Caballo Ávido: Siguiente posición con menor movimiento

Guillermo I. Bautista G. *Estudiante, UPIIZ*

Resumen—Las soluciones a problemas fue lo que nos llevo como especie a donde estamos ahora, el reconocer los problemas y afrontarlo parece tarea del día a día, cosa que de cierta forma lo es, pero que no siempre se resuelven de formas tan empíricas como lo era antes, hoy en día los problemas reales de la humanidad no son algo fácil de analizar, por lo que se crean métodos específicos para solucionar ciertas partes de un problema, o el mismo en ciertas condiciones, estos métodos nace de una idea o heurística, que es básicamente un salto de fe a la hora de resolver el problema.

Index Terms—Caballo, Algoritmo ,Heurística.

1. Introduction

Los algoritmos ávidos o algoritmos voraces, son aquellos que intentan resolver una problemática con una estrategia en la cual se sigue una heurística o un plan para llegar al resultado deseado, esta heurística realmente solo es una manera en la que el programador decide seguir con el algoritmo no hay una necesidad (sin un contexto dado) de que este siempre resuelva el problema, por lo tanto una heurística puede fallar. Muchos problemas de la vida cotidiana se resolvieron gracias a la prueba y error de la humanidad y los algoritmos ávidos grosso modo son eso, una prueba de lo que podría ser una solución general.

2. El problema del caballo

Consiste en recorrer todas casillas del tablero N*N con un caballo de ajedrez, en N*N movimientos, lo que implica no pasar dos veces por la misma casilla.

- *Unidad Interdisciplinaria de Ingenierías campus Zacatecas*
- *Instituto Politécnico Nacional*

3. Como llegamos a nuestra heurística

las heurísticas más comentadas en el grupo eran:

- Ir recorriendo el tablero por la posición con más movimientos disponibles e
- Ir recorriendo el tablero por la posición con menos movimiento disponibles.

Al finalizar el día decidimos usar la segunda heurística.

4. Resultados

```
run:
  3  |  44  |   5  |  22  |  19  |  26  |  37  |  24  |
-----+------+------+------+------+------+------+------+
  6  |  21  |   2  |  43  |  36  |  23  |  18  |  27  |
-----+------+------+------+------+------+------+------+
 45  |   4  |  51  |  20  |  33  |  40  |  25  |  38  |
-----+------+------+------+------+------+------+------+
 52  |   7  |  42  |   1  |  50  |  35  |  28  |  17  |
-----+------+------+------+------+------+------+------+
 11  |  46  |  63  |  34  |  41  |  32  |  39  |  58  |
-----+------+------+------+------+------+------+------+
  8  |  53  |  10  |  49  |  62  |  59  |  16  |  29  |
-----+------+------+------+------+------+------+------+
 47  |  12  |  55  |  64  |  31  |  14  |  57  |  60  |
-----+------+------+------+------+------+------+------+
 54  |   9  |  48  |  13  |  56  |  61  |  30  |  15  |
-----+------+------+------+------+------+------+------+
Cantidad de veces que entra al ciclo: 64
BUILD SUCCESSFUL (total time: 1 second)
```

Figura 1. Tablero de 8, iniciando de 4,4 o D4

Vemos que tanto en el tablero de 8, 7 y 5 en posiciones dadas el algoritmo nos da una

```
run:
  1  |  30  |  3  |  34  |  23  |  40  |  27  |
-----:------:-----:------:------:------:------:
  4  |  33  |  24  |  29  |  26  |  43  |  22  |
-----:------:-----:------:------:------:------:
 31  |   2  |  35  |  48  |  39  |  28  |  41  |
-----:------:-----:------:------:------:------:
 36  |   5  |  32  |  25  |  42  |  21  |  44  |
-----:------:-----:------:------:------:------:
 11  |   8  |  47  |  38  |  49  |  18  |  15  |
-----:------:-----:------:------:------:------:
  6  |  37  |  10  |  13  |  16  |  45  |  20  |
-----:------:-----:------:------:------:------:
  9  |  12  |   7  |  46  |  19  |  14  |  17  |
-----:------:-----:------:------:------:------:
Cantidad de veces que entra al ciclo: 49
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 2. Tablero de 7, iniciando de 1,1 o A1

```
run:
  5  |  20  |  15  |  10  |  3  |
-----:------:------:------:-----:
 14  |   9  |   4  |  21  |  16  |
-----:------:------:------:------:
 19  |   6  |  23  |   2  |  11  |
-----:------:------:------:------:
 24  |  13  |   8  |  17  |  22  |
-----:------:------:------:------:
  7  |  18  |  25  |  12  |  1  |
-----:------:------:------:-----:
Cantidad de veces que entra al ciclo: 25
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 3. Tablero de 5, iniciando de 5,5 o E5

solución al problema mientras que una vez se reduce el tablero a 4, el algoritmo finaliza sin una solución al problema del caballo, esto se debe a que el tablero es tan pequeño que llega un punto que la heurística no puede generar un caso donde se pueda recorrer todas las casillas en esa posición. código en: https://github.com/Memotets/AdA/tree/master/src/Caballo

5. Conclusiones

La selección de una heurística puede parecer algo simple cuando no se tiene mucha conciencia del problema a resolver, se tiene que tener muchos factores para considerarla una buena opción para resolver un problema, la más típica es que resuelva el peor de los casos, esto no quiere decir directamente que resuelva

```
run:
  3  |  0  |  5  |  10  |
-----:-----:-----:------:
  8  |  11 |  2  |  0   |
-----:-----:-----:------:
  0  |  4  |  9  |  6   |
-----:-----:-----:------:
 12  |  7  |  0  |  1   |
-----:-----:-----:------:
Cantidad de veces que entra al ciclo: 12
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 4. Tablero de 4, iniciando de 4,4 o D4

cualquier caso, pero si lo más complicado, o quizás que se resuelvan los casos más comunes teniendo un margen de éxito mayor, siendo cualquiera de estas la meta, o incluso llegar a una solución general de problema, el uso de una heurística hace que se ataque al problema desde una perspectiva y no englobe un gran estudio de la problemática.