Nombres:

Cesar Carrera 00344613

Edgar Guzmán 00344822

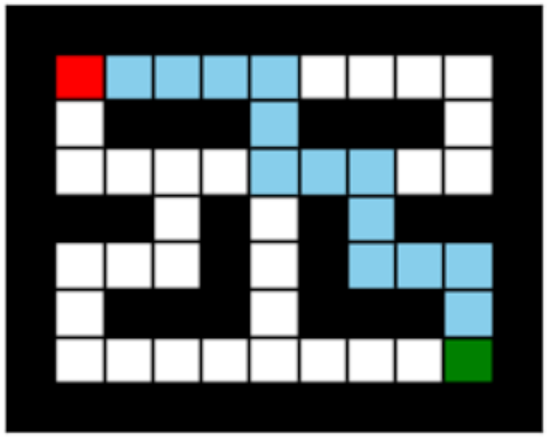
Juan Diego Sánchez 00344455

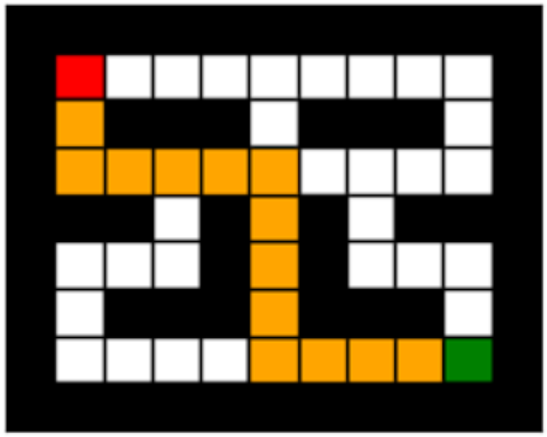
Byron Vinueza 00345908

Taller #2

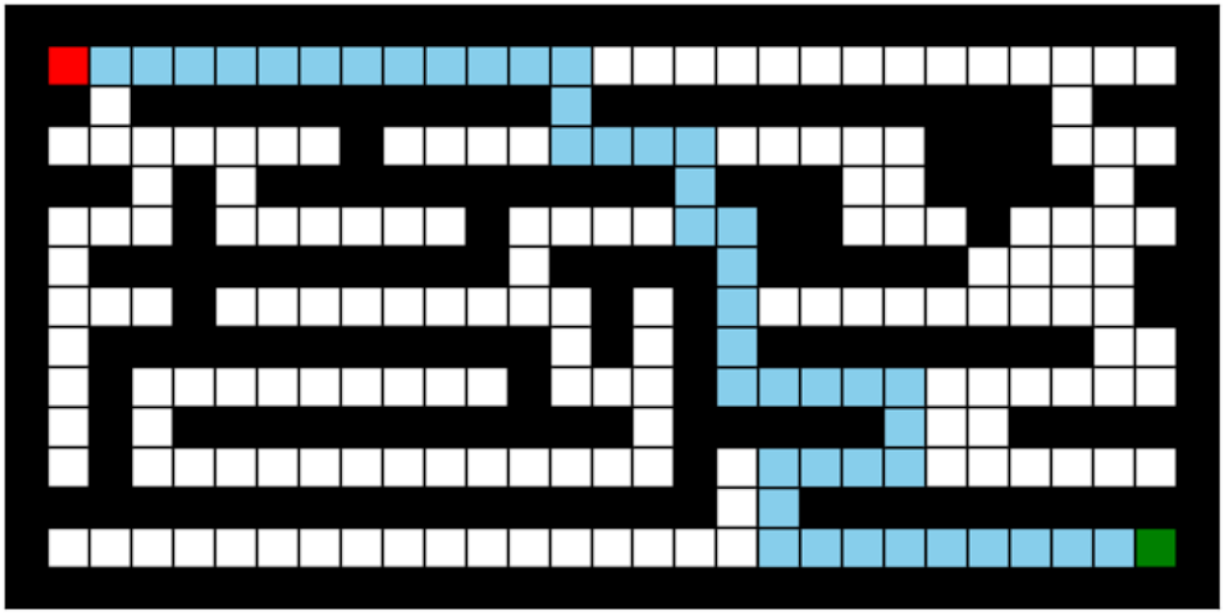
Algoritmos de búsqueda en laberintos

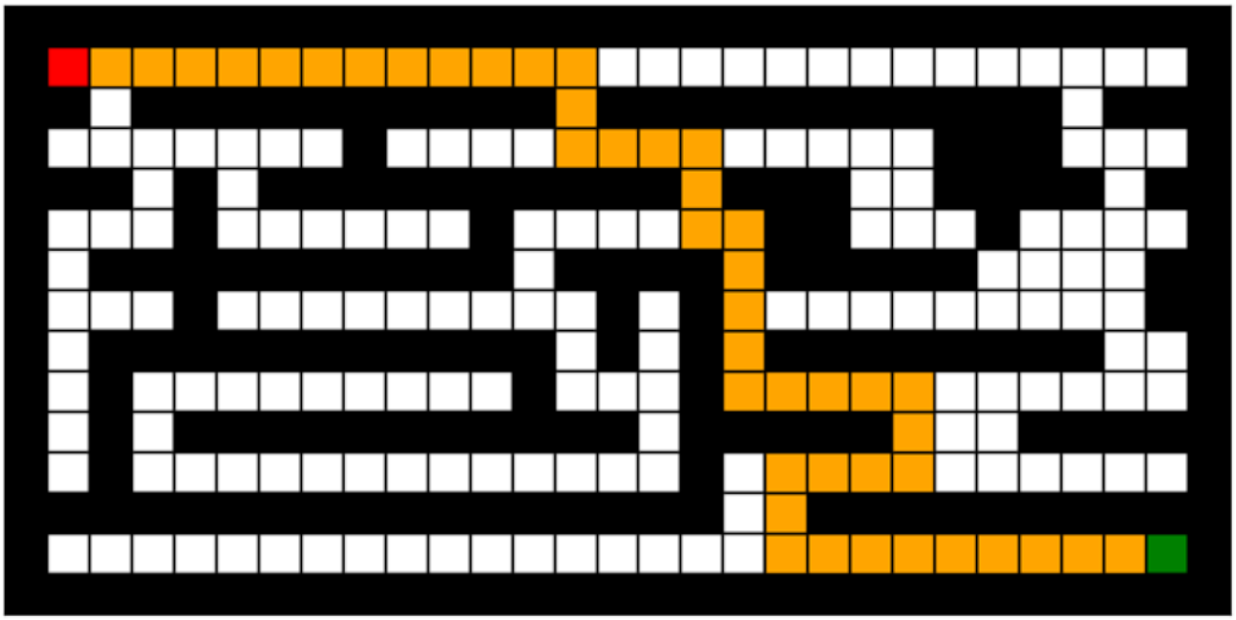
Comparación de Algoritmos:
BFS → Tiempo: 0.000049s, Nodos: 43, Longitud: 15
A*   → Tiempo: 0.000058s, Nodos: 33, Longitud: 15



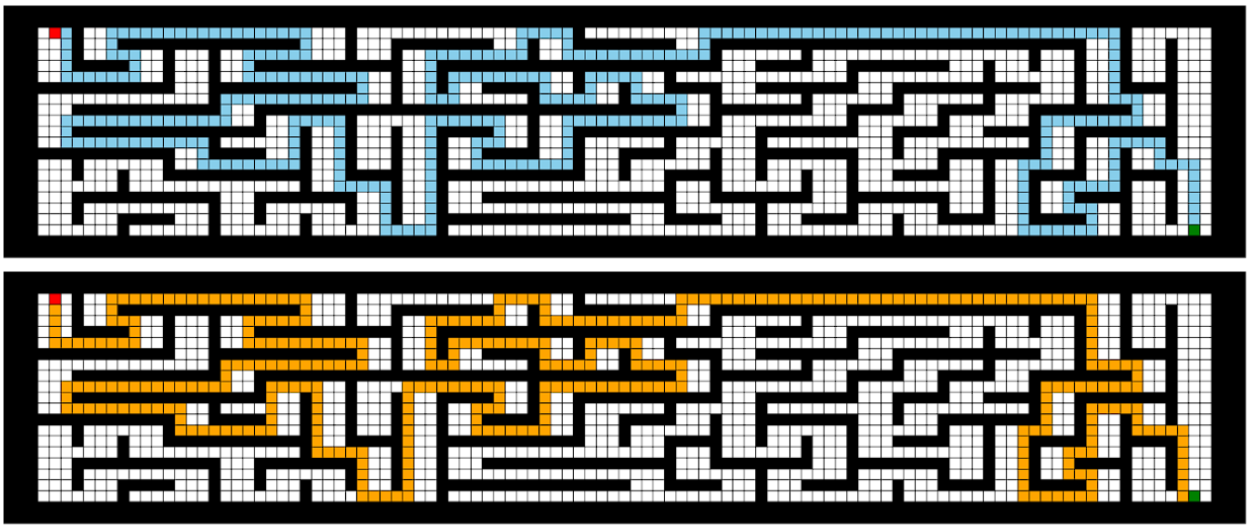


Comparación de Algoritmos:
BFS → Tiempo: 0.000517s, Nodos: 158, Longitud: 45
A*   → Tiempo: 0.000341s, Nodos: 72, Longitud: 45





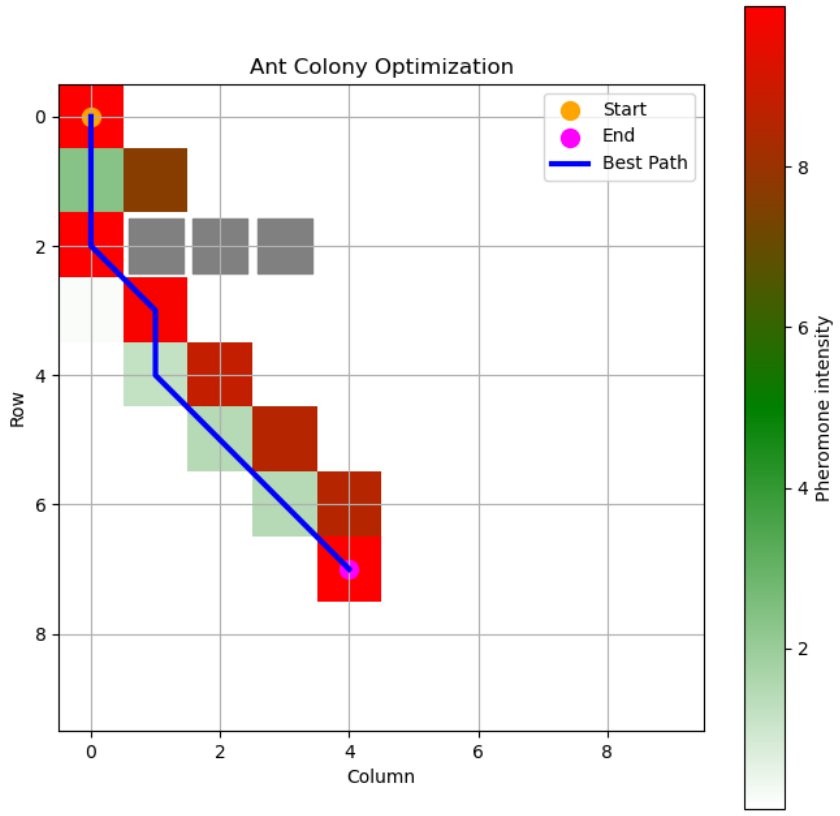
```
Comparación de Algoritmos:
BFS   → Tiempo: 0.003794s, Nodos: 1268, Longitud: 345
A*    → Tiempo: 0.006630s, Nodos: 1262, Longitud: 345
```



Las métricas utilizadas para comparar ambos algoritmos de búsqueda son importantes ya que nos permite evaluar la efectividad de cada uno de ellos. Tanto el tiempo como el número de nodos recorridos son métricas necesarias para saber al final cuál de ellos se deben usar para un óptimo rendimiento.

Por un lado, el tiempo de ejecución nos permite medir la eficiencia práctica en tiempo real. El número de nodos recorridos también es importante saber porque aquel algoritmo que expande el menor número de nodos es el que generalmente consume menos memoria y CPU.

Optimización de colonia de hormigas



Parámetros definidos:

- Inicio: (0, 0)
- Fin: (4, 7)

Obstáculos: tres posiciones bloqueadas en una fila vertical:

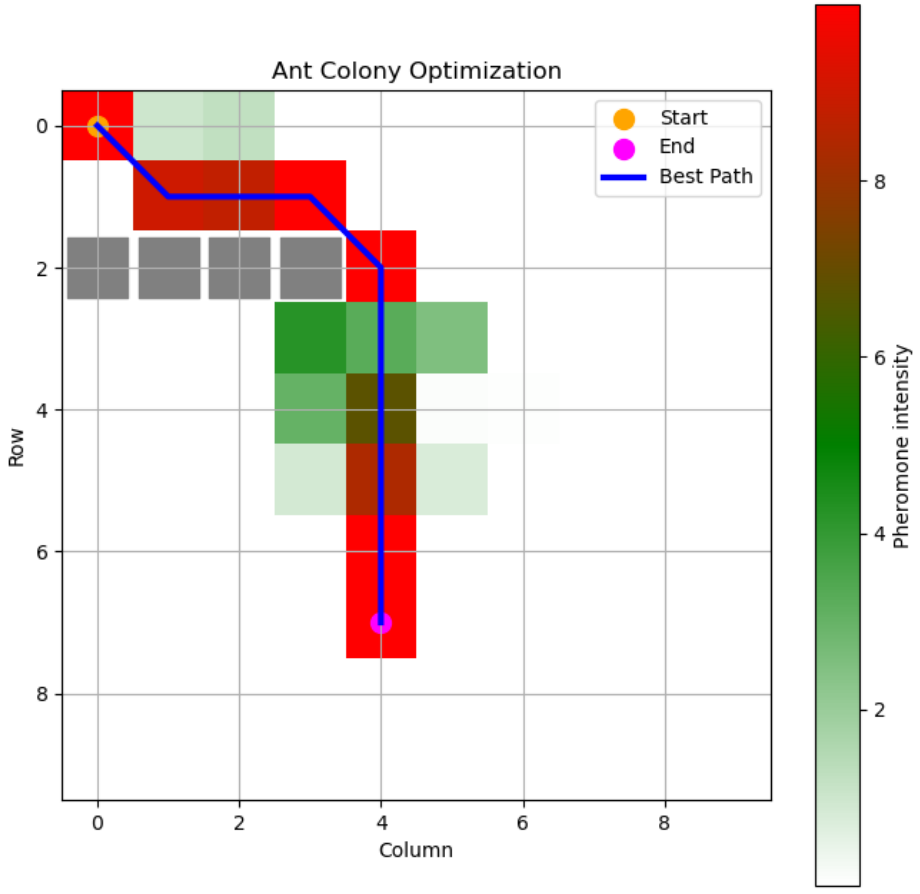
- (1,2), (2,2), (3,2)
- Iteraciones: 100
- Tamaño de la grilla: por defecto (10,10)
- Número de hormigas: 10
- Tasa de evaporación de feromonas: 0.1
- α (influencia de la feromona): 0.1 → poca importancia
- β (influencia de la heurística): 15 → muy importante

Inicialización:

- Se prepara una grilla de 10x10 con todas las celdas disponibles, excepto los obstáculos.
- Se asigna un nivel de feromona inicial de 1 en todas las celdas.

Búsqueda iterativa (100 ciclos):

- En cada iteración:
- 10 hormigas son lanzadas desde el inicio.
- Cada hormiga construye un camino hasta el nodo final o se detiene si no puede avanzar.
- Los caminos encontrados se almacenan.
- Se selecciona el mejor camino (el más corto) entre todos.
- Se aplica evaporación de feromonas en toda la grilla.
- Se deposita feromona adicional en el mejor camino.
- Se actualiza self.best_path si este nuevo camino es mejor o igual al anterior.



1.Problema:

- El algoritmo ACO no encontraba un camino hasta el final (self.end) en el caso de estudio 2.
- El gráfico mostraba que las hormigas daban vueltas sin llegar.

2. Código original

- En el método find_best_path, se elegía el camino más corto de todos, sin importar si llegaba o no al destino**.**

3. Solución

- Se añadio una validación para quedarse solo con caminos que llegaban a self**.**end**.**
- Si no hay ningún camino válido, la iteración se ignora (no se reforzaba nada)**.**
- Se selecciona el camino más corto entre los válidos**.**
- Se reforza con feromonas solo esos caminos válidos, y se evitaba aprender caminos incompletos**.**

4. Resultado

- Ahora el algoritmo encuentra correctamente caminos que llegan al destino**.**
- Ya no se refuerzan trayectorias inútiles**.**
- El mejor camino se actualiza solo si es igual o mejor que el anterior**.**

Conclusión:

- El problema era lógico, no de parámetros.
- La solución correcta fue aplicar una validación del destino antes de elegir el mejor camino.
- No se hizo ningún cambio en los parámetros. Esto no fue necesario para resolver el problema,
- aunque sí podría usarse para mejorar la eficiencia o calidad del resultado más adelante.