ESTADISTICA EN PHYTON



Electiva - II

Jaime Andres Martinez

2

0

2

5

Taller Nº4

Cristhian Cañrar

1. /03/2025
2. Dijkstra

El código proporcionado implementa el algoritmo de Dijkstra para calcular las distancias más cortas desde un nodo origen en un grafo ponderado. Utiliza **heapq** para gestionar prioridades, **timeit** para medir el tiempo promedio de ejecución, **networkx** para crear el grafo y **matplotlib** para visualizarlo. El algoritmo actualiza las distancias mientras explora los nodos, y al final muestra las distancias mínimas. La Figura 1 y 2 corresponde a este código, que incluye la definición del grafo, la lógica del algoritmo, la función para graficar y la medición del tiempo promedio de ejecución mediante **timeit**. Por otro lado, la Figura 3 es la representación gráfica generada por el código, que muestra visualmente el grafo con sus nodos, aristas y los pesos asociados a cada conexión. Esta gráfica ayuda a entender la estructura del grafo y cómo se relacionan las distancias calculadas por el algoritmo.

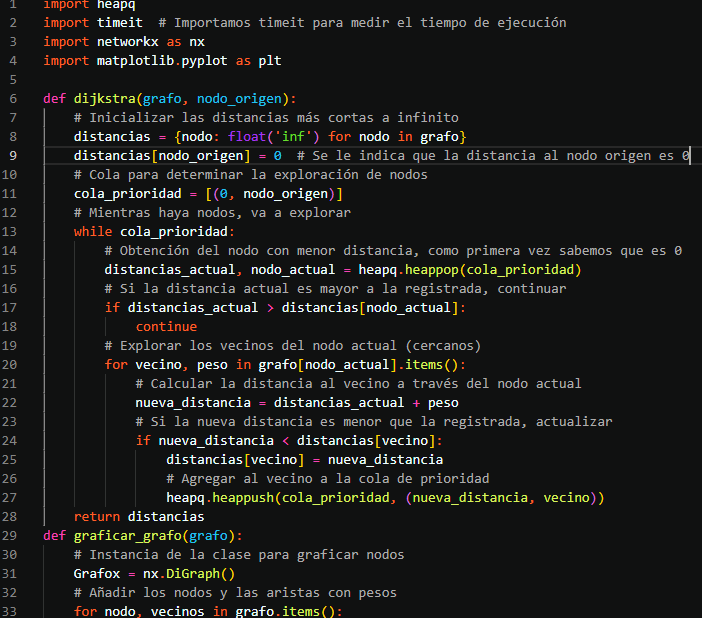


Figura 1: Codigo Pyhton “Ejercicio”

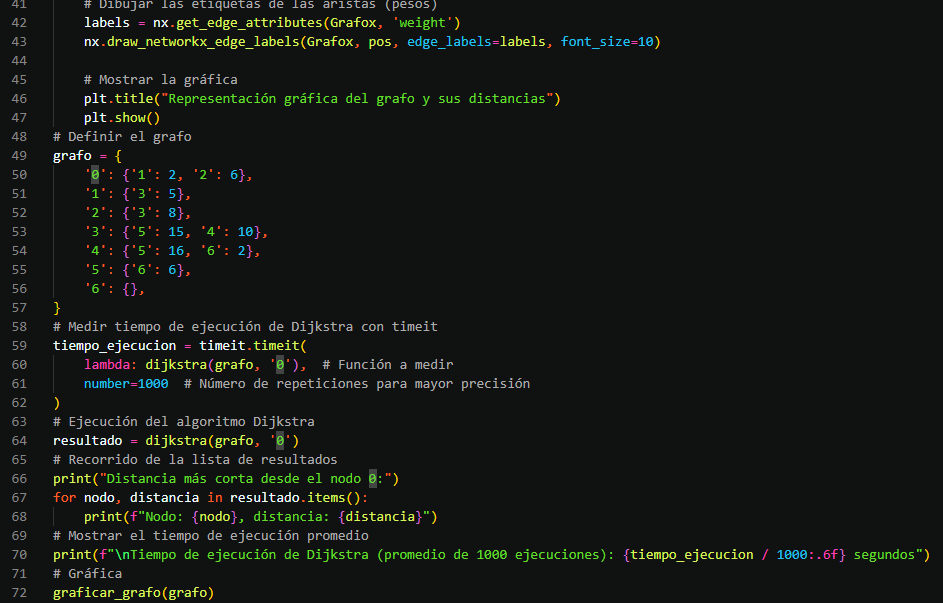


Figura 2: Código Pyhton “Ejercicio”

# Resultado del código

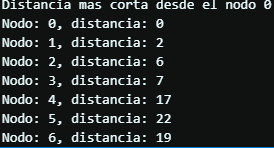
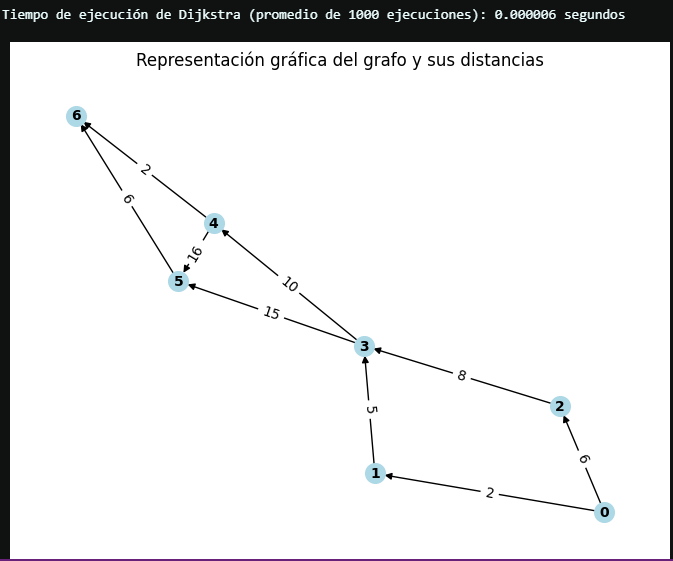


Figura 3: visualización de la grafica

1. ALGORITMO A\* (ESTRELLA)

El código proporcionado implementa el algoritmo A\* (A estrella) para encontrar la ruta más corta desde un nodo origen hasta un nodo objetivo en un grafo ponderado, utilizando una heurística que puede ajustarse según el caso (en este código se define como 0 por simplicidad). Utiliza **heapq** para gestionar la cola de prioridad y explorar los nodos según la suma del costo acumulado y la estimación heurística. Además, incluye una función para graficar el grafo con **networkx** y **matplotlib**, permitiendo visualizar las conexiones, aristas y pesos entre nodos. La Figura 4 y 5 corresponde a este código, que incluye la definición del grafo, la lógica del algoritmo y la función de grafica. Por otro lado, la Figura 6 es la representación gráfica generada por el código, mostrando visualmente el grafo con sus nodos, aristas y pesos asociados. Esta gráfica ayuda a comprender la estructura del grafo y cómo se relacionan las distancias calculadas por el algoritmo. Finalmente, el tiempo de ejecución del algoritmo A\* se mide utilizando **timeit**, repitiendo la ejecución 1000 veces para obtener un promedio preciso.

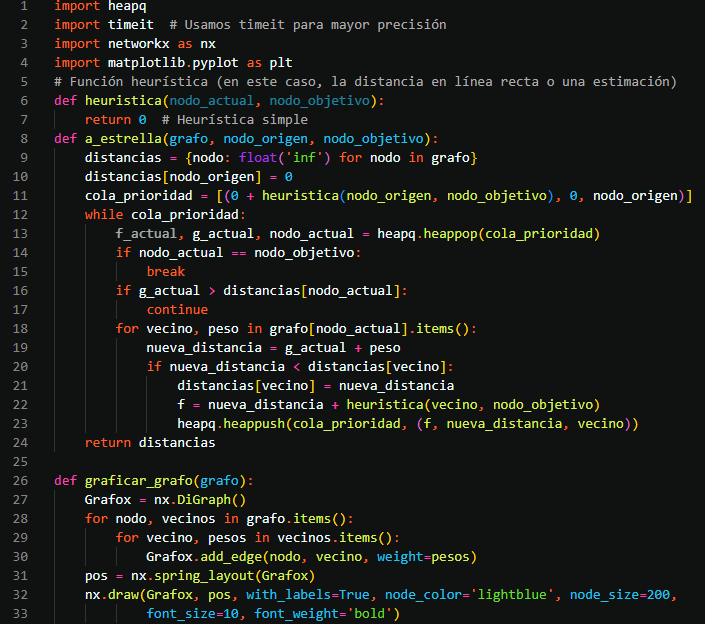


Figura 4: Codigo Pyhton “Algoritmo A\*”

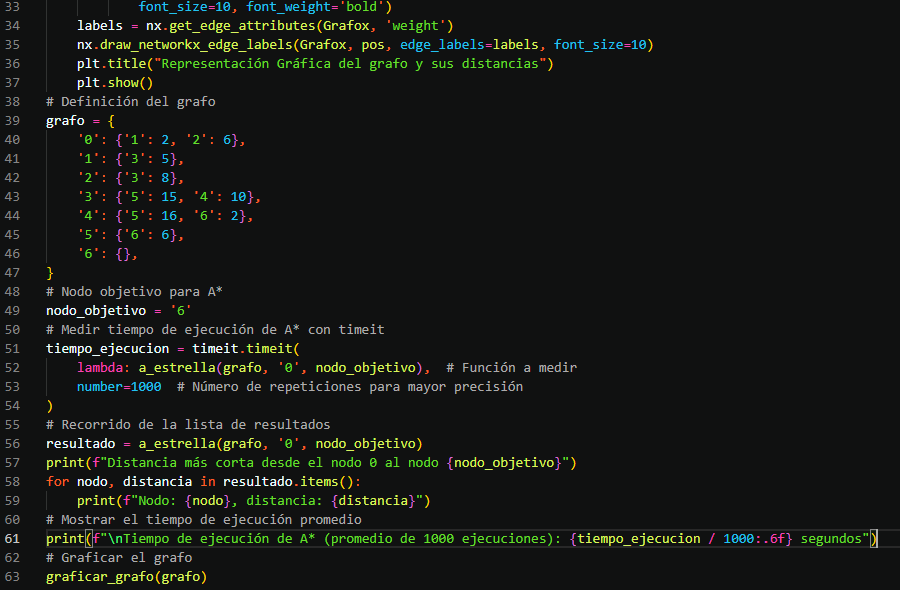
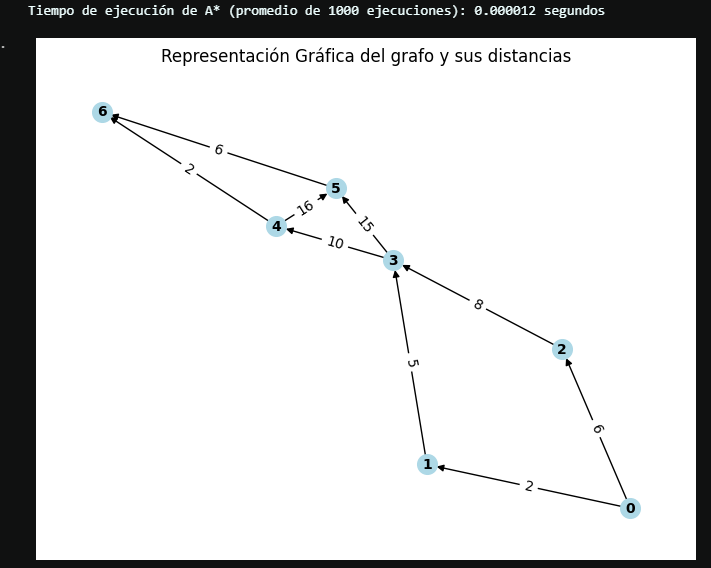


Figura 5: Código Pyhton “Ejercicio”

# Resultado del código

 Texto

El contenido generado por IA puede ser incorrecto.

Figura 6: visualización de la grafica

1. **TABLA COMPARATIVA**

**Tabla

El contenido generado por IA puede ser incorrecto.**

Figura 7: visualización de la tabla de excel – comparación

1. **Minimax con poda alfa-beta en un juego sencillo**

**(Piedra-Papel-Tijera).**

El código de Piedra-Papel-Tijera que implementa **Minimax con poda alfa-beta** utiliza este algoritmo para que la computadora elija siempre la mejor opción basada en la elección del jugador. En este caso, la computadora actúa como el "minimizador", buscando contrarrestar la jugada del jugador para minimizar su posibilidad de ganar. Sin embargo, debido a la simplicidad del juego, el algoritmo siempre lleva a la computadora a ganar o empatar, ya que puede predecir y responder de manera óptima a la elección del jugador. Esto hace que el juego sea predecible y poco divertido, por lo que, en la práctica, no se recomienda usar **Minimax para Piedra-Papel-Tijera.** Una alternativa más justa es que la computadora elija una opción al azar, lo que equilibra las posibilidades y hace el

juego más entretenido.

**Juego:**

N: finaliza juego S: inicia nueva partida

**Comandos:**

Piedra – Papel - Tijera

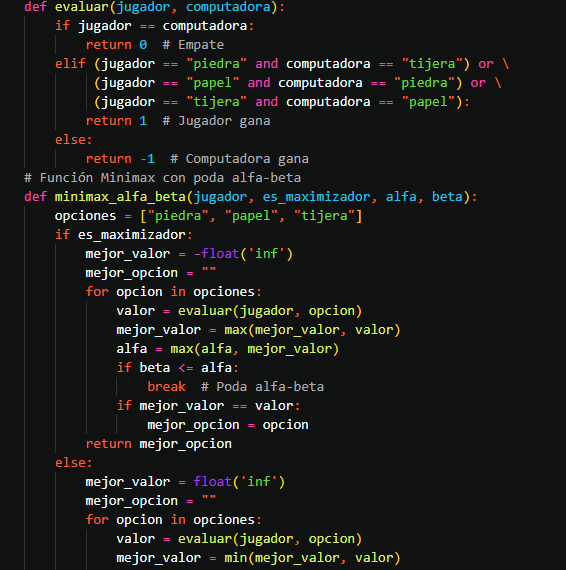
****

Figura 8: Código Pyhton “Juego”

Texto

El contenido generado por IA puede ser incorrecto.

Figura 9: Código Pyhton “Juego”

* + 1. **Resultado del código**

****

Figura 10: Comandos para “Jugar”

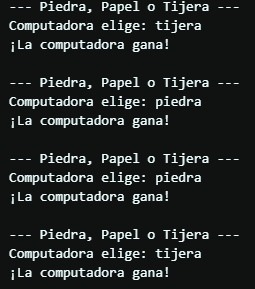
****

Figura 11: Resultado “Juego”

1. **CONCLUSIONES**

* El algoritmo de Dijkstra es un método robusto para encontrar el camino más corto en grafos con pesos no negativos. Su enfoque de exploración uniforme garantiza que siempre encuentre la solución óptima, ya que evalúa todos los nodos sin priorizar direcciones específicas. Sin embargo, esta exhaustividad lo hace menos eficiente en grafos grandes, ya que puede explorar nodos innecesarios. Cuando se mide con timeit, su tiempo de ejecución tiende a ser mayor, especialmente en comparación con algoritmos heurísticos como A\*
* El algoritmo A\* mejora la eficiencia de Dijkstra al incorporar una función heurística que guía la búsqueda hacia el nodo objetivo. Esta heurística permite priorizar nodos prometedores, reduciendo el espacio de búsqueda y el tiempo de ejecución. Sin embargo, su rendimiento depende de la calidad de la heurística: si no es admisible (sobreestima la distancia), A\* puede no garantizar la optimalidad. Al medir con timeit, A\* suele mostrar un tiempo de ejecución menor que Dijkstra, especialmente en grafos grandes o cuando la heurística está bien diseñada.
* El algoritmo Minimax con poda alfa-beta se implementó en el código para que la computadora tome decisiones óptimas basadas en la elección del jugador. Sin embargo, en un juego simple como Piedra-Papel-Tijera, donde no hay un estado del juego que evolucione con el tiempo, este algoritmo resulta excesivo y poco práctico. La computadora siempre elige la mejor opción para contrarrestar al jugador, lo que lleva a que siempre gane o empate, eliminando el factor de aleatoriedad y diversión que caracteriza a este juego.
* Piedra-Papel-Tijera es un juego basado en la suerte y la elección aleatoria, sin un componente estratégico profundo. Aunque el código utiliza técnicas avanzadas como Minimax con poda alfa-beta, estas no son necesarias para este tipo de juego. Una implementación más adecuada y justa sería que la computadora elija una opción al azar, lo que mantendría la esencia del juego y lo haría más equilibrado y entretenido para el jugador.

1. **BIBLIOGRAFIA**

* <https://www.geeksforgeeks.org/a-search-algorithm/>
* <https://docs.python.org/3/library/timeit.html>
* <https://aima.cs.berkeley.edu/>
* <https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/>
* <https://realpython.com/python-rock-paper-scissors/>