PRÁCTICA NO. 1

Resolución de rutas óptimas con el algoritmo A\* (Unidad 1)

|  |  |
| --- | --- |
| Competencia: |  |

Desarrolla soluciones algorítmicas eficientes mediante la implementación del algoritmo A\*, aplicando funciones heurísticas para encontrar rutas óptimas en un espacio de búsqueda determinado.

|  |  |
| --- | --- |
| Material: |  |

Equipo de cómputo.

El algoritmo A\* es un pro total para encontrar rutas en grafos. Básicamente, te saca del apuro cuando necesitas el camino más corto entre un punto de inicio y uno final, sobre todo si hay pesos (o costos) en las conexiones.

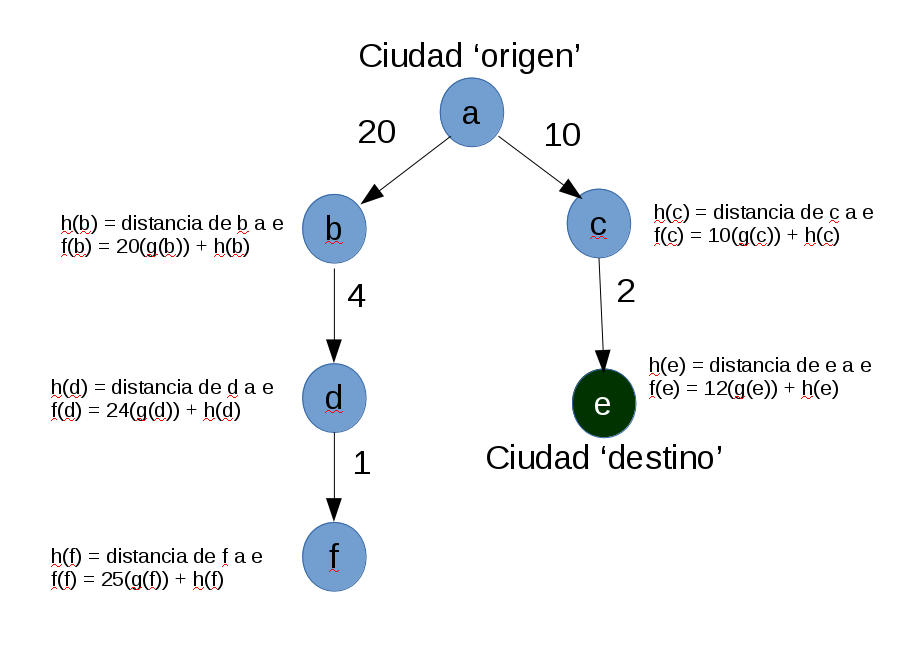


Ilustración 1: Algoritmo A en acción Origen

Para que no te pierdas, hay que entender sus piezas clave:

Nodos: Como las esquinas de un mapa (o los puntos clave).

Aristas (Bordes): Las conexiones entre nodos (como calles que unen dos esquinas).

Costo de ruta: Lo que te cuesta moverte de un nodo a otro (ej: tiempo, distancia).

Heurística: Un "suposición educada" de cuánto falta para llegar al objetivo.

Espacio de búsqueda: Todos los caminos posibles que podrías explorar [1].

El A\* usa una fórmula clave: f(n) = g(n) + h(n), donde:

g(n): El costo real desde el inicio hasta el nodo actual (n).

h(n): La heurística, es decir, un "achunte" del costo que falta hasta el destino.

Para que funcione bien, la heurística no debe pasarse de lista (o sea, ser admisible: nunca sobrestimar el costo real). Ejemplos clásicos son la distancia en línea recta (euclidiana) o la suma de movimientos en cuadrícula (Manhattan).

Este algoritmo es el rey en videojuegos, GPS y robots, porque les ayuda a tomar decisiones inteligentes sin perder tiempo en rutas ineficientes [2].

|  |  |
| --- | --- |
| Desarrollo de la práctica: |  |

La práctica consiste en implementar en Python (o lenguaje similar) el algoritmo A\* para resolver un problema de búsqueda de camino en una cuadrícula o grafo.

Requisitos del diseño:

* Crear un mapa de nodos (puede ser una matriz con obstáculos).
* Aplicar el algoritmo A\* para encontrar el camino más corto entre un nodo origen y uno destino.
* Usar una heurística (distancia Manhattan o Euclidiana).
* Mostrar visualmente o imprimir paso a paso la solución.

La práctica consistió en la implementación del algoritmo A\* para encontrar la ruta más corta entre dos ciudades, utilizando una interfaz gráfica desarrollada con Tkinter y visualización del árbol de búsqueda mediante NetworkX y Matplotlib. A continuación, se describen las fases del desarrollo:

Se diseñó una interfaz gráfica en Python utilizando el módulo Tkinter, la cual permite al usuario seleccionar la ciudad de origen desde un Combobox. El destino es predeterminado como "Bucarest", ya que su heurística es igual a cero (h(n) = 0), lo que representa el nodo objetivo (El destino puede cambiar).



Ilustración 2 Interfaz gráfica de selección

Se desarrolló una función llamada plot\_vertical\_tree que realiza la búsqueda del camino óptimo entre el nodo de inicio y el de destino. Esta función utiliza una cola de prioridad patch y plt para almacenar los nodos pendientes de visitar, priorizándolos con base en su evaluación f(n) = g(n) + h(n), donde g(n) es el costo acumulado y h(n) la heurística estimada al destino.

Tabla

El contenido generado por IA puede ser incorrecto.

Ilustración 3 Implementación del algoritmo A\*

Cuando el algoritmo encuentra la mejor ruta, construye un árbol de búsqueda que refleja todos los nodos que revisó en el proceso. Cada nodo incluye sus valores de g (costo real), h (estimación heurística) y f (suma total), mientras que las líneas entre ellos muestran el camino que se analizó. Para visualizar esto, se usó la librería NetworkX para generar la estructura y Matplotlib para graficarla, integrando el resultado directamente en la interfaz del programa.

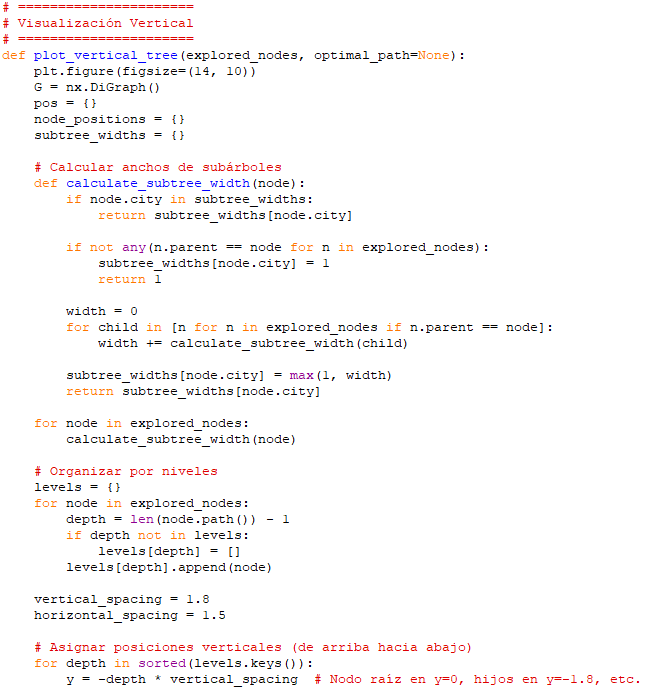


Ilustración 4 Árbol de búsqueda generado por A\*

La interfaz gráfica despliega la ruta óptima calculada por el algoritmo A\*, junto con el costo total en kilómetros que representa seguir ese trayecto. Esta información se actualiza automáticamente y aparece en pantalla cuando el usuario hace clic en el botón "Buscar Ruta", mostrándose en un campo de texto dedicado.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Ilustración 5 Ruta óptima encontrada y su costo total

El grafo se implementó como un diccionario de Python, donde cada ciudad (nodo) almacena sus conexiones junto con los costos asociados. Esta representación permitió modelar eficientemente el mapa de Rumania, un caso de estudio frecuente en algoritmos de búsqueda heurística debido a su estructura bien definida y ampliamente documentada.

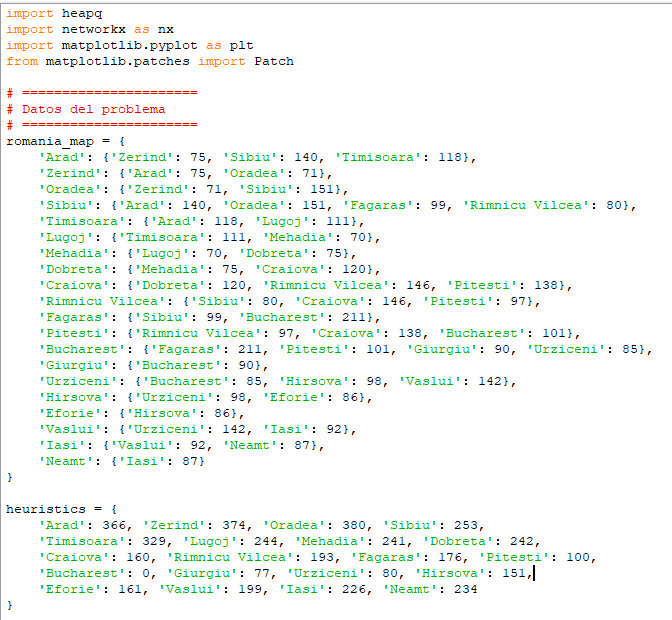


Ilustración 6 Estructura del grafo y conexiones

Para facilitar la visualización, el camino óptimo se resalta con colores distintivos: Nodos de la ruta final: Verde, Aristas del trayecto: Azul y Rutas alternativas sin desglose: Gris.

Este esquema de colores permite identificar de un vistazo la ruta más eficiente calculada por el algoritmo, mejorando la experiencia de usuario y la interpretación de los resultados.

Diagrama

El contenido generado por IA puede ser incorrecto.

Ilustración 7 Visualización del recorrido en el árbol

Tras seleccionar una ciudad de inicio, como Arad, el sistema mostró la ruta más corta hacia Bucarest, junto con el costo total en kilómetros. La ruta generada fue, por ejemplo: *Fagaras → Sibiu → Rimnicu Vilcea → Craiova → Dobreta → Mehadia*, con un costo total de 761 km.

|  |  |
| --- | --- |
| Cuestionario: |  |

NO APLICA

|  |  |
| --- | --- |
| Conclusiones: |  |

Esta práctica fue clave para entender a fondo el algoritmo A\* y su aplicación en búsquedas inteligentes. Al implementarlo, confirmamos cómo combina el costo real recorrido (g(n)) con una estimación heurística (h(n)) para encontrar siempre el camino más corto. Usamos colas de prioridad y una gestión eficiente de nodos abiertos vs. cerrados para optimizar su rendimiento.

La interfaz gráfica hecha en Python no solo hizo más intuitivo el proceso, sino que nos permitió visualizar en tiempo real la ruta óptima (resaltada en azul/verde/gris) y el árbol de búsqueda completo. Esto fue clave para entender el paso a paso del algoritmo, desde la teoría hasta su ejecución práctica.

Experimentamos cómo la heurística define la eficiencia de A\*:

Con una buena heurística (como la distancia en línea recta), el algoritmo encuentra rutas óptimas rápido y con pocos cálculos. Si la heurística está mal ajustada, el algoritmo pierde tiempo en caminos innecesarios o incluso devuelve rutas incorrectas.

Esta práctica no solo reforzó conocimientos en IA y planificación de rutas, sino que hizo valorar el impacto real de estos algoritmos en sistemas de navegación, robótica o videojuegos. Una experiencia que mezcló programación, teoría y análisis visual para resolver problemas complejos de manera inteligente.

|  |  |
| --- | --- |
| Aplicaciones industriales: |  |

NO APLICA

|  |  |
| --- | --- |
| Citas y referencias: |  |

[1] «El Algoritmo A\*: Guía completa», *Datacamp*. https://www.datacamp.com/es/tutorial/a-star-algorithm

[2] E. GraphEverywhere, «Algoritmo A \*», *GraphEverywhere*, 29 de octubre de 2019. https://www.grapheverywhere.com/algoritmo-a/

|  |  |
| --- | --- |
| Criterios de evaluación: |  |

Funcionamiento: la práctica debe funcionar al 100%, en caso contrario la calificación se otorgará de acuerdo con lo que logre hacer sin errores.

Documentación: deberá de entregarse una documentación donde se demuestre el paso a paso del desarrollo que se realizó hasta llegar al resultado final de implementación. Aquí se considera estructura, errores de redacción, ortográficos y referencias.

Se evalúa mediante rúbrica proporcionada previamente de acuerdo con los criterios en ella establecida.