ESPE	INFORME DE PRÁCTICA DE LABORATORIO

DEPARTAMENTO:	**Ciencias de la Computación**	**CARRERA:**	**Ingeniería en tecnologías de la información**		
ASIGNATURA:	Inteligencia Artificial	**NIVEL:**	6to	**FECHA:**	27/10/2025
DOCENTE:	Ing. Kevin Chuquitarco	**PRÁCTICA N°:**	Lab2 U1	**CALIFICACIÓN:**	

Implementación de técnicas de búsquedas Informadas

Juan David Jiménez Romero

RESUMEN

En este informe se realizó el análisis y la implementación de algoritmos de búsqueda informada utilizando Python, enfocándose en su aplicación en estructuras de grafos con pesos y heurísticas. Se desarrollaron la búsqueda voraz primero el mejor, la búsqueda A*, y la búsqueda de memoria acotada, explorando cómo cada uno utiliza heurísticas y costos para encontrar rutas óptimas. La práctica permitió identificar las ventajas y limitaciones de cada técnica según el contexto, destacando su relevancia como base para sistemas de inteligencia artificial en tareas como planificación y navegación. En general, se fortaleció la comprensión del razonamiento computacional y la eficiencia en la exploración de datos.

Palabras Claves: Búsquedas Informadas, Heurística, IA

1. INTRODUCCIÓN:

En la actualidad podemos encontrar algunas maneras de las cuales las Inteligencias Artificiales pueden realizar búsquedas de información una de ellas es la búsqueda no informada la cual consiste en hallar datos por medio de un dato inicial y el dato que se quiera obtener pero sin nada de pistas de por medio esto es lo que le da el nombre y por otro lado tenemos la búsqueda informada el cual nos lleva al punto de este laboratorio el cual nos introduce las búsquedas informadas, como la búsqueda voraz, A*, memoria acotada y voraz primero el mejor, analizando su funcionamiento y aplicación en grafos con pesos, con el objetivo de entender cómo estos algoritmos sientan las bases para capacidades avanzadas en inteligencia artificial.

2. OBJETIVO(S):

2.1 Analizar el comportamiento de los algoritmos de búsqueda informada aplicados a grafos con heurística.
2.2 Comparar los resultados obtenidos entre las distintas estrategias de búsqueda implementadas (Voraz, A*, SMA*).
2.3 Evaluar la eficiencia de cada algoritmo en función del recorrido, costo y nodos expandidos durante la búsqueda

3. MARCO TEÓRICO:

¿Qué es Python?:

En términos técnicos, Python es un lenguaje de programación de alto nivel, orientado a objetos, con una semántica dinámica integrada, principalmente para el desarrollo web y de aplicaciones informáticas.
Es muy atractivo en el campo del Desarrollo Rápido de Aplicaciones (RAD) porque ofrece tipificación dinámica y opciones de encuadernación dinámicas. (aula21, 2025)

¿Qué es la búsqueda voraz el primero mejor?

Es un algoritmo de búsqueda informado donde la función de evaluación es estrictamente igual a la función heurística, ignorando los pesos de las aristas en un grafo ponderado, ya que solo se considera el valor heurístico. Para buscar un nodo objetivo, se expande el nodo más cercano al objetivo, según lo determinado por la función heurística. Este enfoque asume que es probable que se encuentre una solución rápidamente. Sin embargo, la solución de una búsqueda voraz "mejor primero" puede no ser óptima, ya que podría existir una ruta más corta. (Yang, 2023)

¿Qué es la búsqueda A?*

Es un algoritmo potente y ampliamente utilizado para recorrer grafos y encontrar caminos. Encuentra el camino más corto entre un nodo inicial y un nodo objetivo en un grafo ponderado. Imagina que intentas encontrar en un mapa la ruta más corta entre dos ciudades. Mientras que el algoritmo de Dijkstra exploraría en todas direcciones y la Búsqueda

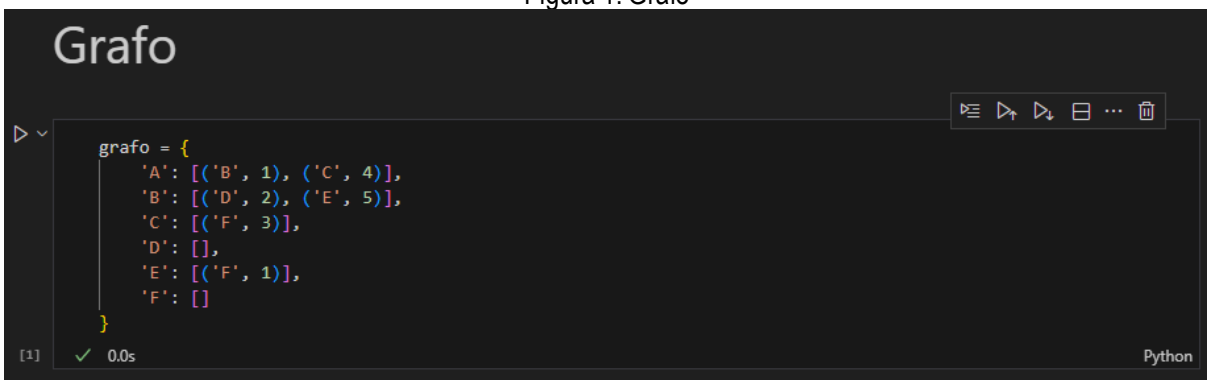
Mejor Primero podría dirigirse directamente hacia el destino (omitiendo potencialmente atajos), A* hace algo más inteligente. (Kumar, 2024)

¿Qué es la búsqueda SMA*?

La búsqueda por SMA* o A* de Memoria Simplificada Limitada es un algoritmo de ruta más corta basado en el algoritmo A*. La principal ventaja de SMA* es que utiliza una memoria limitada, mientras que el algoritmo A* podría requerir memoria exponencial. (UNAD, 2013)

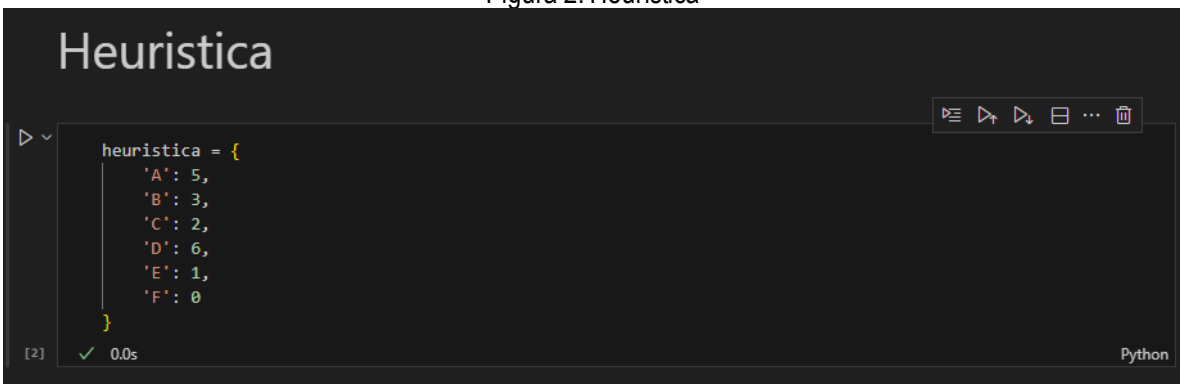
4. DESCRIPCIÓN DEL PROCEDIMIENTO:

Figura 1. Grafo



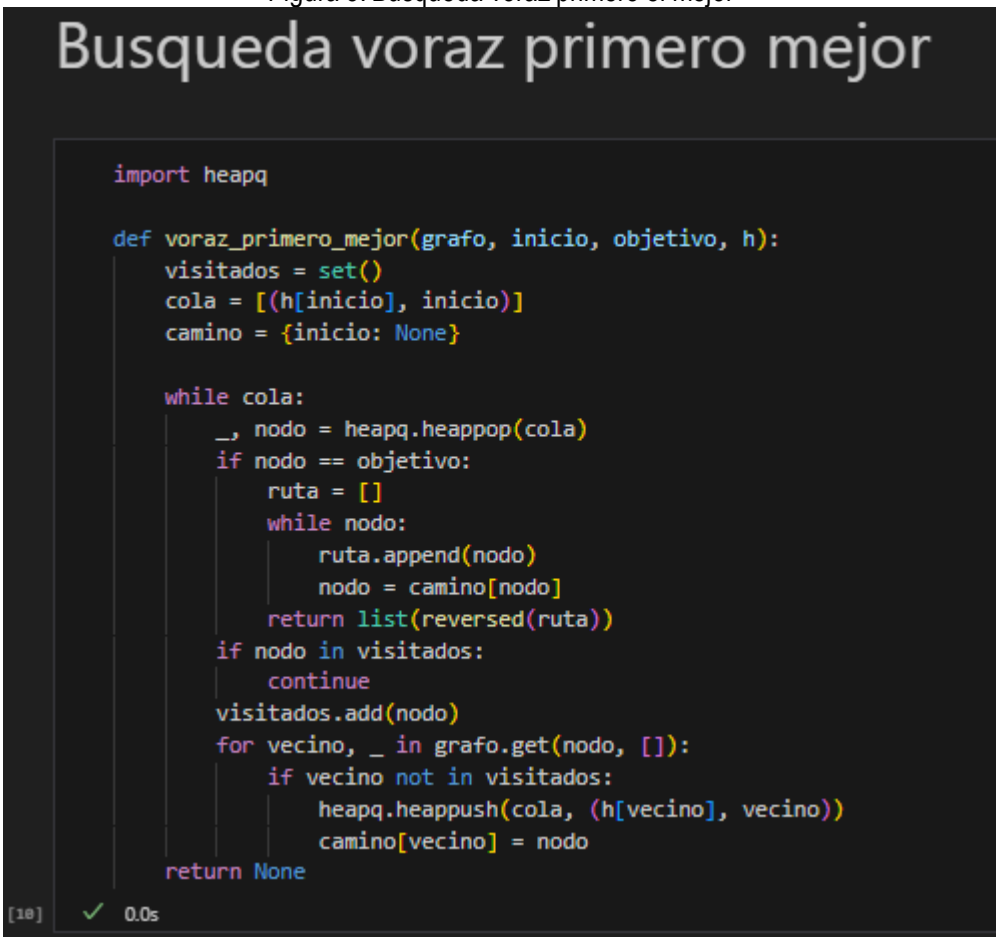
Código para la creación de un grafo con pesos que tiene 6 nodos

Figura 2. Heurística



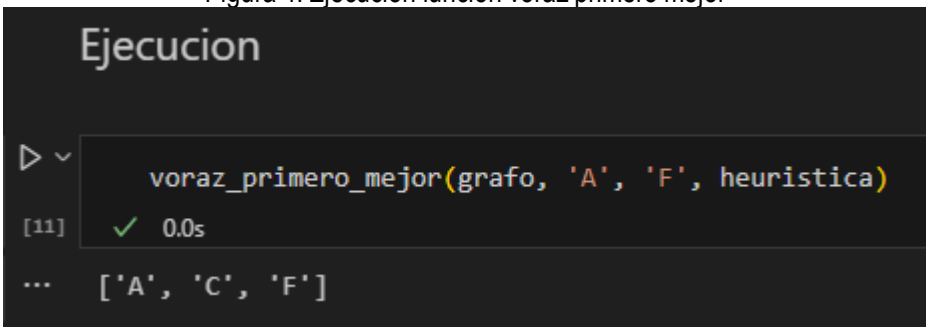
Se crea una heurística con los pesos estimados para las fórmulas de los algoritmos de búsqueda informadas.

Figura 3. Búsqueda voraz primero el mejor



La función de la búsqueda voraz primero el mejor realizada se usa para hallar una ruta óptima desde un nodo inicial hasta un objetivo en un grafo, utilizando una heurística que prioriza nodos prometedores en cada paso. Comienza con un nodo inicial, explora seleccionando el nodo más cercano al objetivo según la heurística, y al encontrarlo, reconstruye la ruta completa devolviéndola como lista; si no lo encuentra, retorna nulo.

Figura 4. Ejecución función voraz primero mejor



Al momento de ejecutar este nos mostrara el camino realizado por la búsqueda a consideración del camino con mejor peso hacia el punto indicado.

Figura 5. Busqueda A*

Busqueda A*

```python
import heapq

def busqueda_a_estrella(grafo, inicio, objetivo, h):
    visitados = set()
    cola = [(h[inicio], 0, inicio)]  # (f, g, nodo)
    camino = {inicio: None}

    while cola:
        f, g, nodo = heapq.heappop(cola)
        if nodo == objetivo:
            ruta = []
            while nodo:
                ruta.append(nodo)
                nodo = camino[nodo]
            return list(reversed(ruta)), g
        if nodo in visitados:
            continue
        visitados.add(nodo)
        for vecino, costo in grafo.get(nodo, []):
            if vecino not in visitados:
                g_nuevo = g + costo
                f_nuevo = g_nuevo + h[vecino]
                heapq.heappush(cola, (f_nuevo, g_nuevo, vecino))
                camino[vecino] = nodo
    return None, float('inf')
```

✓ 0.0s

En el caso del algoritmo de búsqueda A* este trabaja de manera que busca encontrar la ruta más corta desde un nodo inicial hasta un objetivo en un grafo, utilizando una heurística junto con el costo acumulado para guiar la exploración. Este comienza desde el nodo inicial, evalúa los nodos disponibles priorizando aquellos con menor costo estimado y al alcanzar el objetivo, al finalizar este devuelve la ruta óptima como lista y si no lo encuentra, retorna nulo.

Figura 6. Ejecución A*

Ejecucion

```python
busqueda_a_estrella(grafo,'A','F',heuristica)
```

✓ 0.0s

```
(['A', 'B', 'E', 'F'], 7)
```

Al momento de ejecutar la función de la búsqueda A* este nos pedirá los atributos de grafo, inicio, final y la heurística o estimado donde al ejecutarlo este nos devolverá el camino más corto y además nos mostrará el peso total que realizo en todo el trayecto.

Figura 7. Búsqueda de memoria acotado

Busqueda con memoria acotada

```python
import heapq

def sma_estrella(grafo, inicio, objetivo, h, memoria_max=3):
    visitados = set()
    cola = [(h[inicio], 0, inicio)]
    camino = {inicio: None}

    while cola:
        f, g, nodo = heapq.heappop(cola)
        if nodo == objetivo:
            ruta = []
            while nodo:
                ruta.append(nodo)
                nodo = camino[nodo]
            return list(reversed(ruta)), g
        if nodo in visitados:
            continue
        visitados.add(nodo)
        hijos = []
        for vecino, costo in grafo.get(nodo, []):
            g_nuevo = g + costo
            f_nuevo = g_nuevo + h[vecino]
            hijos.append((f_nuevo, g_nuevo, vecino))
            camino[vecino] = nodo
        # Mantener solo los mejores N nodos según memoria_max
        for hijo in sorted(hijos)[:memoria_max]:
            heapq.heappush(cola, hijo)
    return None, float('inf')
```

✓ 0.0s

La función realiza el algoritmo de búsqueda con memoria acotada para hallar la ruta más corta, pero este va limitando el uso de memoria al especificar un tamaño máximo. Comienza desde el nodo inicial, explora los nodos usando una cola de prioridad basada en una heurística, y al encontrar el objetivo, devuelve obtenida y en caso de no encontrarlo retorna nulo. Su función principal es controlar la memoria eliminando nodos menos prometedores cuando se excede el límite.

Figura 8. Ejecución memoria acotada

Ejecucion

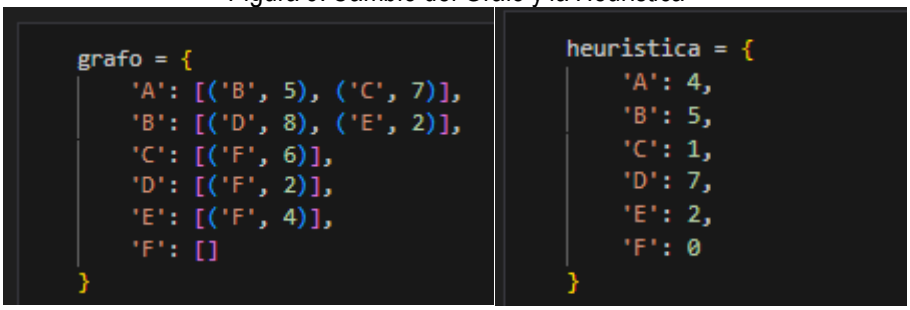
```python
sma_estrella(grafo, 'A', 'F', heuristica, memoria_max=3)
```

✓ 0.0s

```
(['A', 'B', 'E', 'F'], 7)
```

Al momento de ejecutar la función de la búsqueda memoria acotada este nos pedirá los atributos de grafo, inicio, final, la heurística o estimado y la memoria máxima, donde al ejecutarlo este nos devolverá el camino más corto y además nos mostrará el peso total que realizo en todo el trayecto.

Figura 9. Cambio del Grafo y la Heurística

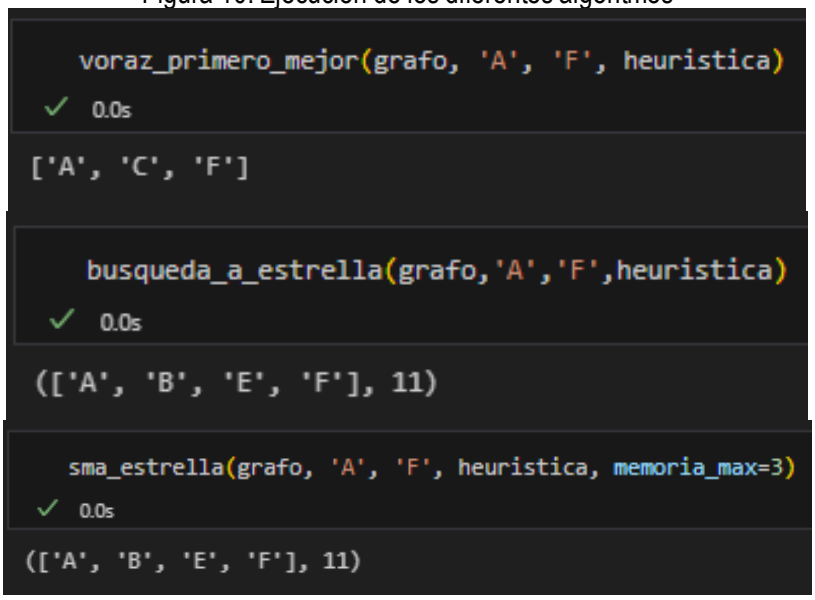


```
grafo = {
    'A': [('B', 5), ('C', 7)],
    'B': [('D', 8), ('E', 2)],
    'C': [('F', 6)],
    'D': [('F', 2)],
    'E': [('F', 4)],
    'F': []
}
```

```
heuristica = {
    'A': 4,
    'B': 5,
    'C': 1,
    'D': 7,
    'E': 2,
    'F': 0
}
```

Se realiza el cambio del grafo y de la heurística para visualizar si se obtiene de esta manera diferentes resultados al ejecutar los diferentes algoritmos de búsqueda.

Figura 10. Ejecución de los diferentes algoritmos



```
voraz_primero_mejor(grafo, 'A', 'F', heuristica)
✓ 0.0s
['A', 'C', 'F']
```

```
busqueda_a_estrella(grafo,'A','F',heuristica)
✓ 0.0s
(['A', 'B', 'E', 'F'], 11)
```

```
sma_estrella(grafo, 'A', 'F', heuristica, memoria_max=3)
✓ 0.0s
(['A', 'B', 'E', 'F'], 11)
```

Aunque se hayan realizado cambios en los costos y se le agregara otra conexión al nodo F que tenga un costo menor al punto el camino resulto terminando con el mismo recorrido, pero se ve que cambia el costo final obtenido.

5. ANÁLISIS DE RESULTADOS:

Búsqueda voraz primero el mejor:

Este tipo de algoritmo realiza un recorrido por el camino más corto que este encuentre primero esto destacando que no siempre será el más optimo y depende mucho de un buen calculo heurístico, aun así esto nos permite obtener una respuesta rápida y sin un gran uso de memoria al hacerse uso. En la (Figura 4), se obtuvo una ruta óptima desde el nodo inicial al objetivo, destacando su eficiencia en problemas con heurísticas bien definidas.

Búsqueda A*:

Este tipo de búsqueda consiste en que la sumatoria del costo real y el coste estimado por la heurística, esta búsqueda encontró la ruta más corta por medio de la comparación entre los datos y sus estimados, el elegir el camino con menor coste, pero debido a que analiza todos los nodos este realiza un gran gasto en la memoria. Al ejecutarla (Figura 6), mostró el camino óptimo y el peso total, evidenciando su precisión en grafos complejos.

Búsqueda de memoria acotada:

Limitando el uso de memoria con un tamaño máximo, esta función exploró nodos más prometedores donde dependiendo de su memoria máxima este va a realizar la búsqueda dependiendo de los costes siendo que también realiza la búsqueda de la ruta más corta, pero con menor uso de memoria máxima usada. Su ejecución (Figura 8) devolvió la ruta más corta y el peso total, demostrando control eficiente de recursos en árboles profundos.

6. GRÁFICOS O FOTOGRAFÍAS:

NO APLICA

7. DISCUSIÓN:

Los resultados resaltan las particularidades de cada método de búsqueda informada. La búsqueda voraz primero el mejor es efectiva en escenarios donde la heurística es confiable, aunque puede no garantizar la ruta óptima si la estimación falla. La búsqueda A* destacó por su capacidad de encontrar la ruta más corta al combinar costos reales y estimados, siendo ideal para aplicaciones como la navegación. En cambio, la búsqueda de memoria acotada ofreció una solución eficiente al limitar el consumo de memoria, evitando expansiones excesivas en grafos profundos. Estos experimentos subrayan la importancia de seleccionar el algoritmo según las características del problema, como la estructura del grafo y los recursos disponibles.

8. CONCLUSIONES:

1. La Se analizó el comportamiento de los algoritmos de búsqueda informada (voraz, A*, SMA*) aplicados a grafos con heurísticas, observando cómo la priorización basada en estimaciones guía la exploración y afecta la eficiencia en la búsqueda de rutas óptimas.
2. Se compararon los resultados entre las estrategias implementadas, destacando que A* ofrece la ruta más corta al combinar costos reales y heurísticos, mientras que la búsqueda voraz prioriza velocidad y SMA* controla el uso de memoria, siendo cada uno adecuado según el contexto del problema.
3. Se evaluó la eficiencia de cada algoritmo en función del recorrido, costo y nodos expandidos, concluyendo que A* minimiza el costo total, la búsqueda voraz reduce el número de nodos expandidos, y SMA* optimiza el uso de recursos en grafos profundos, reafirmando la importancia de seleccionar el método según las necesidades específicas.

9. BIBLIOGRAFÍA:

aula21. (2025). Python: qué es, para qué sirve y cómo se programa. Cursosaula21.com. https://www.cursosaula21.com/que-es-python/

Kumar, R. (2024, noviembre 7). *El Algoritmo A*: Guía completa*. Datacamp.com. https://www.datacamp.com/es/tutorial/a-star-algorithm

UNAD. (2013). *Todo Sobre Inteligencia Artificial*. Blogspot.com. https://inteligenciaartificialgrupo33.blogspot.com/p/metodos-de-busqueda-y-ejemplos.html

Yang, C. (2023, abril 22). *Búsqueda codiciosa de IA "mejor primero"*. Codecademy. https://www.codecademy.com/resources/docs/ai/search-algorithms/greedy-best-first-search