-limit)
                else next-f <- min(next-f, new-f)
        mreturn(NIL next-f)
end DFS_Contour
```

```
;;;  ida.lisp
;;;; Iterative Deepening A* (IDA*) Search
(setf (problem-iterative? problem) t)
(let* ((root (create-start-node problem)) (f-limit (node-f-cost root)) (solution nil))
  (loop (multiple-value-setq (solution f-limit) (DFS-contour root problem f-limit))
    (dprint "DFS-contour returned" solution "at" f-limit)
    (if (not (null solution)) (RETURN solution))
    (if (= f-limit infinity) (RETURN nil)))))

(defun DFS-contour (node problem f-limit)
"Return a solution and a new f-cost limit."
  (let ((next-f infinity))
    (cond ((> (node-f-cost node) f-limit)  (values nil (node-f-cost node)))
          ((goal-test problem (node-state node)) (values node f-limit))
          (t (for each s in (expand node problem) do
               (multiple-value-bind (solution new-f) (DFS-contour s problem f-limit)
                 (if (not (null solution))
                     (RETURN-FROM DFS-contour (values solution f-limit)))
                 (setq next-f (min next-f new-f))))
             (values nil next-f)))))
```