Discusión Proyecto: Laberinto

Desafíos de programación

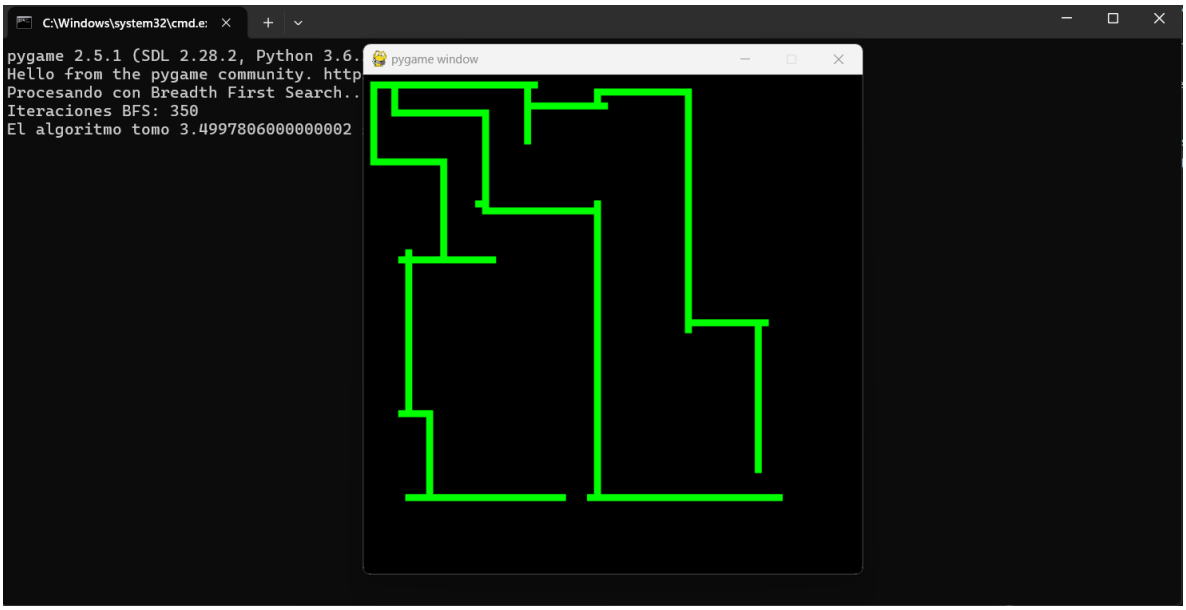
Uno de los principales desafíos fue el modelar el problema en datos que pudieran leerse desde archivos de excel, con el objetivo de continuar utilizando la misma implementación realizada para el laboratorio no. 1 del curso. Para esto se hizo un planteamiento en el que las posiciones de la matriz en donde se considerara "camino", fuesen nodos. Así mismo, los nodos izquierdo, derecho, arriba y abajo que también fueran considerados "camino" serían los nodos "vecinos". De igual forma, las heurísticas fueron calculadas según las posiciones en la matriz.

Otro desafío fue la parte visual, puesto que el utilizar librerías como pyplot de matplot no permitía la visualización en tiempo real de la manera óptima. Para esto se utilizó pygame, librería utilizada para videojuegos en python. También fue un desafío el definir cuando y como se irían modificando las posiciones del laberinto, de manera que esto funcionara a una velocidad entendible. Modificando el clock.tick() de la ventana de pygame, así como también utilizando delay se logró simular el funcionamiento del algoritmo sobre el algoritmo de buena manera.

¿En qué casos es mejor cada algoritmo?

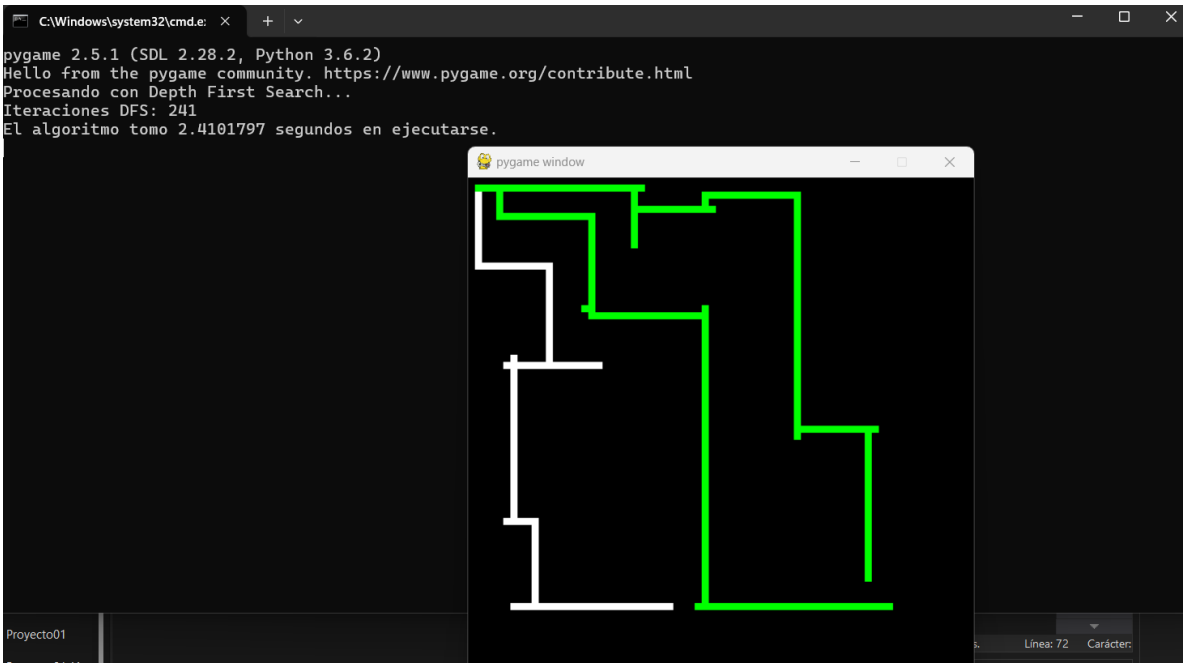
Los algoritmos BFS, DFS y DDS son sensibles al orden en que son colocados los nodos vecinos. En el caso de este proyecto, el orden en que fueron colocados es izquierdo, derecho, arriba y abajo. Para realizar las comparativas de cada uno de los algoritmos, se tomará como referencia el archivo "Prueba_1.txt".

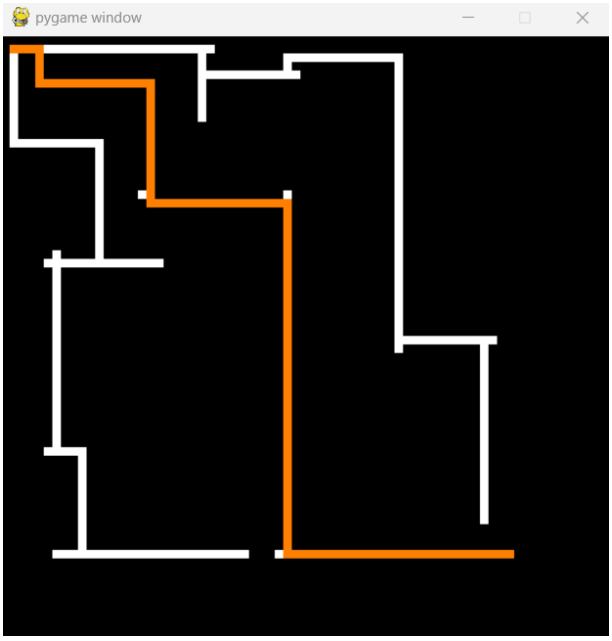
BFS



El algoritmo resuelve el laberinto aparentemente de forma paralela, puesto que da prioridad de visita a los hijos de un nodo, antes que a los hijos de sus hijos. Por lo tanto, es posible inducir que el algoritmo BFS es mejor cuando la meta se encuentra cercana al inicio, sin importar la posición.

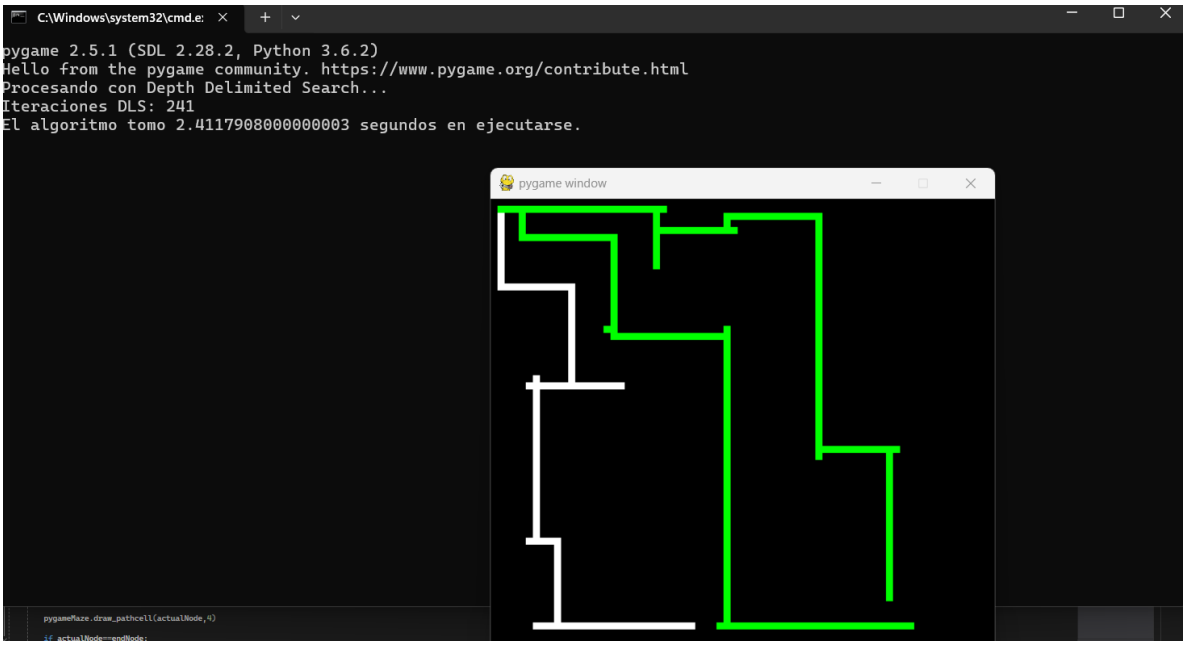
DFS

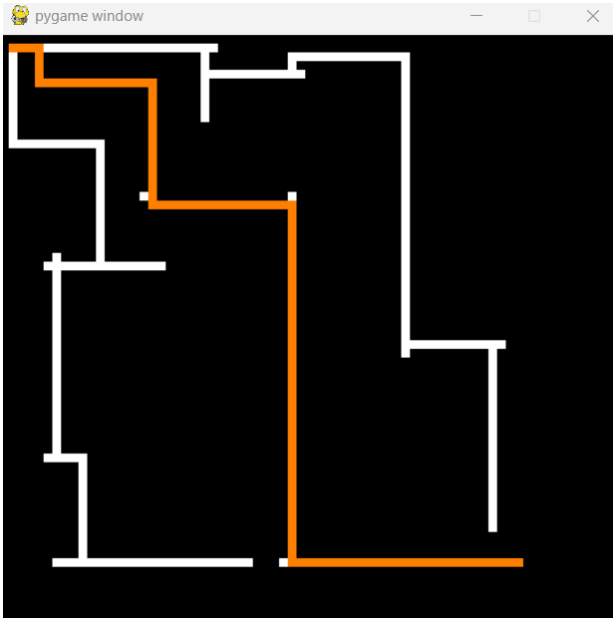




Dado que el orden en que fueron colocados los nodos vecinos de cada nodo comienza por izquierda y luego por derecha, este algoritmo da prioridad a recorrer todos los nodos vecinos derechos en este caso. Por esta razón, el algoritmo revisa todo el camino superior y posteriormente el camino de en medio. Es posible inducir que el algoritmo DFS es mejor cuando la meta se encuentra a partir de un camino que puede ser recorrido por los nodos izquierdos vecinos desde el inicio.

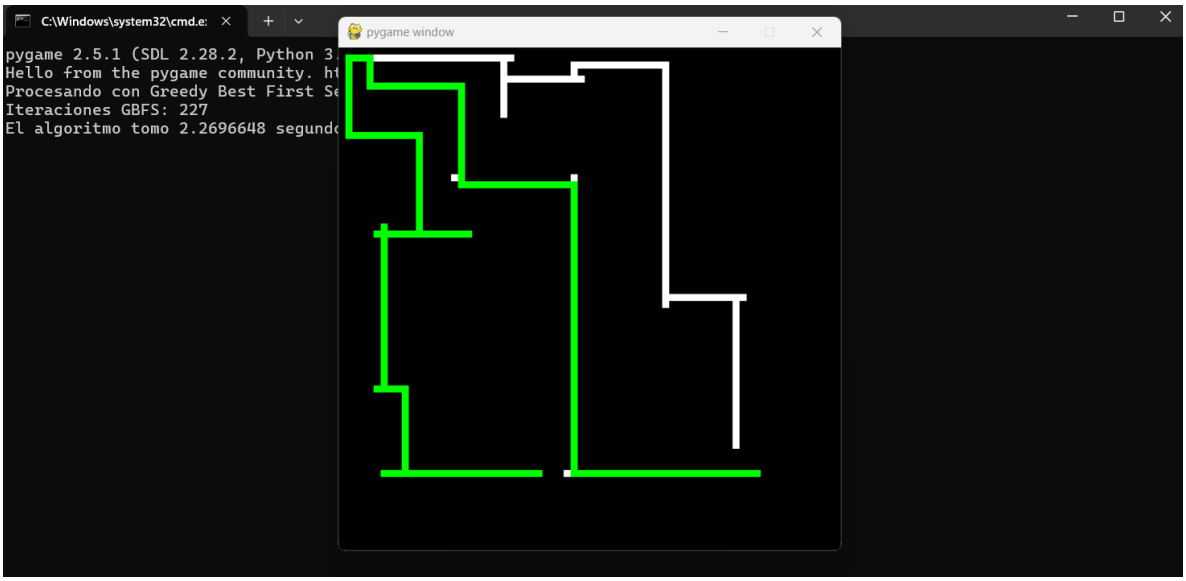
DDS



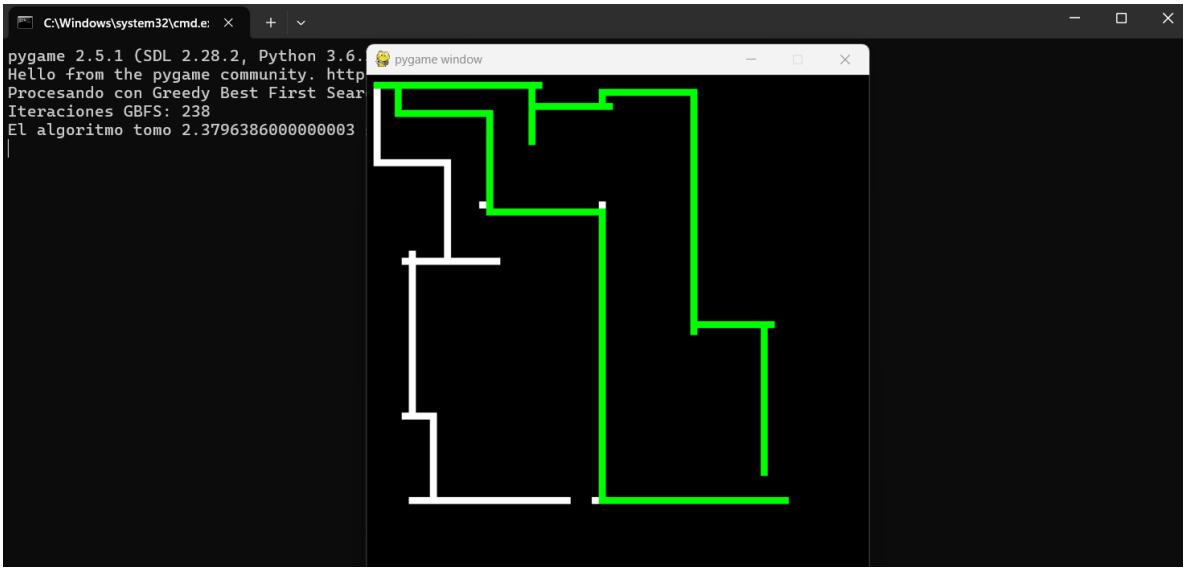


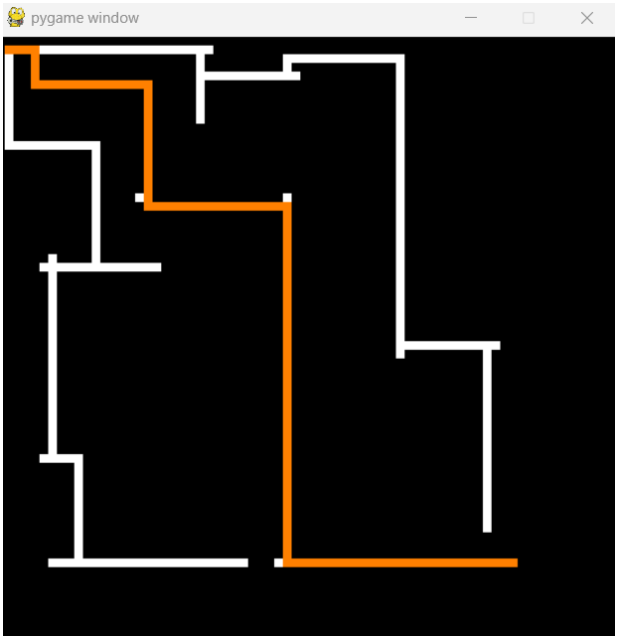
Este algoritmo funciona de la misma manera en que funciona DFS. Por lo que es posible inducir que el algoritmo DDS es mejor cuando la meta se encuentra a partir de un camino que puede ser recorrido por los nodos izquierdos vecinos desde el inicio, siempre y cuando se encuentre dentro del límite establecido para el algoritmo.

GBFS Euclidiana



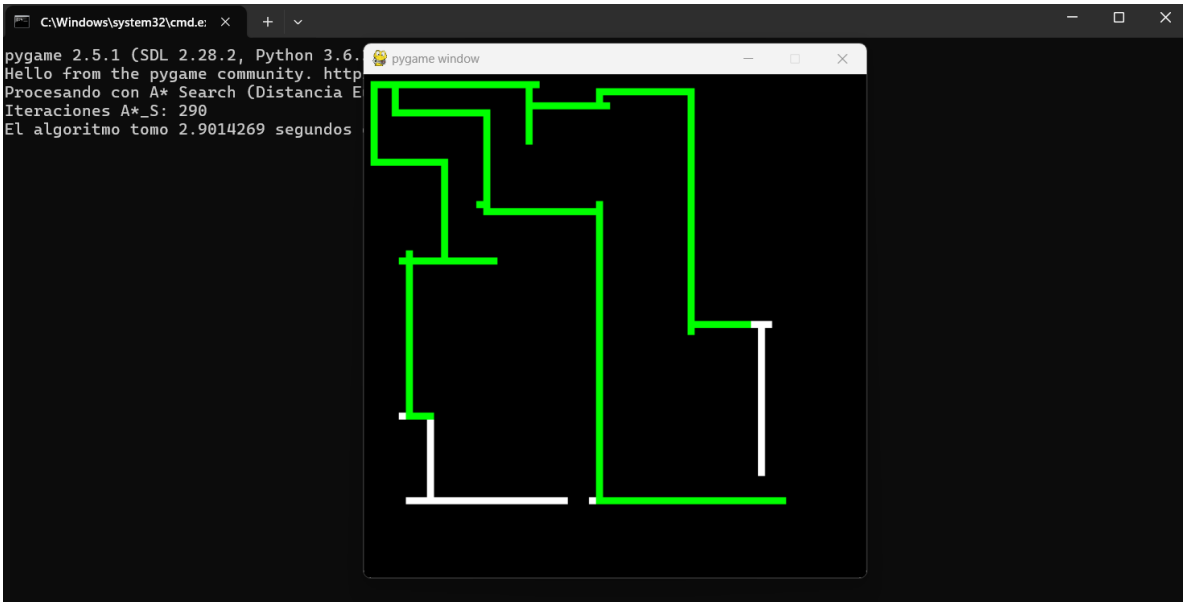
GBFS Manhattan

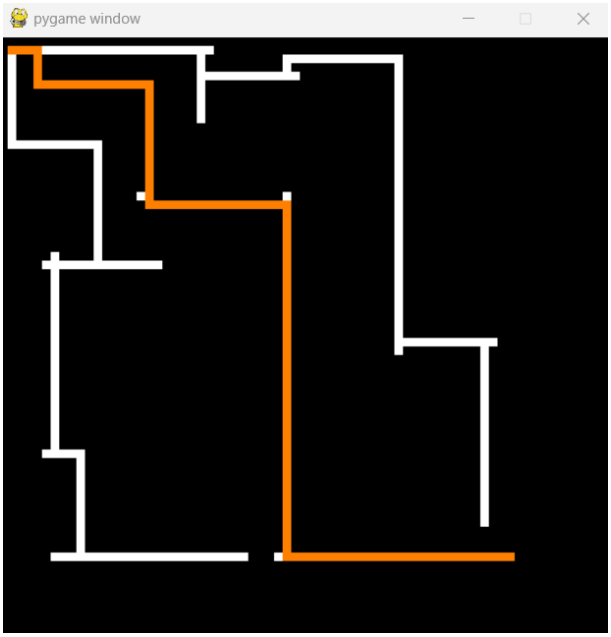




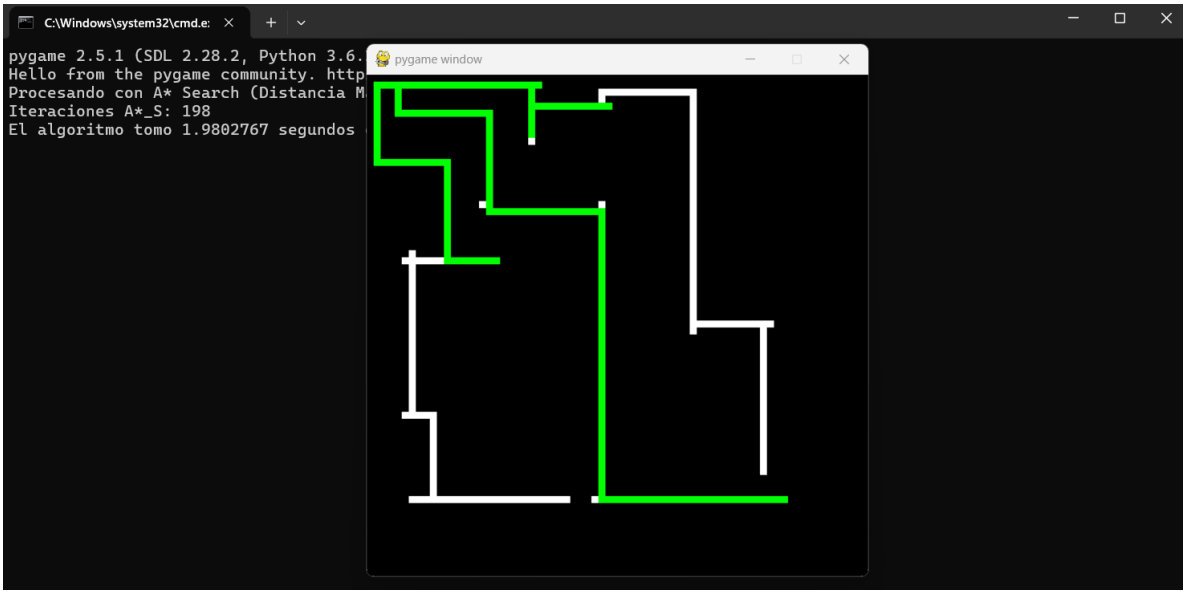
La base del algoritmo Greedy es tomar el nodo vecino con la mejor heurística, en este caso la menor distancia euclidiana o manhattan hasta la meta, según sea el caso utilizado. Por lo tanto, este algoritmo da prioridad a seleccionar la mejor solución en ese momento, sin considerar las consecuencias futuras de dicha elección. Esto se refleja en que primero fue revisado el camino inferior, y posteriormente el camino de en medio para ambas heurísticas. Por lo tanto es posible inducir que el algoritmo GBFS es mejor cuando las mejores soluciones inmediatas llevan directamente a la meta.

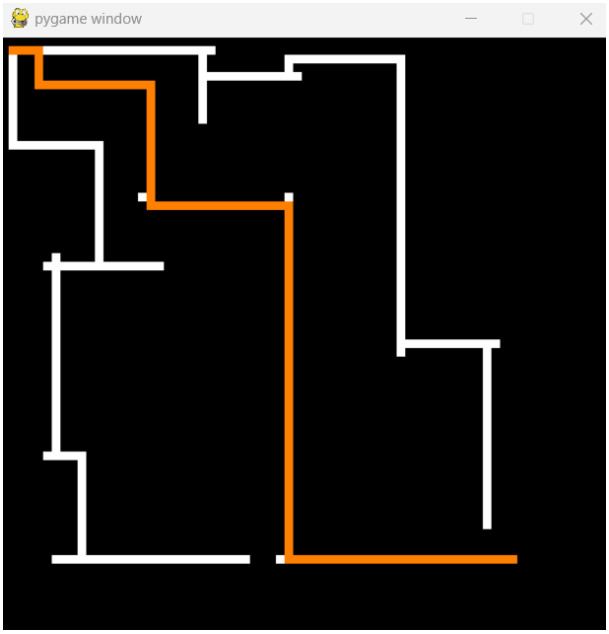
A* Euclidiana




pygame window

A* Manhattan


C:\Windows\system32\cmd.e:
pygame 2.5.1 (SDL 2.28.2, Python 3.6.
Hello from the pygame community. http
Procesando con A* Search (Distancia M
Iteraciones A*_S: 198
El algoritmo tomo 1.9802767 segundos
pygame window



De igual manera que BFS, este algoritmo resuelve el laberinto aparentemente de forma paralela, puesto que su prioridad se basa tanto en escoger el nodo vecino con la mejor heurística, como el camino más corto según el costo, que en este problema es el mismo para todos los nodos (1). Dicha elección se hace a partir de una suma entre el valor de la heurística y el costo acumulado del camino. Por lo tanto, es posible inducir que el algoritmo A* es mejor aprovechado en grafos en los que las aristas tienen pesos con importancia para resolver el problema.

¿Alguno es siempre mejor que otro?

Resultados test_maze

```
pygame 2.5.1 (SDL 2.28.2, Python 3.6.2)
Hello from the pygame community. https://www.pygame.org/contribute.html
Procesando con Breadth First Search...
Iteraciones BFS: 1339
El algoritmo tomo 13.390372 segundos en ejecutarse.
Procesando con Depth First Search...
Iteraciones DFS: 1285
El algoritmo tomo 12.8502899 segundos en ejecutarse.
Procesando con Depth Delimited Search...
Iteraciones DLS: 1285
El algoritmo tomo 12.851642300000002 segundos en ejecutarse.
Procesando con Greedy Best First Search (Distancia Euclidiana)...
Iteraciones GBFS: 924
El algoritmo tomo 9.242007800000003 segundos en ejecutarse.
Procesando con A* Search (Distancia Euclidiana)...
Iteraciones A*_S: 1245
El algoritmo tomo 12.461148499999993 segundos en ejecutarse.
Procesando con Greedy Best First Search (Distancia Manhattan)...
Iteraciones GBFS: 952
El algoritmo tomo 9.526014099999998 segundos en ejecutarse.
Procesando con A* Search (Distancia Manhattan)...
Iteraciones A*_S: 1228
El algoritmo tomo 12.282448599999995 segundos en ejecutarse.
Presione una tecla para continuar . . .
```

Menor tiempo y menor cantidad de iteraciones: GBFS con distancia euclidiana

Mayor tiempo y mayor cantidad de iteraciones: DFS y DDS

Resultados Prueba_1

```
pygame 2.5.1 (SDL 2.28.2, Python 3.6.2)
Hello from the pygame community. https://www.pygame.org/contribute.html
Procesando con Breadth First Search...
Iteraciones BFS: 350
El algoritmo tomo 3.5021559 segundos en ejecutarse.
Procesando con Depth First Search...
Iteraciones DFS: 241
El algoritmo tomo 2.4103593000000005 segundos en ejecutarse.
Procesando con Depth Delimited Search...
Iteraciones DLS: 241
El algoritmo tomo 2.411243299999999 segundos en ejecutarse.
Procesando con Greedy Best First Search (Distancia Euclidiana)...
Iteraciones GBFS: 227
El algoritmo tomo 2.270310500000001 segundos en ejecutarse.
Procesando con A* Search (Distancia Euclidiana)...
Iteraciones A*_S: 290
El algoritmo tomo 2.8997334000000023 segundos en ejecutarse.
Procesando con Greedy Best First Search (Distancia Manhattan)...
Iteraciones GBFS: 238
El algoritmo tomo 2.3833982999999996 segundos en ejecutarse.
Procesando con A* Search (Distancia Manhattan)...
Iteraciones A*_S: 198
El algoritmo tomo 1.9804560000000002 segundos en ejecutarse.
Presione una tecla para continuar . . .
```

Menor tiempo y menor cantidad de iteraciones: A* con distancia manhattan

Mayor tiempo y mayor cantidad de iteraciones: BFS

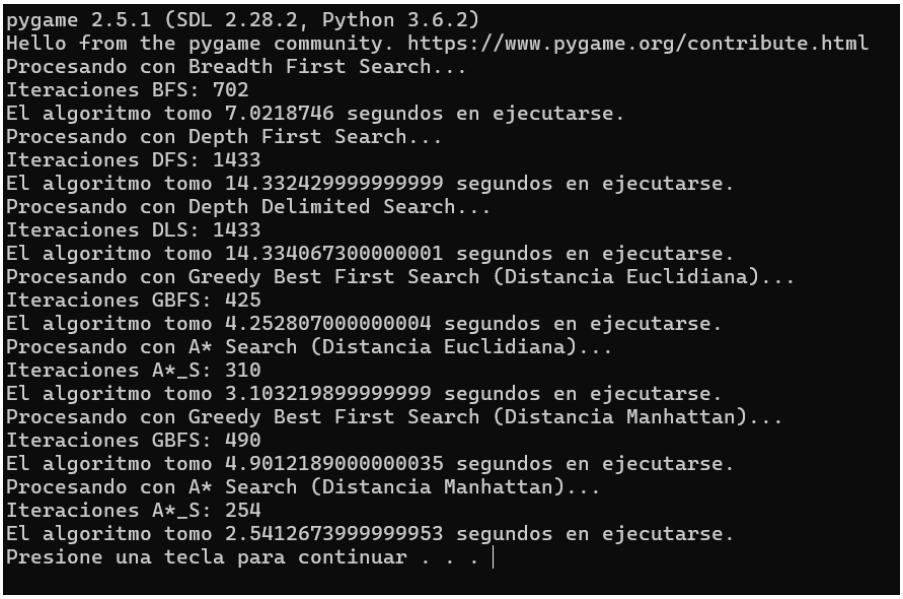
Resultados Prueba_2

```
pygame 2.5.1 (SDL 2.28.2, Python 3.6.2)
Hello from the pygame community. https://www.pygame.org/contribute.html
Procesando con Breadth First Search...
Iteraciones BFS: 794
El algoritmo tomo 7.9405286 segundos en ejecutarse.
Procesando con Depth First Search...
Iteraciones DFS: 674
El algoritmo tomo 6.742473699999998 segundos en ejecutarse.
Procesando con Depth Delimited Search...
Iteraciones DLS: 674
El algoritmo tomo 6.740883800000002 segundos en ejecutarse.
Procesando con Greedy Best First Search (Distancia Euclidiana)...
Iteraciones GBFS: 101
El algoritmo tomo 1.0095070999999969 segundos en ejecutarse.
Procesando con A* Search (Distancia Euclidiana)...
Iteraciones A*_S: 369
El algoritmo tomo 3.6930551000000023 segundos en ejecutarse.
Procesando con Greedy Best First Search (Distancia Manhattan)...
Iteraciones GBFS: 186
El algoritmo tomo 1.8613424999999992 segundos en ejecutarse.
Procesando con A* Search (Distancia Manhattan)...
Iteraciones A*_S: 249
El algoritmo tomo 2.4901827999999995 segundos en ejecutarse.
Presione una tecla para continuar . . .
```

Menor tiempo y menor cantidad de iteraciones: GBFS con distancia euclidiana

Mayor tiempo y mayor cantidad de iteraciones: BFS

Resultados Prueba_3



Menor tiempo y menor cantidad de iteraciones: A* con distancia manhattan

Mayor tiempo y mayor cantidad de iteraciones: DFS y DDS

Se puede visualizar que el algoritmo A* con distancia manhattan obtuvo mejores resultados que ese mismo algoritmo utilizando distancia euclidiana. Puede explicarse que la distancia manhattan es mejor heurística que la distancia euclidiana debido a la naturaleza cuadriculada de la matriz del laberinto y la capacidad del algoritmo A* de escoger el camino óptimo. Así mismo, los algoritmos que utilizaban costo y/o heurística como lo son el GFBS y el A*, obtuvieron en su mayoría mejores tiempos y menor cantidad de iteraciones que sus contrapartes de búsquedas desinformadas. Finalmente, el algoritmo menos eficiente para resolver el laberinto fue el BFS, a excepción de la primera y última prueba en la que la forma del laberinto perjudicó a los algoritmos DFS y DDS.