UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, INFORMÁTICA Y MECÁNICA

ESCUELA PROFESIONAL DE INGENIERÍA DE INFORMÁTICA Y DE SISTEMAS

Comparamos la eficiencia de Disjtra vs diferente algoritmos

Docente: HUILLCA HUALLPARIMACHI, Raul

INTEGRANTES:

- Barazorda Cuellar Herctor Paolo
- Ccori Taco Esmaydes
- Diaz Misme Pamela

Objetivo de nuestro proyecto

Objetivo General:

Evaluar y comparar el rendimiento pr´actico de los algoritmos de b´usqueda de caminos m´as cortos (Dijkstra, Bellman-Ford, BFS y A*) en grafos de topolog´ıa diversa y tama˜no moderado, para determinar cu´al ofrece el mejor equilibrio entre eficiencia computacional y facilidad de implementaci´on en escenarios reales.

Objetivos Específicos:

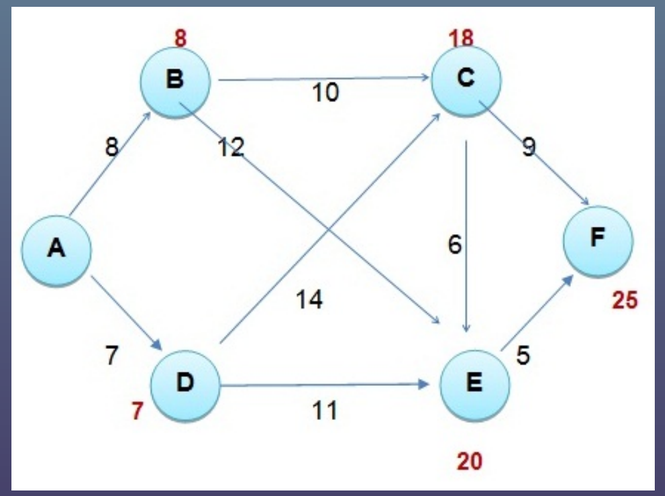
1. **I**mplementar los algoritmos seleccionados en un entorno controlado utilizando Python y la librer´ıa NetworkX, asegurando la reproducibilidad de los resultados.
2. Medir experimentalmente el tiempo de ejecuci´on y el consumo de memoria de cada algoritmo al procesar grafos sociales, mallas geogr´aficas y grafos aleatorios sint´eticos.
3. Analizar c´omo influyen las caracter´ısticas del grafo (densidad de aristas, presencia de coordenadas espaciales y distribuci´on de pesos) en la ventaja relativa del algoritmo A* frente a Dijkstra.
4. Contrastar la eficiencia emp´ırica de los algoritmos cl´asicos frente a las expectativas te´oricas, verificando si las mejoras asint´oticas modernas son necesarias para aplicaciones de escala moderada.

tabla de contenido VS Di

01 Bellman–Ford y Dijkstra

El problema de los caminos más cortos desde un único origen (SSSP) busca la distancia mínima desde un vértice fuente s a todos los demás vértices del grafo[1].

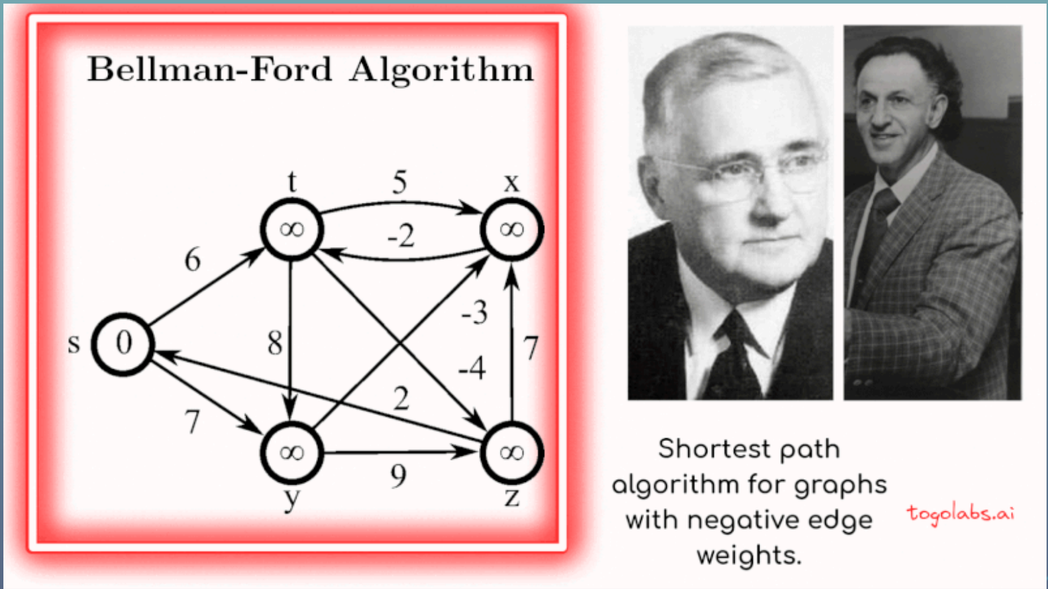
Comportamiento Visual: Dijkstra expande una "frontera" de búsqueda hacia el objetivo, mientras que Bellman–Ford realiza un escaneo global repetitivo de todo el grafo.



I.2.-Bellman-Ford

Se basa en la relajación sistemática. Revisa todas las aristas del grafo |V|−1 veces[10]

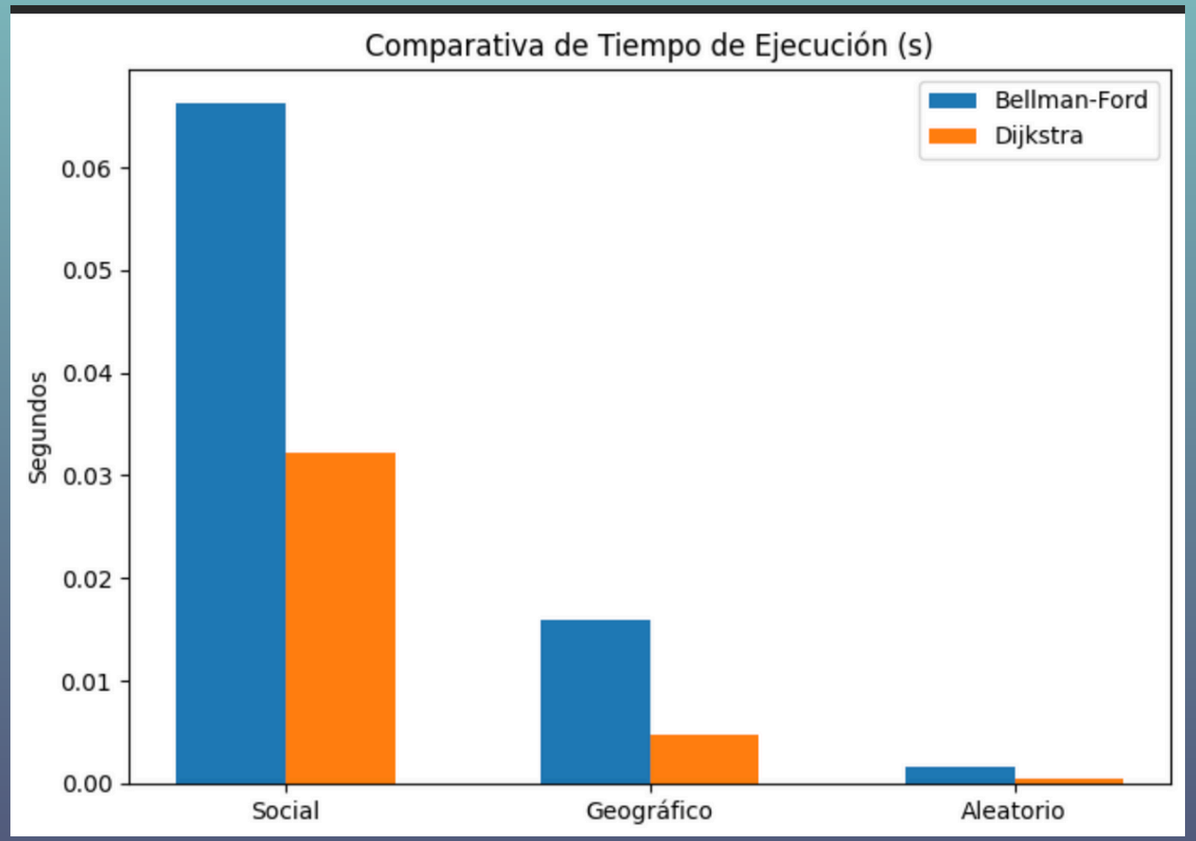
O (nm)

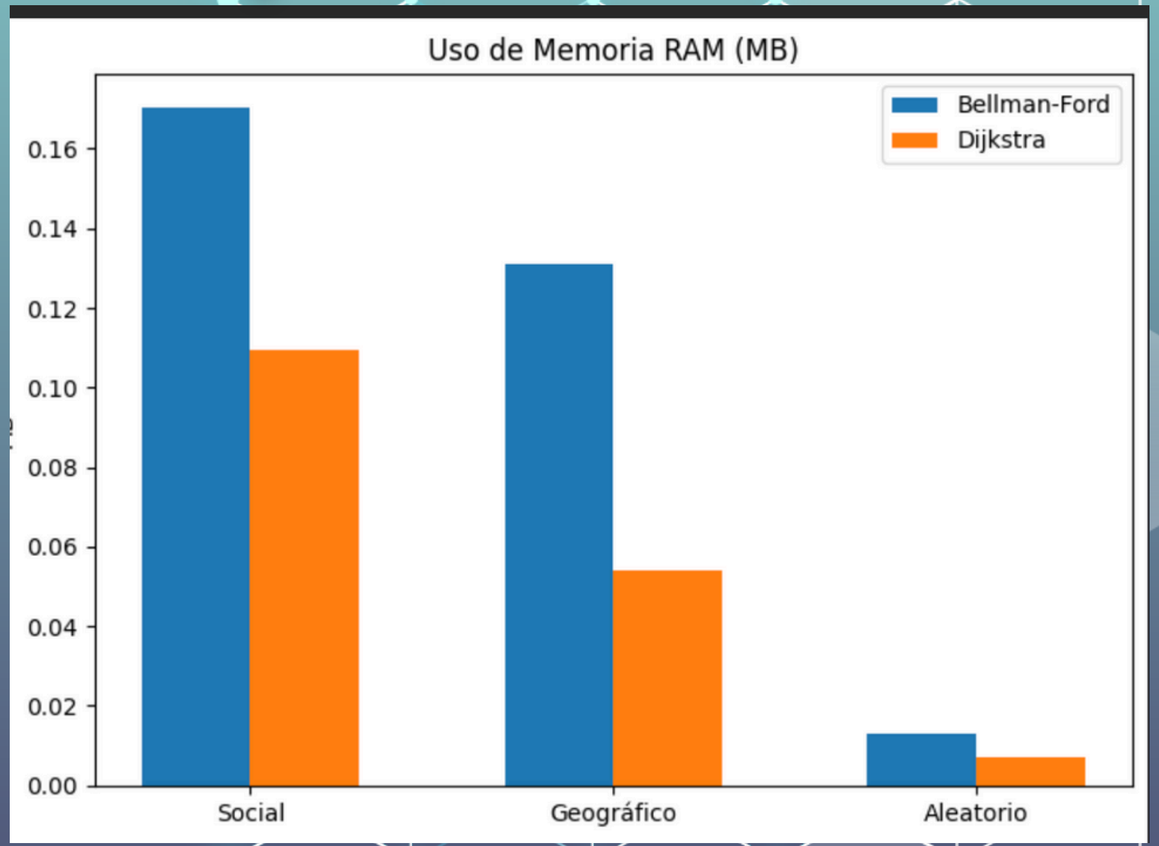


Ventaja Única: Puede manejar pesos negativos y, lo más importante, detectar ciclos negativos

Uso: Esencial en dominios financieros o protocolos de red donde la exactitud bajo cualquier condición es

Comparativa Experimental
Bellman-Ford vs Dijkstra





Velocidad: Dijkstra es entre 20 y 100 veces más rápido que Bellman-Ford en redes sociales y geográficas

Escalabilidad: Mientras Dijkstra maneja bien grafos de hasta 50,000 nodos, Bellman-Ford sufre una degradación de tiempo cuadrática que lo hace poco práctico para aplicaciones de gran escala.

Cuando elegir uno u otro?
Bellman-Ford vs Dijkstra

Situación	Algoritmo Ganador	Razón Técnica
Mapas y GPS	**Dijkstra**	Es de 20 a 100 veces más rápido en grafos grandes.
Detección de Fraude/Arbitraje	**Bellman-Ford**	Único capaz de detectar ciclos negativos.
Redes de Sensores (Energía)	**Bellman-Ford**	Robusto si hay nodos que "ganan" energía (peso negativo).
Grafos con millones de nodos	**Dijkstra**	Su complejidad $O(m \log n)$ escala mucho mejor que la de Bellman-Ford $O(nm)$.

02 algoritmos BFS vs Dijkstra

En el estudio de grafos, los algoritmos BFS (Breadth-First Search) y Dijkstra se utilizan para encontrar caminos entre nodos. Aunque ambos permiten hallar rutas óptimas, su funcionamiento, objetivos y aplicaciones son diferentes, por lo que es importante compararlos.

VS

BFS

Encuentra el camino más corto en grafos no ponderados

Dijkstra

Encuentra el camino más corto en grafos ponderados

Tipo de grafo y Manejo de pesos

BFS

- BFS trabaja sobre grafos no ponderados, es decir, donde todas las aristas tienen el mismo valor o costo.
- Dijkstra trabaja con grafos ponderados, donde cada arista tiene un peso o costo asociado.

Dijkstra

- BFS no admite pesos, o asume que todos valen lo mismo.
- Dijkstra admite pesos positivos, como distancia, tiempo o costo.

Dijkstra no funciona con pesos negativos.

Objetivo principal

- BFS busca el camino más corto en número de aristas.
- Dijkstra busca el camino de menor costo total, aunque tenga más pasos.

Diferencia conceptual clave:

- BFS minimiza pasos
- Dijkstra minimiza costo

Estrategia de exploración

- BFS explora el grafo nivel por nivel, avanzando de forma uniforme.
- Dijkstra explora según el menor costo acumulado, priorizando los nodos más baratos.

Camino más corto en aristas

Exploración nivel por nivel

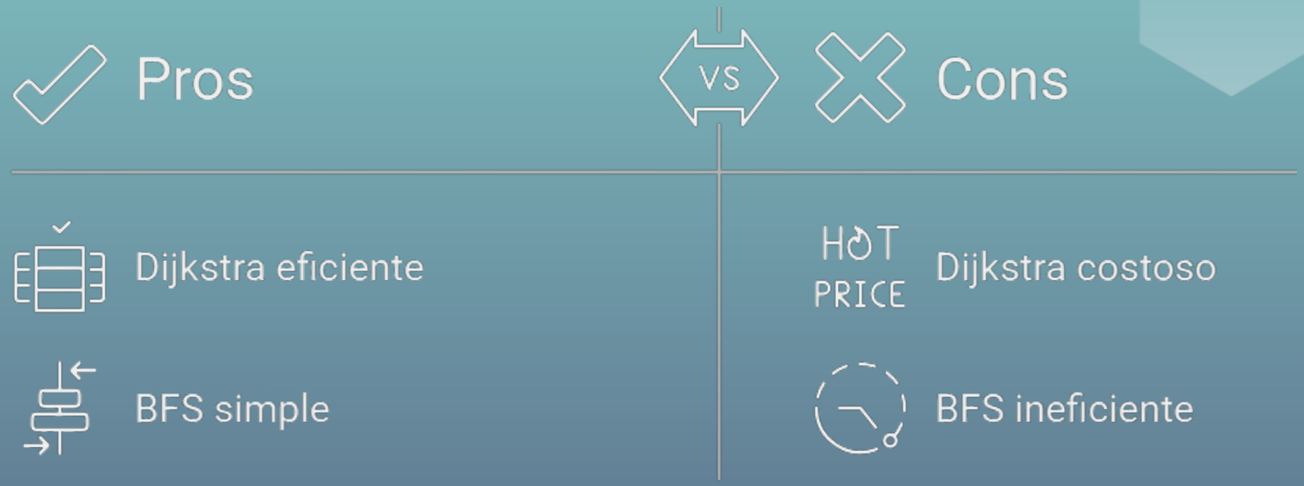
Camino de menor costo total

Exploración por costo acumulado

Estructuras de datos usadas

- BFS utiliza una cola (FIFO).
- Dijkstra utiliza una cola de prioridad (heap).

Esto hace que Dijkstra sea más costoso computacionalmente.



Estrategia de exploración

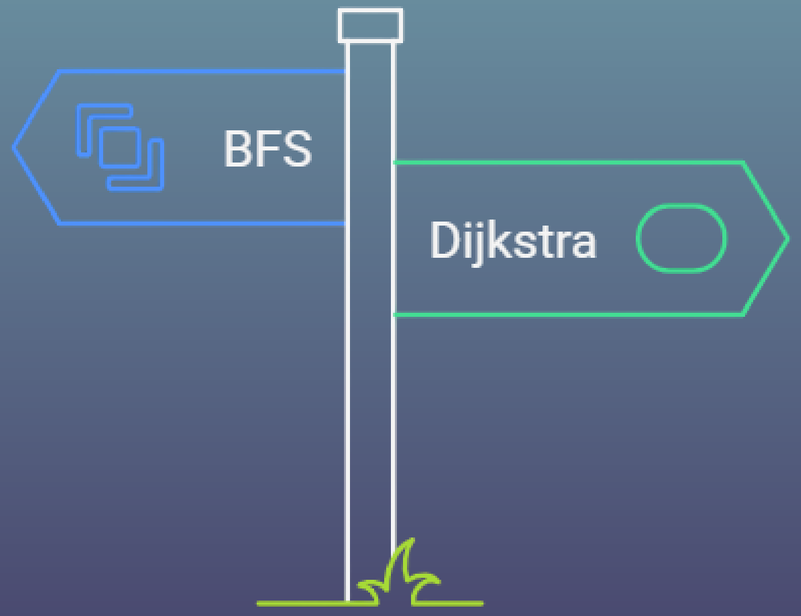
- **BFS:** $O(V + E)$
- **Dijkstra:** $O((V + E) \log V)$

- Donde:
- V = vértices
- E = aristas

BFS es más rápido en grafos grandes sin pesos.

Garantía de optimalidad

- BFS garantiza el camino más corto solo en grafos no ponderados.
- Dijkstra garantiza el camino mínimo si los pesos son positivos.



Complejidad de implementación

- BFS es simple de implementar y entender.
- Dijkstra es más complejo, requiere manejo de distancias y prioridades.

Memoria utilizada

- BFS usa menos memoria.
- Dijkstra usa más memoria debido a la cola de prioridad y el registro de costos.

Memoria utilizada

BFS

- Búsqueda en laberintos
- Redes sociales (menor número de conexiones)
- Análisis de niveles
- Sistemas educativos

Dijkstra

- Sistemas GPS
- Redes de transporte
- Redes de computadoras
- Optimización de rutas

Ventajas y desventajas

BFS

✅ Simple

✅ Rápido

❌ No maneja pesos

Dijkstra

✅ Maneja pesos

✅ Más preciso

❌ Más lento

❌ No acepta pesos negativos

Casos de uso ideales

Situación	Algoritmo recomendado
Grafo sin pesos	BFS
Grafo con pesos positivos	Dijkstra
Prioridad: velocidad	BFS
Prioridad: costo	Dijkstra

03

Algoritmo A* vs Dijkstra

El algoritmo A* (o A-estrella) es un algoritmo de búsqueda de caminos que encuentra la ruta de menor costo entre dos nodos en un grafo ponderado.

Lo hace combinando:
g(n): el costo real desde el inicio hasta el nodo actual
h(n): una heurística que estima el costo restante desde el nodo actual hasta la meta
Entonces calcula para cada nodo:
f(n) = g(n) + h(n)
y siempre expande (explora) el nodo con menor f(n).
Esto significa que A* no solo mira qué tan barato es llegar al nodo actual, sino también qué tan prometedor parece ese nodo para llegar rápido al destino, según la heurística.

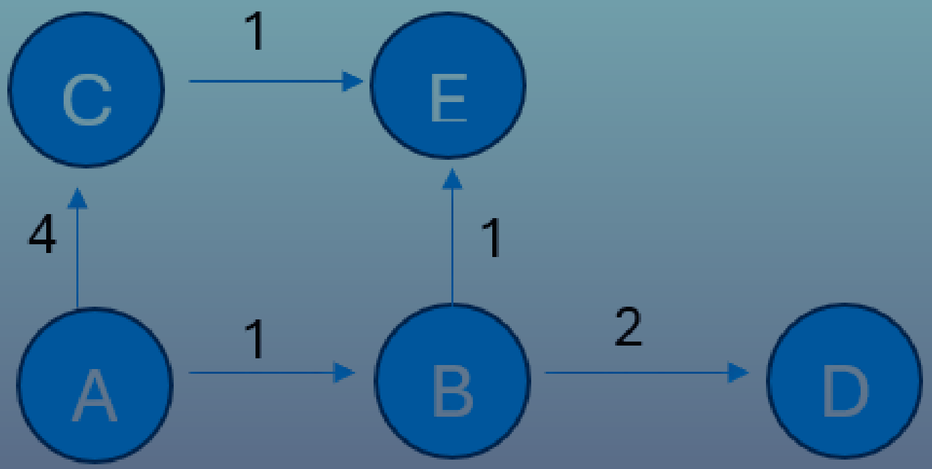
La heurística h(n) es una estimación de lo que falta para llegar al objetivo, por ejemplo:

- Distancia Euclidiana (línea recta)
- Distancia Manhattan (en un grid)

Una buena heurística debe ser admisible: nunca debe sobreestimar el costo real restante.

Eso le permite a A*:
Encontrar siempre la ruta más corta si la heurística es admisible
Reducir la cantidad de nodos explorados comparado con búsquedas sin heurística
Si la heurística siempre es cero (h(n)=0), A* se vuelve Dijkstra (es decir, idéntico).

EJEMPLO: A - > D



NODO	COORDENADA
A	(0,0)
B	(1,0)
C	(0,1)
D	(2,0)
E	(1,1)

EJEMPLO: A - > D

NODO	g	h	f
A	0	2	2
A - > B	1	1	2
A - > C	4	2.24	6.24
B - > D	3	0	3
B - > E	2	1.41	3.41

Casos en los cuales usar A*	Casos en los cuales no usar A*
Buscar un camino entre dos puntos específicos (A - > B). Ejemplo: ir desde una ciudad a otra A* está diseñado para este tipo de búsqueda dirigida.	Cuando no tienes una buena heurística.Si no puedes estimar razonablemente qué tan “cerca” está un nodo del objetivo, A* pierde su ventaja y se comporta como Dijkstra (o peor en tiempo).
Problemas con espacio físico o geométrico. Ejemplo: mapas, laberintos, grillas, planos 2D/3D. Aquí heurísticas como distancia euclidiana o Manhattan funcionan muy bien y reducen muchas exploraciones innecesarias.	Grafos sin relación espacial o lógica clara.Ejemplo: redes abstractas donde no existe una “distancia” estimable hacia la meta. En estos casos la heurística no aporta información útil.
Cuando necesitas rapidez en la búsqueda.Muy usado en videojuegos, robots móviles y sistemas de navegación, porque suele encontrar la solución explorando menos nodos que Dijkstra.	Cuando la memoria es un problema.A* guarda muchos nodos en memoria (abiertos y cerrados). En grafos muy grandes puede consumir demasiada memoria.
Cuando los costos de las aristas son positivos.A* funciona correctamente y garantiza la mejor ruta solo si no hay pesos negativos.	Si hay pesos negativos.A* no garantiza optimalidad con costos negativos, igual que Dijkstra.

Característica	A*	Dijkstra
Uso de heurística	Sí (guiada hacia la meta)	No
Orden de expansión	f = g + h	Solo g
Velocidad típica	Más rápido si heurística buena	Más lento pero sistemático
Garantía de ruta corta	Sí con heurística admisible	Sí
Explora menos nodos	En general sí	No
Aplicación típica	Camino A - > B	Árbol de caminos desde un origen

Graficos A* vs Dijkstra