ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA
Universitat Rovira i Virgili

PRÁCTICA 1

Búsqueda informada

Alumno/s: Anna Gracia Colmenarejo

Asignatura: Inteligencia Artificial

Grado: Ingeniería Informática

Profesor: David Sánchez Ruenes

Fecha: 26/10/2022

ÍNDICE

1. Formalización del problema

Los estados son las casillas del tablero, la información que se guarda en estos estados es la posición del tablero en la que se encuentran y su altura. Los operadores es el movimiento para pasar a otro estado por lo que en este problema se entiende como una suma o diferencia de la columna o fila de su posición.

2. Definición de las 3 heurísticas

La primera heurística se basa en ir trazando el camino con los nodos de menos peso que vamos encontrando. La lista de pendientes se ordena poniendo el nodo de menos peso en primera posición para que este sea el siguiente en ser tratado.

```
public void ordenar_pendientes(ArrayList<Estado> pend) {
    pend.sort(new Comparator<Estado>(){
        public int compare(Estado e1, Estado e2){
         return Integer.compare(e1.getAltura(),
      e2.getAltura());
        }
    });
}
```

La segunda heurística se basa en ir trazando el camino con los nodos que están más cerca del nodo final mediante la fórmula de la distancia euclidiana. La lista de pendientes se ordena poniendo el nodo más cercano en primera posición para que este sea el siguiente en ser tratado.

```
public void ordenar_pendientes(ArrayList<Estado> pend) {
    pend.sort(new Comparator<Estado>(){
        public int compare(Estado e1, Estado e2){
            Coordenada destine = new Coordenada(9,9);
            double dist1, dist2;
            dist1 = dist_euclidiana(e1, destine);
            dist2 = dist_euclidiana(e2, destine);
            return Double.compare(dist1,dist2);
```

```
        }
    });
}

public double dist_euclidiana(Estado origen, Coordenada
dest){
    double dist;
    int x1 = origen.getPos().getX();
    int y1 = origen.getPos().getY();
    int x2 = dest.getX();
    int y2 = dest.getY();
    dist =
Math.sqrt((Math.pow((x2-x1),2)+Math.pow((y2-y1),2)));
    return dist;
}
```

La tercera heurística se basa en ir trazando el camino con los nodos que están más cerca del nodo final y que pesan menos, es decir, es una mezcla de las dos heurísticas anteriores. Mediante la suma del resultado de la fórmula de la distancia euclidiana y la altura de cada nodo se ordena la lista de pendientes poniendo el nodo más cercano y menos pesado en primera posición para que este sea el siguiente en ser tratado.

```
public void ordenar_pendientes(ArrayList<Estado> pend) {
    pend.sort(new Comparator<Estado>(){
        public int compare(Estado e1, Estado e2){
            Coordenada destine = new Coordenada(9,9);
            double dist_peso1, dist_peso2;
            dist_peso1 = dist_eucli_peso(e1,destine);
            dist_peso2 = dist_eucli_peso(e2,destine);
            return Double.compare(dist_peso1,dist_peso2);
        }
    });
}
```

```
  public double dist_eucli_peso(Estado origen, Coordenada
  dest){
     double dist;
     int x1 = origen.getPos().getX();
     int y1 = origen.getPos().getY();
     int x2 = dest.getX();
     int y2 = dest.getY();

   dist = Math.sqrt((Math.pow((x2-x1),2)+Math.pow((y2-y1),2)))
+ origen.getAltura();
     return dist;
  }
```

3. Best first

Este algoritmo va mirando todas las posibilidades pero no garantiza que la solución sea la óptima.

```
public void busqueda(Heuristica h, Estado[][] grafo){
   ArrayList<Estado> pendientes = new ArrayList<>();
   ArrayList<Estado> tratados = new ArrayList<>();
   ArrayList<Estado> vecinos = null;
   Coordenada ini = new Coordenada(0,0);
   Coordenada fin = new Coordenada(9,9);
   Estado inicial = new Estado(ini,
grafo[ini.getX()][ini.getY()].getAltura());
   pendientes.add(inicial);
   boolean trobat = false;

   while(!trobat && !pendientes.isEmpty()){

       Estado e = pendientes.get(0);
       pendientes.remove(0);

       if(!e.getPos().compare(fin)){
           vecinos = e.obtener_vecinos(grafo);
           for(int i=0; i< vecinos.size(); i++){
               if(!tratados.contains(vecinos.get(i)) &&
!pendientes.contains(vecinos.get(i))){
                   pendientes.add(vecinos.get(i));
                   h.ordenar_pendientes(pendientes);
               }
           }
           tratados.add(e);
       }else{
           trobat=true;
           e.setCamino();
           e.getCamino().add(e);
```

```
            mostrar_camino(e.getCamino(),grafo);
            System.out.println("Nodos tratados: "
+tratados.size());
            System.out.println("Tiempo "
+calcularTiempo(e.getCamino(), inicial));
            System.out.println("");
        }
    }
}
```

4. A star

En este algoritmo se han añadido parámetros nuevos a tener en cuenta f,g y h. *G* es el coste que ha tenido llegar al nodo actual y *H* la estimación del coste del camino al nodo final. F es la suma de G y H que su resultado será el criterio a seguir para ordenar la lista de pendientes.

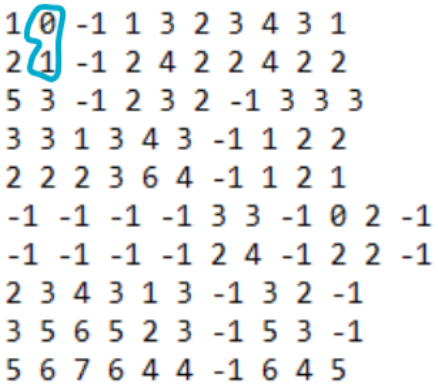
```
public void busqueda(Heuristica h, Estado[][] grafo) {
   ArrayList<Estado> pendientes = new ArrayList<>();
   ArrayList<Estado> tratados = new ArrayList<>();
   ArrayList<Estado> vecinos;

   Coordenada ini = new Coordenada(0,0);
   Coordenada fin = new Coordenada(9,9);
   Estado inicial = new
Estado(ini,grafo[ini.getX()][ini.getY()].getAltura());
   Estado efinal = new
Estado(fin,grafo[ini.getX()][ini.getY()].getAltura());
   pendientes.add(inicial);
   boolean trobat = false;

   inicial.setG();
   inicial.setF(inicial.getG(), h, efinal);

   while(!trobat && !pendientes.isEmpty()){

       Estado e = pendientes.get(0);
       pendientes.remove(0);

       if(!e.getPos().compare(fin)){
           vecinos = e.obtener_vecinos(grafo);
           for(int i=0; i< vecinos.size(); i++){
               if(!tratados.contains(vecinos.get(i)) &&
!pendientes.contains(vecinos.get(i))){
                   vecinos.get(i).setG();
```

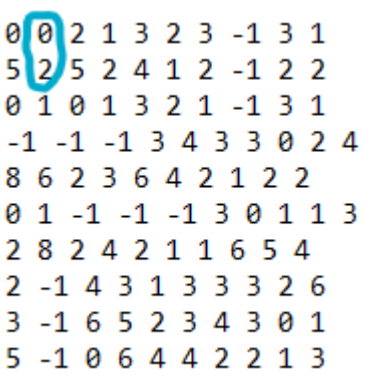
```
                    vecinos.get(i).setF(e.getG(),h,efinal);
                    pendientes.add(vecinos.get(i));
                    h.ordenar_pend(pendientes);
                }
            }
            tratados.add(e);
        }else{
            trobat=true;
            e.setCamino();
            e.getCamino().add(e);
            mostrar_camino(e.getCamino(),grafo);
            System.out.println("Nodos tratados:
"+tratados.size());
            System.out.println("Tiempo "+e.calcularTiempo());
            System.out.println("");
        }
    }
}
```

5. Hill Climbing

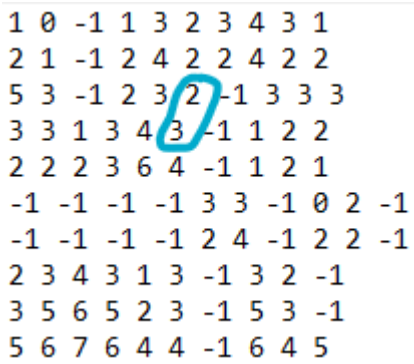
Este algoritmo no mira todas las posibilidades y no garantiza encontrar la solución óptima. En este caso la lista de pendientes desaparece y su funcionamiento se basa en mirar los vecinos del estado actual y mira cual es mejor si este no resulta mejor que el padre se para por lo que tampoco garantiza encontrar una solución.

- Heurística 1 → en el caso del tablero 1 y 2 no encontraría la solución ya que los nodos hijos no siempre son mejores que el padre en cuanto a peso.
  Como podemos ver en la imágenes a continuación para pasar del nodo (0,1) al (1,1) el peso aumenta





- Heurística 2 → en el caso del tablero 1 no encontraría la solución ya que los nodos hijos no siempre son mejores que el padre en cuanto a peso.
  Como podemos ver en la imágenes a continuación para pasar del nodo (3,5) al (2,5) el peso aumenta. Pero para el tablero 2 si que encontraría ya que no tendría que saltar el precipicio y por pasar por nodos peores.



- Heurística 3 → Pasaría lo mismo que para la heurística 2.

6. Resultados

TABLERO 1: BEST FIRST VS A*

```
Heuristica 1: Nodo de menos peso
1   0   -1  1   3   2   3   4   3   1
2   1   -1  2   4   2   2   4   2   2
5   3   -1  2   3   2   -1  3   3   3
3   3   1   3   4   3   -1  1   2   2
2   2   2   3   6   4   -1  1   2   1
-1  -1  -1  -1  3   3   -1  0   2   -1
-1  -1  -1  -1  2   4   -1  2   2   -1
2   3   4   3   1   3   -1  3   2   -1
3   5   6   5   2   3   -1  5   3   -1
5   6   7   6   4   4   -1  6   4   5
Nodos tratados: 69
Tiempo 31.0
```

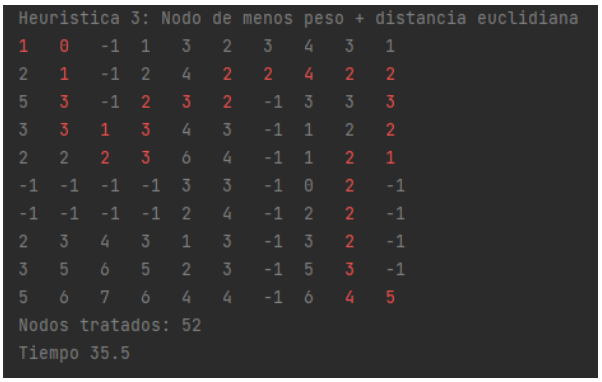
```
Heuristica 1: Nodo de menos peso
1   0   -1  1   3   2   3   4   3   1
2   1   -1  2   4   2   2   4   2   2
5   3   -1  2   3   2   -1  3   3   3
3   3   1   3   4   3   -1  1   2   2
2   2   2   3   6   4   -1  1   2   1
-1  -1  -1  -1  3   3   -1  0   2   -1
-1  -1  -1  -1  2   4   -1  2   2   -1
2   3   4   3   1   3   -1  3   2   -1
3   5   6   5   2   3   -1  5   3   -1
5   6   7   6   4   4   -1  6   4   5
Nodos tratados: 77
Tiempo 36.5
```

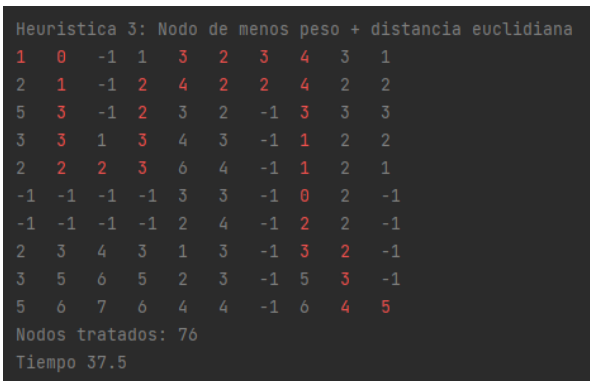
En esta primera heurística como se esperaba el best first muestra mejores resultados, tratando menos nodos, ya que el a* está diseñado para manejar heurísticas que traten la distancia de los nodos.

```
Heuristica 2: Distancia euclidiana
1   0   -1  1   3   2   3   4   3   1
2   1   -1  2   4   2   2   4   2   2
5   3   -1  2   3   2   -1  3   3   3
3   3   1   3   4   3   -1  1   2   2
2   2   2   3   6   4   -1  1   2   1
-1  -1  -1  -1  3   3   -1  0   2   -1
-1  -1  -1  -1  2   4   -1  2   2   -1
2   3   4   3   1   3   -1  3   2   -1
3   5   6   5   2   3   -1  5   3   -1
5   6   7   6   4   4   -1  6   4   5
Nodos tratados: 46
Tiempo 39.5
```

```
Heuristica 2: Distancia euclidiana
1   0   -1  1   3   2   3   4   3   1
2   1   -1  2   4   2   2   4   2   2
5   3   -1  2   3   2   -1  3   3   3
3   3   1   3   4   3   -1  1   2   2
2   2   2   3   6   4   -1  1   2   1
-1  -1  -1  -1  3   3   -1  0   2   -1
-1  -1  -1  -1  2   4   -1  2   2   -1
2   3   4   3   1   3   -1  3   2   -1
3   5   6   5   2   3   -1  5   3   -1
5   6   7   6   4   4   -1  6   4   5
Nodos tratados: 71
Tiempo 34.0
```

En esta segunda heurística como se esperaba el a* muestra mejores resultados, habiendo tratado más nodos, ya que esta heurística busca el nodo más cercano al final, el problema en este caso es que al ignorar los pesos el tiempo sigue siendo más elevado que el de la heurística anterior en el best first.





Y por último, vemos que el best first encuentra un camino más rápido y tratando menos nodos que el A* por lo que resulta mejor.

Cabe destacar que la distancia euclidiana se basa en el teorema de pitágoras por lo que siempre intenta realizar movimientos hacia el nodo final yendo hacia la derecha y hacia abajo intentando trazar la hipotenusa entre los dos nodos el problema con el que se encuentra esta heurística es que los precipicios le impiden dar mejores resultados ya que tiene que acabar desplazándose hacia nodos más lejanos del final para poder cruzar estos precipicios.

TABLERO 2: BEST FIRST VS A*

Para el segundo tablero con esta primera heurística mantenemos los mismos resultados que con el primer tablero, el best first muestra mejores resultados tratando menos nodos.

```
Heuristica 2: Distancia euclidiana
0   0   2   1   3   2   3   -1  3   1
5   2   5   2   4   1   2   -1  2   2
0   1   0   1   3   2   1   -1  3   1
-1  -1  -1  3   4   3   3   0   2   4
8   6   2   3   6   4   2   1   2   2
0   1   -1  -1  -1  3   0   1   1   3
2   8   2   4   2   1   1   6   5   4
2   -1  4   3   1   3   3   3   2   6
3   -1  6   5   2   3   4   3   0   1
5   -1  0   6   4   4   2   2   1   3
Nodos tratados: 18
Tiempo 30.5
```

```
Heuristica 2: Distancia euclidiana
0   0   2   1   3   2   3   -1  3   1
5   2   5   2   4   1   2   -1  2   2
0   1   0   1   3   2   1   -1  3   1
-1  -1  -1  3   4   3   3   0   2   4
8   6   2   3   6   4   2   1   2   2
0   1   -1  -1  -1  3   0   1   1   3
2   8   2   4   2   1   1   6   5   4
2   -1  4   3   1   3   3   3   2   6
3   -1  6   5   2   3   4   3   0   1
5   -1  0   6   4   4   2   2   1   3
Nodos tratados: 61
Tiempo 35.5
```

En esta segunda heurística el a* muestra peores resultados que el best first tratando muchos más nodos.

```
Heuristica 3: Nodo de menos peso + distancia euclidiana
0   0   2   1   3   2   3   -1  3   1
5   2   5   2   4   1   2   -1  2   2
0   1   0   1   3   2   1   -1  3   1
-1  -1  -1  3   4   3   3   0   2   4
8   6   2   3   6   4   2   1   2   2
0   1   -1  -1  -1  3   0   1   1   3
2   8   2   4   2   1   1   6   5   4
2   -1  4   3   1   3   3   3   2   6
3   -1  6   5   2   3   4   3   0   1
5   -1  0   6   4   4   2   2   1   3
Nodos tratados: 28
Tiempo 30.0
```

```
Heuristica 3: Nodo de menos peso + distancia euclidiana
0   0   2   1   3   2   3   -1  3   1
5   2   5   2   4   1   2   -1  2   2
0   1   0   1   3   2   1   -1  3   1
-1  -1  -1  3   4   3   3   0   2   4
8   6   2   3   6   4   2   1   2   2
0   1   -1  -1  -1  3   0   1   1   3
2   8   2   4   2   1   1   6   5   4
2   -1  4   3   1   3   3   3   2   6
3   -1  6   5   2   3   4   3   0   1
5   -1  0   6   4   4   2   2   1   3
Nodos tratados: 58
Tiempo 27.0
```

Finalmente al haber eliminado el precipicio que impedía la mejor aplicación de la distancia euclidiana y haciendo que esta a su vez también compruebe el peso de los nodos conseguimos el mejor resultado, pero claro no deja de dar los mejores resultados en un terreno donde los inconvenientes no se presentan tan drásticamente como en el mapa anterior.