# Práctica 3

# Implementación

# del problema de

# las jarras

Javier Pellejero Ortega & Zhaoyan Ni

Inteligencia Artificial

Grupo 11

Doble grado Matemáticas e ingeniería informática

1. **Tabla comparativa de los resultados obtenidos con los 8 algoritmos.**
   1. **Capacidad garrafa 1 = 5, capacidad garrafa 2 = 3, cantidad objetivo = 4.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Coste del camino | Nodos explorados | Tamaño de la cola | Tamaño máximo de la cola | Tiempo de ejecución en nanosegundos |
| Búsqueda en Anchura  *(TreeSearch)* | 6 | 129 | 244 | 244 | 8702726 |
| Búsqueda en Anchura  *(GraphSearch)* | 6 | 11 | 2 | 4 | 351773 |
| Búsqueda en Profundidad  *(GraphSearch)* | 10 | 10 | 9 | 10 | 562587 |
| Coste uniforme  (*TreeSearch)* | 6 | 476 | 883 | 884 | 7301624 |
| Coste uniforme  *(GraphSearch)* | 6 | 13 | 1 | 5 | 197493 |
| Método Voraz  *(GraphSearch)* | 6 | 9 | 6 | 7 | 255489 |
| Búsqueda A\*  (*TreeSearch)* | 6 | 66 | 123 | 124 | 980568 |
| Búsqueda A\*  *(GraphSearch)* | 6 | 11 | 1 | 5 | 222531 |

* 1. **Capacidad garrafa 1 = 7, capacidad garrafa 2 = 3, cantidad objetivo = 1.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Coste del camino | Nodos explorados | Tamaño de la cola | Tamaño máximo de la cola | Tiempo de ejecución en nanosegundos |
| Búsqueda en Anchura  *(TreeSearch)* | 4 | 16 | 30 | 30 | 6165975 |
| Búsqueda en Anchura  *(GraphSearch)* | 4 | 7 | 2 | 4 | 371992 |
| Búsqueda en Profundidad  *(GraphSearch)* | 10 | 10 | 10 | 11 | 577661 |
| Coste uniforme  (*TreeSearch)* | 4 | 51 | 94 | 95 | 1741414 |
| Coste uniforme  *(GraphSearch)* | 4 | 9 | 1 | 5 | 864064 |
| Método Voraz  *(GraphSearch)* | 4 | 5 | 4 | 5 | 221509 |
| Búsqueda A\*  (*TreeSearch)* | 4 | 9 | 17 | 18 | 311185 |
| Búsqueda A\*  *(GraphSearch)* | 4 | 7 | 3 | 5 | 242460 |

* 1. **Capacidad garrafa 1 = 12, capacidad garrafa 2 = 3, cantidad objetivo = 1.**

Para estos datos, el programa no encuentra solución, puesto que no la tiene. Al ser ambas garrafas de capacidades múltiplos de 3, en la garrafa de 12 sólo podremos conseguir las cantidades de 0, 3, 6 ,9 y 12, mientras que en la garrafa de 3 sólo podremos tener 0 ó 3.

1. **Análisis de los resultados obtenidos.**

En primer lugar, podemos observar que *la búsqueda en profundidad* con *GraphSearch* que tiene en cuenta los estados repetidos no encuentra la solución óptima en los dos casos (caso a: capacidad garrafa 1 = 5, capacidad garrafa 2 = 3, cantidad objetivo = 4 y caso b: capacidad garrafa 1 = 7, capacidad garrafa 2 = 3, cantidad objetivo = 1). Esto es debido a que este método de búsqueda no incorpora conocimiento que guíe la búsqueda, se decide a priori qué camino sigue.

La *búsqueda en anchura* encuentra la solución óptima, tanto con *TreeSearch* como con *GraphSearch*. Este algoritmo explora todo el árbol de nodos hasta llegar a una solución y, además, si el coste de los operadores de búsqueda es aproximadamente uniforme la solución es óptima.

La *búsqueda de coste uniforme* considera primero los nodos que tienen menor coste hasta llegar a ellos y continua hasta encontrar la solución óptima. Este método encuentra la solución óptima si todos los operadores tienen coste mayor o igual que 0, como en este caso.

La *búsqueda A\** encuentra la solución óptima porque la heurística que hemos definido es admisible y consistente: en caso de que hemos llegado al estado objetivo, el coste estimado es nulo; el coste estimado es 1 cuando podemos llegar al estado objetivo en un paso y el coste real también es 1; en otros casos, el coste estimado es 2 y coincide con el coste real. Esto garantiza que la solución encontrada por A\* tanto con *TreeSearch* como con *GraphSearch* es óptima.

En cuando al coste de memoria, cabe la pena destacar la diferencia de memoria usada por el mismo algoritmo con el uso de *TreeSearch* y *GraphSearch*. Este último previene la repetición de estados; es decir, memoriza los estados que ya ha comprobado para no comprobarlo sucesivas veces, esto produce un uso de memoria mucho menor.

Teniendo en cuenta el algoritmo utilizado, la *búsqueda de coste uniforme* con *TreeSearch* explora mayor número de nodos puesto que es una búsqueda no informada y puede provocar una visita masiva de nodos. Sin embargo, si alguna de las primeras soluciones encontradas es óptima, el algoritmo puede realizarse de manera rápida visitando pocos nodos y utilizando poca memoria. En cambio, el *método voraz* es el que menos memoria utiliza puesto que este método busca una solución directa cogiendo los mejores candidatos sin tener en cuenta el coste de llegar a los estados previos. Así logra una solución rápida, visitando pocos nodos, pero pocas veces óptima.

Entre las búsquedas informadas, podemos observar que la *búsqueda A\** explora más nodos que el *método voraz* pues este método busca candidatos óptimos, pero siempre teniendo en cuenta el coste previo de llegar al estado en cuestión. Esto hace considerar más estados que el método voraz a cambio de encontrar una solución óptima.

Desde punto de vista del tiempo de ejecución, en el caso a, el método que ha tardado menos tiempo es la *búsqueda de coste uniforme* con el uso de *GraphSearch* que ha tardado 197493 nanosegundos y en el caso b, es el *método voraz*, que ha tardado 221509 nanosegundos. Esto es debido probablemente a que la *búsqueda de coste uniforme* ha encontrado una primera solución óptima y el *método voraz* encuentra rápidamente una solución es debido a la razón anteriormente mencionada. El método de *búsqueda en anchura* con *TreeSearch* es el método que ha tardado más tiempo en ambos casos, ha tardado 8702726 nanosegundos y 6165975 nanosegundos respectivamente. Es normal que este método tarda mucho puesto que se recorre sin ningún tipo de criterio de elección, todos los hijos de un estado, previa visita previa de todos los hermanos del primero, por tanto, añade muchos nodos a visitar tras analizar alguno de ellos (sobre todo con *TreeSearch*).