Problema 2 – Laberinto

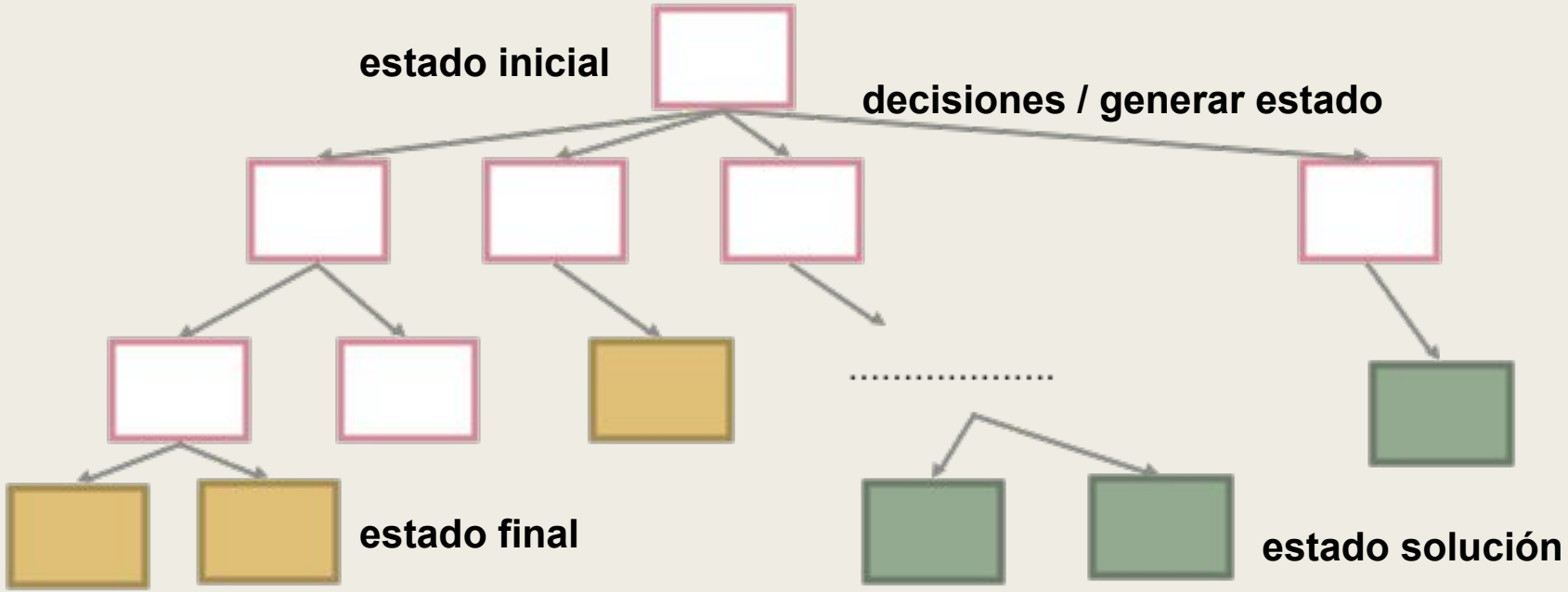
Dado un laberinto consistente en una matriz cuadrada que tiene en cada posición un valor natural y cuatro valores booleanos, indicando estos últimos si desde esa casilla se puede ir al norte, este, sur y oeste, encontrar **un camino de longitud mínima** entre dos casillas dadas, siendo la longitud de un camino la suma de los valores naturales de las casillas por las que pasa.

Idea:
podría representarse el laberinto como una matriz, de objetos, donde cada objeto tiene el valor natural, y cuatro booleanos, uno para cada dirección a la que se permite ir desde allí.

Problema 2 – Laberinto

Técnicas de programación

Backtracking



camino [(0,0)]
costo=1

1 (d,r)	2 (l,d)	6 (l,d,r)	2 (l)
3 (r)	4 (l,d,r,u)	7 ()	17 (l,d)
7 (r,d)	2 (l,d,r,u)	10 (u,r)	3 (d)
2 (r,u)	4 (l,u,r)	4 (l,u,r)	3 (l,u)

Estado inicial

Estado inicial

1 (d,r)	2 (l,d)	6 (l,d,r)	2 (l)
3 (r)	4 (l,d,r,u)	7 ()	17 (l,d)
7 (r,d)	2 (l,d,r,u)	10 (u,r)	3 (d)
2 (r,u)	4 (l,u,r)	4 (l,u,r)	3 (l,u)

camino [(0,0)]
costo=1

d

caminoActual [(0,0),(1,0)]
costo=4

1 (d,r)	2 (l,d)	6 (l,d,r)	2 (d)
3 (r)	4 (l,d,r,u)	7 ()	17 (l,d)
7 (r,d)	2 (l,d,r,u)	10 (u,r)	3 (d)
2 (r,u)	4 (l,u,r)	4 (l,u,r)	3 (l,u)

r

camino[(0,0), (0,1)]
costo = 3

1 (d,r)	2 (l,d)	6 (l,d,r)	2 (d)
3 (r)	4 (l,d,r,u)	7 ()	17 (l,d)
7 (r,d)	2 (l,d,r,u)	10 (u,r)	3 (d)
2 (r,u)	4 (l,u,r)	4 (l,u,r)	3 (l,u)

Estado inicial

1 (d,r)	2 (l,d)	6 (l,d,r)	2 (l)
3 (r)	4 (l,d,r,u)	7 ()	17 (l,d)
7 (r,d)	2 (l,d,r,u)	10 (u,r)	3 (d)
2 (r,u)	4 (l,u,r)	4 (l,u,r)	3 (l,u)

camino [(0,0)]
costo=1

d

caminoActual [(0,0),(1,0)]
costo=4

1 (d,r)	2 (l,d)	6 (l,d,r)	2 (d)
3 (r)	4 (l,d,r,u)	7 ()	17 (l,d)
7 (r,d)	2 (l,d,r,u)	10 (u,r)	3 (d)
2 (r,u)	4 (l,u,r)	4 (l,u,r)	3 (l,u)

r

camino[(0,0), (0,1)]
costo = 3

1 (d,r)	2 (l,d)	6 (l,d,r)	2 (d)
3 (r)	4 (l,d,r,u)	7 ()	17 (l,d)
7 (r,d)	2 (l,d,r,u)	10 (u,r)	3 (d)
2 (r,u)	4 (l,u,r)	4 (l,u,r)	3 (l,u)

r

camino [(0,0),(1,0), (1,1)]
costo=8

1 (d,r)	2 (l,d)	6 (l,d,r)	2 (d)
3 (r)	4 (l,d,r,u)	7 ()	17 (l,d)
7 (r,d)	2 (l,d,r,u)	10 (u,r)	3 (d)
2 (r,u)	4 (l,u,r)	4 (l,u,r)	3 (l,u)

l

d

camino [(0,0),(0,1), (1,1)]]
costo = 7

1 (d,r)	2 (l,d)	6 (l,d,r)	2 (d)
3 (r)	4 (l,d,r,u)	7 ()	17 (l,d)
7 (r,d)	2 (l,d,r,u)	10 (u,r)	3 (d)
2 (r,u)	4 (l,u,r)	4 (l,u,r)	3 (l,u)

Estado inicial

1 (d,r)	2 (l,d)	6 (l,d,r)	2 (l)
3 (r)	4 (l,d,r,u)	7 ()	17 (l,d)
7 (r,d)	2 (l,d,r,u)	10 (u,r)	3 (d)
2 (r,u)	4 (l,u,r)	4 (l,u,r)	3 (l,u)

camino [(0,0)]
costo=1

d

r

caminoActual [(0,0),(1,0)]
costo=4

1 (d,r)	2 (l,d)	6 (l,d,r)	2 (d)
3 (r)	4 (l,d,r,u)	7 ()	17 (l,d)
7 (r,d)	2 (l,d,r,u)	10 (u,r)	3 (d)
2 (r,u)	4 (l,u,r)	4 (l,u,r)	3 (l,u)

camino[(0,0), (0,1)]
costo = 3

1 (d,r)	2 (l,d)	6 (l,d,r)	2 (d)
3 (r)	4 (l,d,r,u)	7 ()	17 (l,d)
7 (r,d)	2 (l,d,r,u)	10 (u,r)	3 (d)
2 (r,u)	4 (l,u,r)	4 (l,u,r)	3 (l,u)

camino [(0,0),(1,0), (1,1)…(3,3)]
costo=X

r

1 (d,r)	2 (l,d)	6 (l,d,r)	2 (d)
3 (r)	4 (l,d,r,u)	7 ()	17 (l,d)
7 (r,d)	2 (l,d,r,u)	10 (u,r)	3 (d)
2 (r,u)	4 (l,u,r)	4 (l,u,r)	3 (l,u)

l

d

camino [(0,0),(0,1), (1,1)]]
costo = 7

1 (d,r)	2 (l,d)	6 (l,d,r)	2 (d)
3 (r)	4 (l,d,r,u)	7 ()	17 (l,d)
7 (r,d)	2 (l,d,r,u)	10 (u,r)	3 (d)
2 (r,u)	4 (l,u,r)	4 (l,u,r)	3 (l,u)

```java
public class Backtracking {

	private Casillero destino;
	private Camino mejorCamino;

	public Camino back(Casillero origen) {
		Camino camino = new Camino();
		camino.agregarAlCamino(origen);
		camino.marcarVisitado(origen);
		camino.incrementar(origen.getValor());
		this.back(origen, camino);
		return mejorCamino;
	}

	private void back(Casillero actual, Camino caminoActual) {
		if (actual.equals(this.destino)) { // Si llegue a destino / condicion de corte
			if (mejorCamino == null || mejorCamino.getValor() > caminoActual.getValor()) {
				mejorCamino = caminoActual;
			}
		} else {
			ArrayList<Casillero> vecinos = actual.getVecinos(); // Me da los casilleros a los cuales me puedo mover
			for (Casillero vecino: vecinos) {
				if (!caminoActual.estaVisitado(vecino)) {
					// Aplicar / Hacer cambios / Agregar solucion
					caminoActual.agregarAlCamino(vecino);
					caminoActual.marcarVisitado(vecino);
					caminoActual.incrementar(vecino.getValor());

					if (mejorCamino == null || caminoActual.getValor() <= mejorCamino.getValor()) // Poda
						back(vecino,caminoActual); // Llamado recursivo

					// Deshacer
					caminoActual.quitarUltimo();
					caminoActual.quitarVisitado(vecino);
					caminoActual.decrementar(vecino.getValor());
				}
			}
		}
	}
}
```