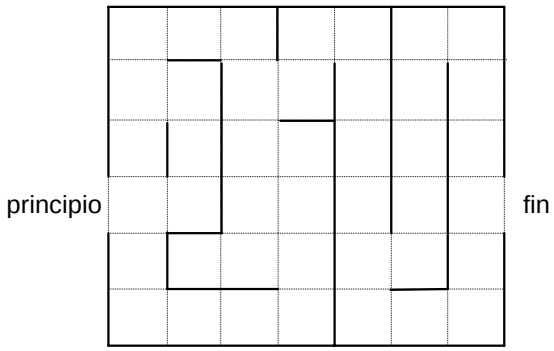
Examen conv. 2 curso 97-98 **26 Junio 1997** **2 horas *[80 ptos.]***

Apellidos y Nombre: .....

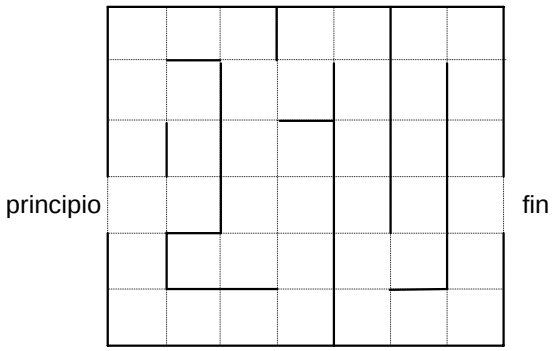
Problema 1: Búsquedas *[20 puntos]*

Distintas búsquedas pueden resolver este laberinto, encontrando un camino desde la localización inicial hasta la final. Los posibles operadores (y en este orden) son: **arriba**, **abajo**, **izquierda**, **derecha,** que son sólo válidos si hay una línea de puntos entre las celdas del laberinto. Asumiremos que en las búsquedas que lo requieran hay siempre chequeo de nodos duplicados y que hay un coste unitario por cada movimiento. Indica con un numero en cada casilla el orden de visita utilizando una:

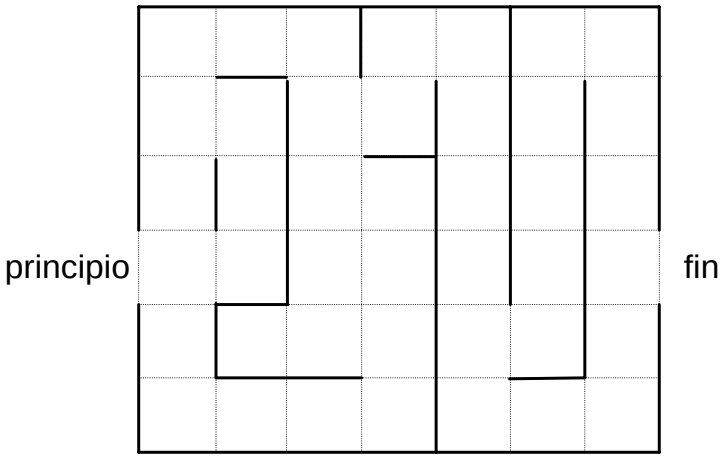
1. Búsqueda en anchura



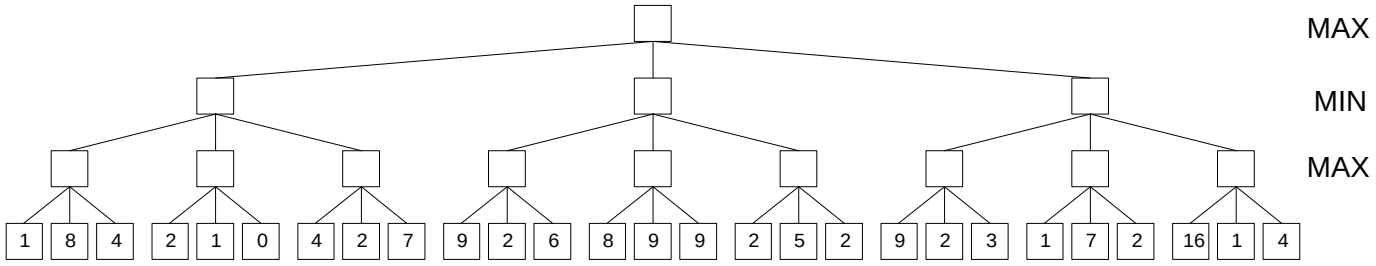
2. Búsqueda en profundidad



3. Búsqueda A* (indicar en cada estado los valores de g' y h'). Heurística: distancia mínima si no hubiera obstáculos.



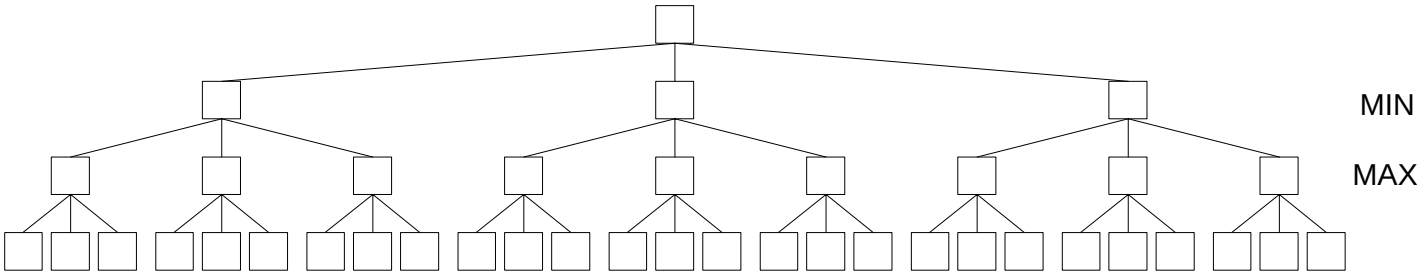
Problema 2: Búsqueda en juegos *[15 puntos]*



1. Completa el árbol de juego dibujado arriba, rellenando los valores de todos los nodos, incluyendo el nodo raiz, utilizando búsqueda min-max.

2. Señala con un círculo los nodos que serian podados en la poda alfa-beta.

3. Reordena los nodos (de izquierda a derecha) de forma que den como resultado el máximo número de nodos podados. Reordena los hijos preservando las relaciones hijo-padre.



NOTA: **Alfa**: (para un nodo MAX) es el valor más alto visto hasta el momento de los valores finales, calculados hacia atrás, de sus sucesores. **Beta**: (para un nodo MIN) es el valor más bajo visto hasta el momento de los valores finales, calculados hacia atrás, de sus sucesores.
Poda (fin de la llamada recursiva): Puede suspenderse la exploración por debajo de cualquier nodo MIN que tenga valores de Beta menores o iguales que el valor Alfa de cualquiera de sus nodos MAX ascendientes suyos. Puede suspenderse la exploración por debajo de cualquier nodo MAX que tenga valores de Alfa mayores o iguales que el valor Beta de cualquiera de sus nodos MIN ascendientes suyos.

Problema 3: Búsquedas no informadas/Common Lisp/Prácticas *[25 puntos]*

El siguiente código corresponde al programa de búsqueda que has utilizado en las prácticas 2 y 3.

```
(defconstant estado-inicial ...)
(defvar el-nodo-inicial nil "Nodo que plantea la situacion inicial")
(defvar nodos-a-expandir nil "Lista de estados a considerar")
(defvar nodos-expandidos nil "Lista de estados que ya han sido visitados
una vez")

 (defun crea-nodo (estado padre &optional (simbolo (gensym "NODO-")))
  (setf (get simbolo 'estado) estado)
  (setf (get simbolo 'padre) padre)
  simbolo)

 (defun mc ()
  (setq nodos-expandidos nil                          ;(1)
        el-nodo-inicial (crea-nodo estado-inicial nil 'nodo-inicial)
        nodos-a-expandir (list el-nodo-inicial)       ;(2)
        )
  (do* ((el-nodo (pop nodos-a-expandir)
                 (if nodos-a-expandir
                   (pop nodos-a-expandir)             ;(4)
                   (return (mensaje-de-error))))      ;(3)
        )
       ((objetivop el-nodo)                           ;(5)
        (escribe-solucion el-nodo)
        )
    (push el-nodo nodos-expandidos)
    (reorganizar-nodos-a-expandir                     ;(7)
      (expandir-nodo el-nodo))))                      ;(6)

 (defun mensaje-de-error ()
  (format t "~%~%ERROR!!!, no es posible seguir con el proceso de
busqueda."))

(defun escribe-solucion (solucion)
  (format t "~%~%Resuelto ! El camino de la solucion es: ")
  (escribe-un-camino solucion)
  'hecho)

(defun escribe-un-camino (nodo)
  (if (get nodo 'padre) (escribe-un-camino (get nodo 'padre)))
  (print (get nodo éstado)))

(defun objetivop (nodo) ...)

(defun expandir-nodo (nodo) {let* ... }}

(defun reorganizar-nodos-a-expandir (nodos)
  (and nodos
       (setq nodos-a-expandir (append nodos-a-expandir nodos))
      ))
```

a – Dado el código anterior que estrategia de búsqueda se realizaría con una llamada a `(mc)`.

NOTA - Al hacer los siguientes apartados pon especial cuidado en que el código que escribas sea el mínimo y lo mas “elegante” posible.

b – Modifica el código anterior para que se produzca una búsqueda en profundidad con límite 6.

c – Crea una nueva función, llamada `mc2`, que realice una búsqueda en profundidad con profundización iterativa. El incremento de profundidad será el indicado por el parámetro `incremProfundidad`.

Problema 4: Representaciones estructuradas/Common Lisp/Prácticas *[20 puntos]*

[Tendrá gran valor en la nota del problema que en tu solución demuestres tu conocimiento sobre frames y que el código resulte claro, sencillo y elegante]

El siguiente código corresponde a la base de datos sobre apartamentos del programa que has utilizado en la práctica 5.

```
    (form :is-a nil :name 'object)
    (form :name 'cosa
          :is-a 'object)
    (form :name 'vivienda
          :is-a 'cosa
          :slots (list (list 'ruido-calle 'if-needed #'calcula-ruido)
                       (list 'propietario)
                       (list 'calle-nombre)
                       (list 'calle-numero)
                       (list 'piso)
                       (list 'color-pared)
                       (list 'material-suelo)
                       (list 'numero-habitaciones)
                       (list 'distancia-carretera)))
    (form :name 'apartamento
          :is-a 'vivienda
          :slots '((numero-habitaciones = 2)
                   (color-pared = blanco)
                   (material-suelo = gres)
                   (piso)
                   (mano)))
    (form :name 'chalet
          :is-a 'vivienda
          :slots (list (list 'ruido-calle
                             'if-needed #'calcula-ruido-en-casa)
                       (list 'num-pisos '= '2)))
    (form :name 'apt-breton-22
          :is-a 'apartamento
          :slots '((calle-nombre = Breton)
                   (calle-numero = 22)
                   (material-suelo = madera)))
    (form :name 'apt-22-1
          :is-a 'apt-breton-22
          :slots '((numero-habitaciones = 3)
                   (mano = derecha)
                   (piso = 2)))
    (form :name 'apt-22-2
          :is-a 'apt-breton-22
          :slots '((numero-habitaciones = 4)
                   (mano = izquierda)
                   (piso = 2)))
))
...

(defun calcula-ruido (form slot)
  (let* ((valor (find-aspect-from-supers form 'ruido-calle '=)) piso)
    (cond (valor valor)
          (t (setq piso (get-value form 'piso))
             (cond ((> piso 15) 'muy-bajo) ((> piso 8) 'bajo)
                   ((> piso 4) 'medio) ((> piso 1) 'alto)
                   (t 'muy-alto))))))
```

a – Escribe el código necesario para que siempre que se pida el valor del slot `propietario` de una vivienda (`(get-value 'apt-22-1 'propietario)`) ocurra lo siguiente>

- se incremente un contador que lleve la contabilidad del número de veces que se ha pedido dicho valor de dicha vivienda
- se escriba dicho valor en el dispositivo de salida

p.e.

```
(get-value 'apt-22-1 'propietario)
dispositivo de salida--> 1
(get-value 'apt-22-1 'propietario)
dispositivo de salida--> 2
(get-value 'apt-22-2 'propietario)
dispositivo de salida--> 1
```

b – Haz las modificaciones de código necesarias para propagar la contabilidad de acceso al slot `propietario` a todos sus antecesores en la jerarquía. De esta forma, al acceder a cualquier objeto descendiente de `object` se escribirá en el dispositivo de salida la suma de accesos correspondientes a todos sus descendientes.
p.e.

```
(get-value 'apt-22-2 'propietario)
```

dispositivo de salida--> 2

```
 (get-value 'vivienda 'propietario)
```

dispositivo de salida--> 5