Universidad San Francisco de Quito
Maestría en Inteligencia Artificial
Módulo: Inteligencia artificial
Taller 2

Edwin Montenegro, Alejandra Pinto, Gustavo Recalde, Galo Travez

02 de marzo del 2024

Índice

1. Uso de algoritmos de búsqueda

Hay varios algoritmos que se utilizan para buscar caminos en laberintos o laberintos. Estos algoritmos aprovechan los conceptos de la teoría de grafos, ya que los laberintos se pueden representar como grafos. Estos son algunos de los algoritmos más utilizados:

- Búsqueda de amplitud (BFS): Este algoritmo explora el laberinto nivel por nivel, asegurándose de que se visiten todos los nodos a una profundidad determinada antes de pasar al siguiente nivel. BFS es eficaz para encontrar el camino más corto en grafos no ponderados, como laberintos simples.
- Búsqueda en profundidad (DFS): DFS explora lo más lejos posible a lo largo de cada rama antes de retroceder. A menudo se usa en algoritmos de resolución de laberintos.
- Algoritmo de Dijkstra: Este algoritmo encuentra la ruta más corta en un gráfico ponderado. Si bien no se menciona específicamente en los resultados de búsqueda, el algoritmo de Dijkstra se aplica comúnmente en escenarios de resolución de laberintos, especialmente cuando se consideran los costos variables para diferentes rutas.
- Un algoritmo:* A* es un algoritmo popular de búsqueda de rutas que combina elementos tanto del algoritmo de Dijkstra como de la búsqueda codiciosa de lo mejor primero. Considera tanto el costo para llegar al nodo actual como una estimación de la distancia restante hasta la meta.

1.1. Procedimiento

Para el desarrollo de la primera sección del taller, el procedimiento fue el siguiente:

1. Diseño de Laberintos: Existen tres laberintos distintos, en los cuales se tiene caracteres o valors asignados así:

Cuadro 1: Condiciones iniciales

Descripción	**A***	**BFS - DFS**
Pared u obstáculo	#	1
Posibles rutas	Espacio vacio	0
Entrada	E	1-0
Salida	S	0-1

2. Selección de Algoritmos: Se deben seleccionar tres algoritmos de búsqueda diferentes para la resolución de los laberintos; en este caso se aplicó Método A*, Algoritmo de búsqueda en anchura (BFS), Algoritmo de búsqueda en profundidad (DFS).
3. Implementación en Python: Se debe escribir el código en Python para cada algoritmo de búsqueda seleccionado. Es necesario implementar funciones que tomen el laberinto como entrada y devuelvan el camino encontrado junto con la métrica de rendimiento. Para el método de A* se establecen las siguientes funciones

    a. Establece heurística de Manhattan.

    b. Define los parámetros de Inicio y fin del laberinto.

    c. Define costos para elegir dirección y movimiento.

    d. Marca las paredes y el camino más óptimo.

    Para los métodos BFS y DFS las funciones son:

    a. Transformar el laberitno en un grafo(Adyacencias).

    b. Realizar búsqueda en anchura y/o profundidad.

    c. Visualización de ruta.

4. Evaluación de Desempeño: Se debe medir el tiempo que tarda cada algoritmo en encontrar la solución; así también se debe calcular la longitud del camino encontrado por cada algoritmo.
5. Medición de Resultados: Se deben ejecutar cada algoritmo en los tres laberintos y registrar los tiempos y las distancias.
6. Análisis de Resultados: Se deben comparar los resultados obtenidos para cada algoritmo y laberinto. Se debe analizar cuál algoritmo fue más rápido y cuál encontró el camino más corto.
7. Documentación: Se debe documentar el código y los resultados obtenidos. Se puede usar Jupyter Notebook o un archivo .py para una presentación interactiva.
8. Visualización: Se puede implementar una visualización para mostrar los laberintos y los caminos encontrados por cada algoritmo. Librerías como matplotlib pueden ser útiles para esto.
9. Conclusión: Se debe redactar una conclusión basada en los datos recopilados que resuma el desempeño de cada algoritmo en los diferentes laberintos.

1.2. Resultados

Se obtuvo los siguientes resultados:

1.2.1. Laberinto 1

Para la resolución del primer laberinto por A*, la figura muestra el resultado, con un tiempo de ejecución de 0.000997 segundos y 14 pasos hasta el destino.

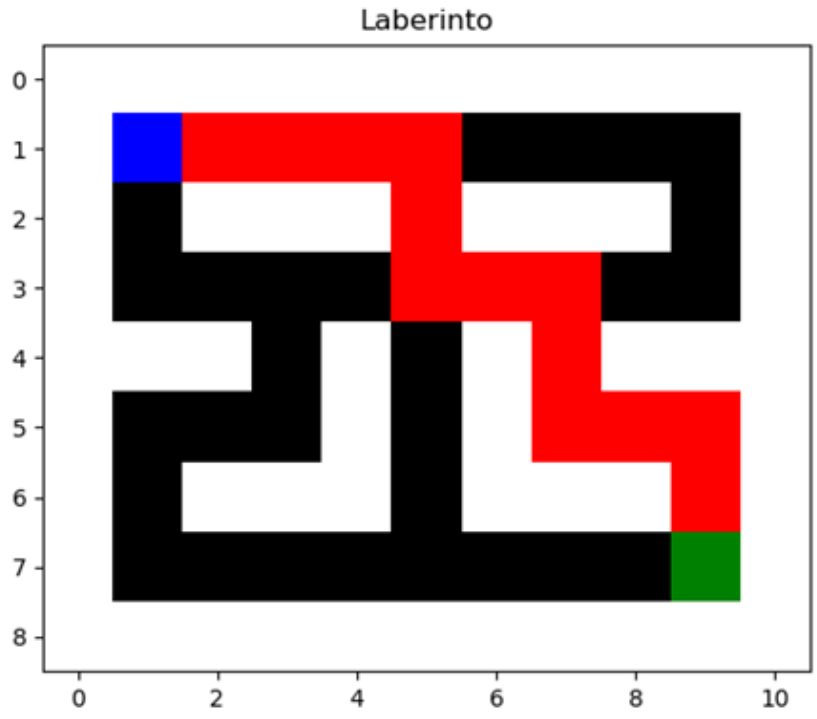

Figura 1: Resolución por A*

Para la resolución del primer laberinto por BFS, la figura muestra el resultado, con un tiempo de ejecución de 0.000992 segundos y 14 pasos hasta el destino.

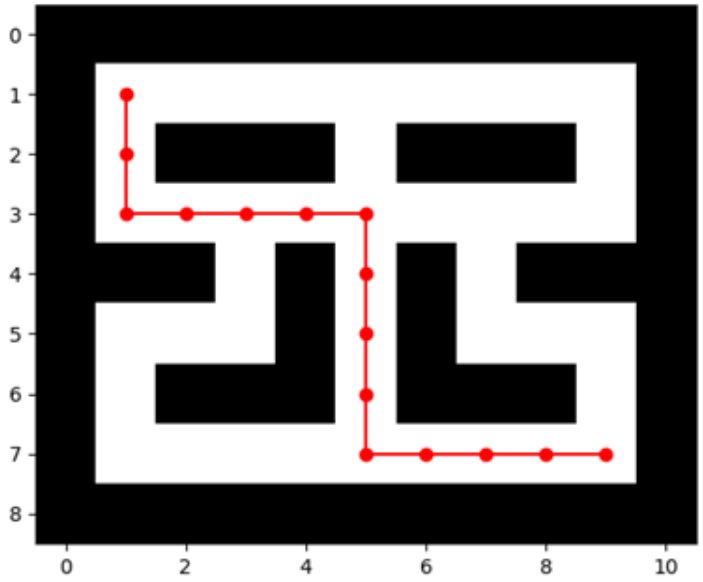

Figura 2: Resolución por BFS

Para la resolución del primer laberinto por DFS, la figura muestra el resultado, con un tiempo de ejecución de 0.0000001 segundos y 34 pasos hasta el destino.

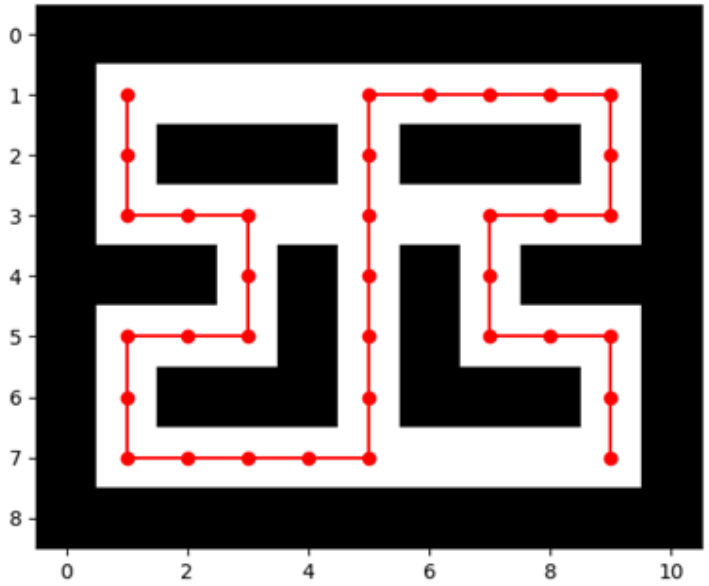

Figura 3: Resolución por DFS

1.2.2. Laberinto 2

Para la resolución del primer laberinto por A*, la figura muestra el resultado, con un tiempo de ejecución de 0.0009965896606445312 segundos y 44 pasos hasta el destino.

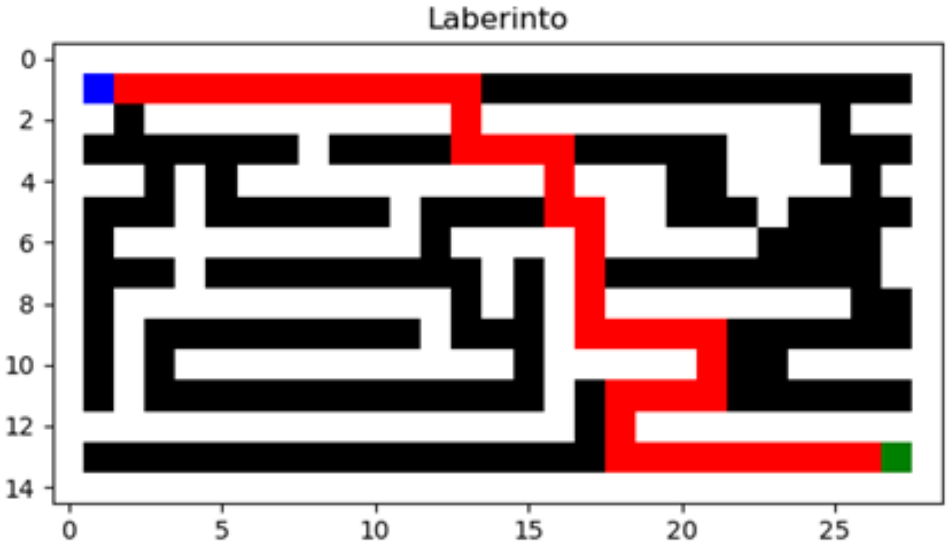

Figura 4: Resolución por A*

Para la resolución del primer laberinto por BFS, la figura muestra el resultado, con un tiempo de ejecución de 0.000998 segundos y 44 pasos hasta el destino.

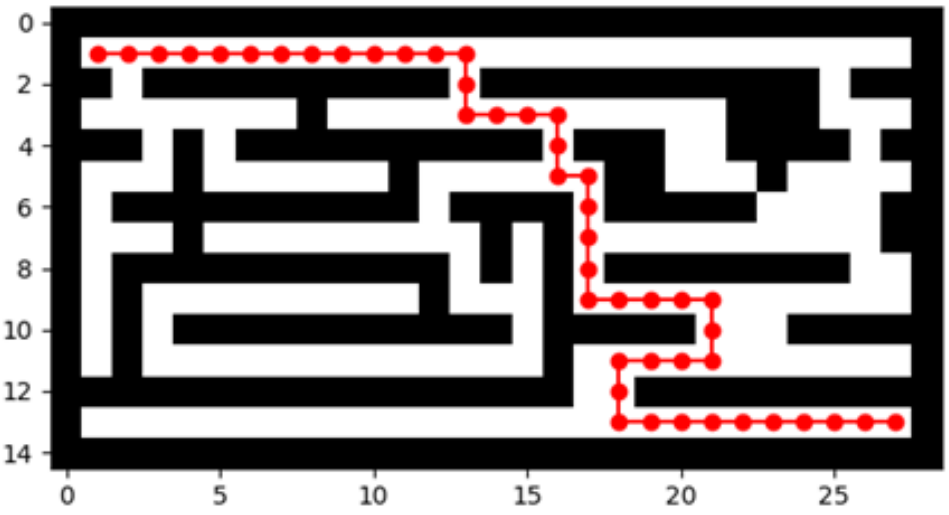

Figura 5: Resolución por BFS

Para la resolución del primer laberinto por DFS, la figura muestra el resultado, con un tiempo de ejecución de 0.000997 segundos y 44 pasos hasta el destino.

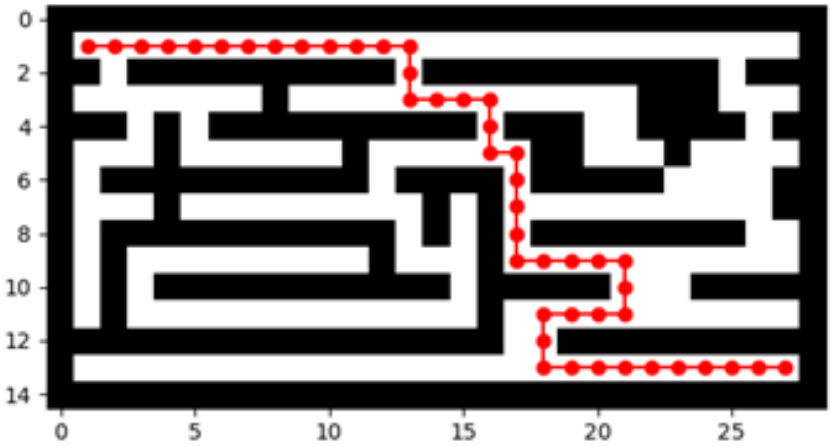

Figura 6: Resolución por DFS

1.2.3. Laberinto 3

Para la resolución del primer laberinto por A*, la figura muestra el resultado, con un tiempo de ejecución de 0.003989219665527344 segundos y 344 pasos hasta el destino.

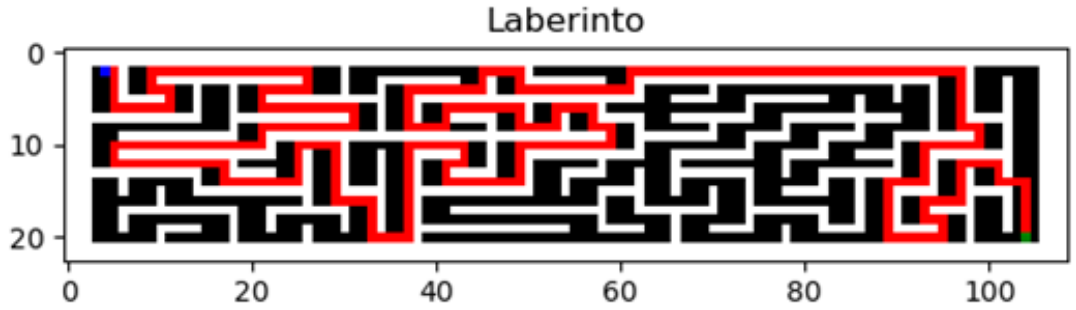

Figura 7: Resolución por A*

Para la resolución del primer laberinto por BFS, la figura muestra el resultado, con un tiempo de ejecución de 0.006982 segundos segundos y 344 pasos hasta el destino.

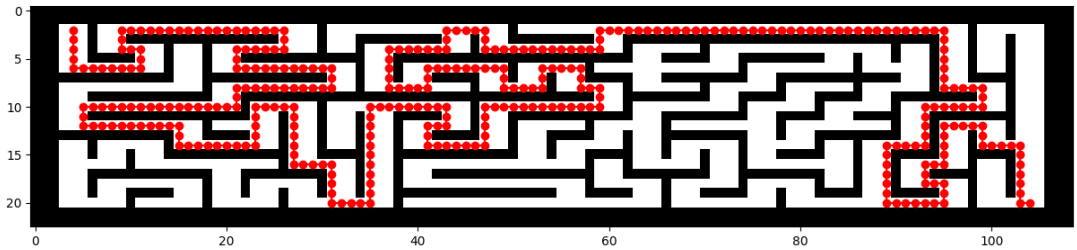

Figura 8: Resolución por BFS

Para la resolución del primer laberinto por DFS, la figura muestra el resultado, con un tiempo de ejecución de 0.004988 segundos y 498 pasos hasta el destino.

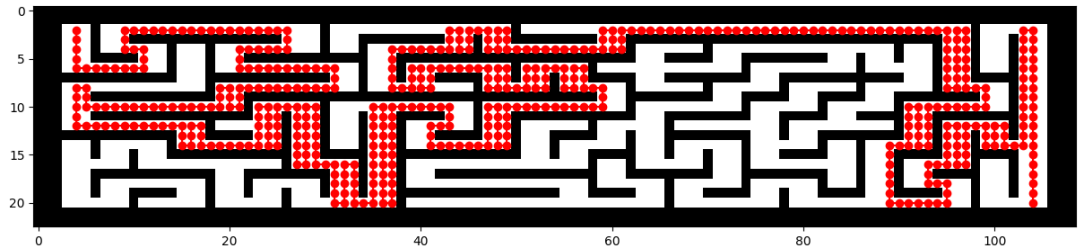

Figura 9: Resolución por DFS

1.3. Comparativa

Maze	A* Time (s)	BFS Time (s)	DFS Time (s)	A* Steps	BFS Steps	DFS Steps
1	0.000997	0.000992	0.00000015	14	14	34
2	0.000997	0.000998	0.000997	44	44	44
3	0.003989	0.004988	0.006982	344	344	498

- A* supera a BFS y DFS, mostrando ser eficiente y rápido en los tres laberintos.
- A* también destaca al realizar menos pasos en comparación con BFS y DFS, evidenciando su eficacia en encontrar la ruta más corta.
- La variedad de métodos ofrece flexibilidad para diferentes escenarios, desde laberintos simples hasta aquellos con costos variables en las rutas.
- Específicas: La elección del método depende de las necesidades específicas del problema, ya sea eficiencia, camino más corto o consideración de costos.
- A* destaca por su eficiencia, pero puede tener una mayor complejidad de implementación en comparación con BFS y DFS.

2. Optimización de colonia de hormigas

2.1. Ejecutar el lcódigo planteado

Analizando el código para el primer caso, la elección de los parámetros de la traza de feromonas (alfa) y de la heurística de distancia (beta) es lo más importante para el desempeño del algoritmo. Valores altos de beta favorecen la exploración de caminos cortos, mientras que valores altos de alpha pueden llevar a una rápida convergencia a soluciones menos óptimas si las feromonas se concentran demasiado en ciertos caminos.

2.2. ¿Qué ocurre con e segundo caso de estudio?

- En el segundo caso, el método find_best_path actualiza el mejor camino basándose únicamente en la longitud del camino, sin considerar la intensidad acumulada de las feromonas en ese camino, lo cual genera que retorne al inicio y no encuentre el camino. Es decir, aunque se depositaran feromonas en el mejor camino encontrado en cada iteración, la decisión sobre cuál era el mejor camino se tomaba solo por su longitud, lo que no siempre es la la mejor métrica cuando hay múltiples caminos de igual longitud pero con diferentes intensidades de feromonas.
- La solución seria considerar esta intensidad y luego ordenar los caminos primero por su longitud mas corta y, en caso de igualdad en la longitud, por la suma negativa de las intensidades de las feromonas a lo largo del camino (prefiriendo caminos con mayor intensidad de feromonas). Con esto priorizamos caminos más cortos pero, y en caso de caminos de igual longitud, elige aquellos con mayor intensidad de feromonas.
- Para esto modificamos el codigo con este criterio en el codigo:

  $ll\_paths.sort(key = lambdax : (len(x[0]), -x[1]))$

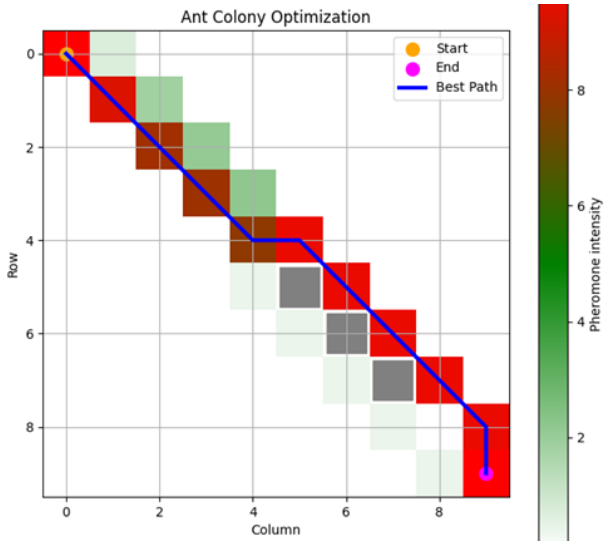

Figura 10: Modelo ACO

2.3. Parámetros del modelo

Se revisa cada uno de los parámetros así:

- num_ants (número de hormigas): Especifica el número de hormigas que participarán en la búsqueda del camino. Un número mayor de hormigas puede aumentar la posibilidad de encontrar un camino eficiente, pero también incrementa el costo computacional.
- evaporation_rate (tasa de evaporación): Define la tasa a la que se evaporan las feromonas dejadas por las hormigas en la cuadrícula. Este parámetro ayuda a evitar la convergencia prematura hacia soluciones subóptimas y fomenta la exploración.
- alpha (alfa): Es un parámetro que controla la influencia de las feromonas en la decisión de movimiento de las hormigas. Un valor más alto hace que el camino con más feromonas sea mucho más atractivo.
- beta (beta): Controla la influencia de la heurística (en este caso, la distancia al objetivo) en la decisión de movimiento de las hormigas. Un valor más alto favorece las direcciones que están más cerca del objetivo, según la heurística.
- pheromones (feromonas): Es una matriz que representa la intensidad de las feromonas en cada posición de la cuadrícula. Inicialmente, se establece en un valor uniforme en todas las posiciones.

2.4. Pregunta de investigación: ¿Será que se puede utilizar este algoritmo para resolver el Travelling Salesman Problema (TSP)?

Sí, el algoritmo de colonia de hormigas puede ser utilizado para resolver el problema de TSP, ya que es un problema clásico en la optimización combinatoria que consiste en encontrar la ruta más corta que permite visitar una serie de ciudades y volver al punto de partida.

El algoritmo de colonia de hormigas es especialmente adecuado para este tipo de problemas debido a su capacidad para explorar eficientemente el espacio de soluciones y converger hacia la solución óptima. En el contexto del TSP, las "hormigas" son utilizadas para explorar diferentes rutas entre las ciudades, depositando "feromonas" en las rutas que resultan ser más cortas. A medida que más hormigas siguen estas rutas, la cantidad de feromonas aumenta, lo que a su vez atrae a más hormigas hacia estas rutas.

Sin embargo, es importante tener en cuenta que los parámetros del algoritmo (como el número de hormigas, la tasa de evaporación de las feromonas, y los parámetros alfa y beta que controlan la importancia relativa de las feromonas y la heurística de distancia) deben ser cuidadosamente ajustados para asegurar un buen rendimiento. Además, el algoritmo de colonia de hormigas no garantiza que se encontrará la

solución óptima global, especialmente para instancias grandes del TSP. En su lugar, proporciona una solución que es una buena aproximación al óptimo global.

Por lo tanto, aunque el algoritmo de colonia de hormigas puede ser una herramienta útil para resolver el TSP, también es importante considerar otras técnicas de optimización, dependiendo de las características específicas del problema.

El proceso es el siguiente:

- Inicialización: Se coloca una cantidad de feromonas uniforme en todos los caminos posibles.
- Construcción de soluciones: Las hormigas construyen soluciones visitando ciudades, prefiriendo caminos con feromonas más intensas y considerando la distancia entre ciudades (heurística).
- Actualización de feromonas: Después de que todas las hormigas han construido soluciones, se actualiza la cantidad de feromonas en los caminos en función de la calidad de las soluciones encontradas.
- Evaporación: Se aplica una tasa de evaporación para simular el proceso natural de disminución de feromonas con el tiempo.

3. Conclusiones

- La Búsqueda en Amplitud (BFS) destaca en laberintos simples al encontrar el camino más corto en grafos no ponderados. Por otro lado, la Búsqueda en Profundidad (DFS) es comúnmente utilizada, aunque su tiempo de ejecución puede variar, siendo más rápido en ciertos casos.
- A* emerge como un algoritmo versátil. Su capacidad para considerar tanto el costo actual como una estimación de la distancia restante lo posiciona como una opción eficaz en laberintos complejos.
- La elección entre BFS, DFS o A* depende de las características específicas del laberinto. La eficiencia, el tiempo de ejecución y la complejidad de implementación son factores cruciales.
- Los algoritmos de búsqueda ofrecen flexibilidad para adaptarse a distintos tipos de laberintos. La selección del método adecuado implica considerar la topología del laberinto y los requisitos de optimización.
- La optimización por colonias de hormigas (ACO) demuestra su eficacia como una metaheurística basada en el comportamiento real de estos insectos.
- La ACO se aplica con éxito en la planificación de la fuerza de trabajo en empresas, mostrando su versatilidad en la resolución de problemas complejos.
- La ACO se emplea en la planificación de celdas de manera discreta, destacando su aplicación en escenarios específicos de optimización.
- La ACO se integra en modelos de optimización matemática, proporcionando soluciones robustas y adaptativas.