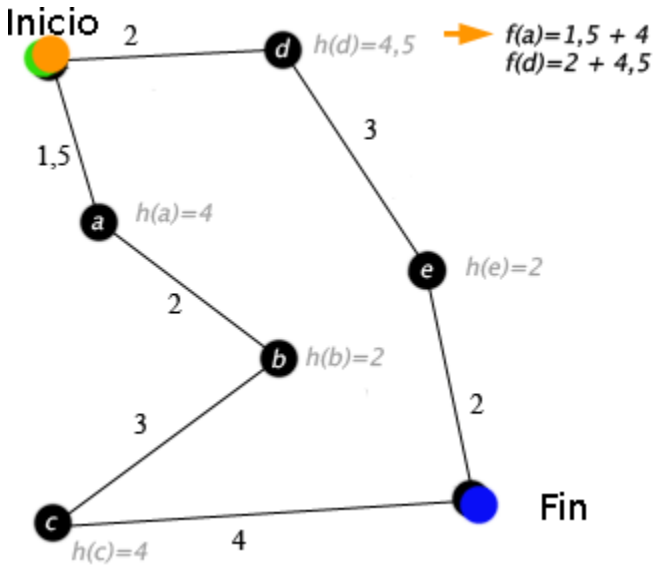


Complejidad Algorítmica

Unidad 1: Comportamiento asintótico, métodos de búsquedas y grafos

Módulo 5: Grafos - Técnicas de recorrido y búsquedas

Complejidad Algorítmica

Semana 5 / Sesión 2

MÓDULO 5: Grafos - Técnicas de recorrido y búsquedas

Contenido

Parte #2

1. Búsquedas por costo uniforme (UCS)
2. Búsquedas por profundidad
    - Profundidad limitada (DLS)
    - Profundidad iterativa (IDS)

Preguntas

1. Búsquedas por costo uniforme (UCS)

¿Qué es UCS en un grafo?

- La búsqueda de coste uniforme o **Uniform-Cost Search (UCS)** en inglés, es una estrategia de **<u>búsqueda informada</u>**.

  Un algoritmo de **búsqueda es informado**, cuando usa:

  - ✓ Información sobre el **<u>costo de la ruta</u>**
  - ✓ **Heurísticas** (funciones o reglas) para encontrar la solución.

- La función que guía el proceso de búsqueda toma el nombre de **función heurística**.

- **A-Star** (también conocido como **A***) así como el Algoritmo Dijkstra, son los **algoritmos de búsqueda HEURISTICA** más exitosos <u>utilizados para encontrar la ruta más corta entre **nodos** o **gráfos ponderados.**</u>

1. Búsquedas por costo uniforme (UCS)

ALGORITMO DIJKSTRA (Algoritmo de caminos mínimos)

- Utilizado para **encontrar la ruta más corta entre dos nodos en un grafo ponderado** (es decir, un grafo en el que las aristas tienen un peso o valor asociado positivo).
- Fue diseñado por el holandés **Edsger Wybe Dijkstra** en 1959.
- Este algoritmo consiste en ir visitando todos los caminos más cortos que parten del nodo inicial (origen) y que llevan a todos los demás nodos.
- Cuando se obtiene el camino más corto desde el nodo inicial hasta el resto de los nodos que componen el grafo, el algoritmo se detiene.

Edsger Wybe Dijkstra

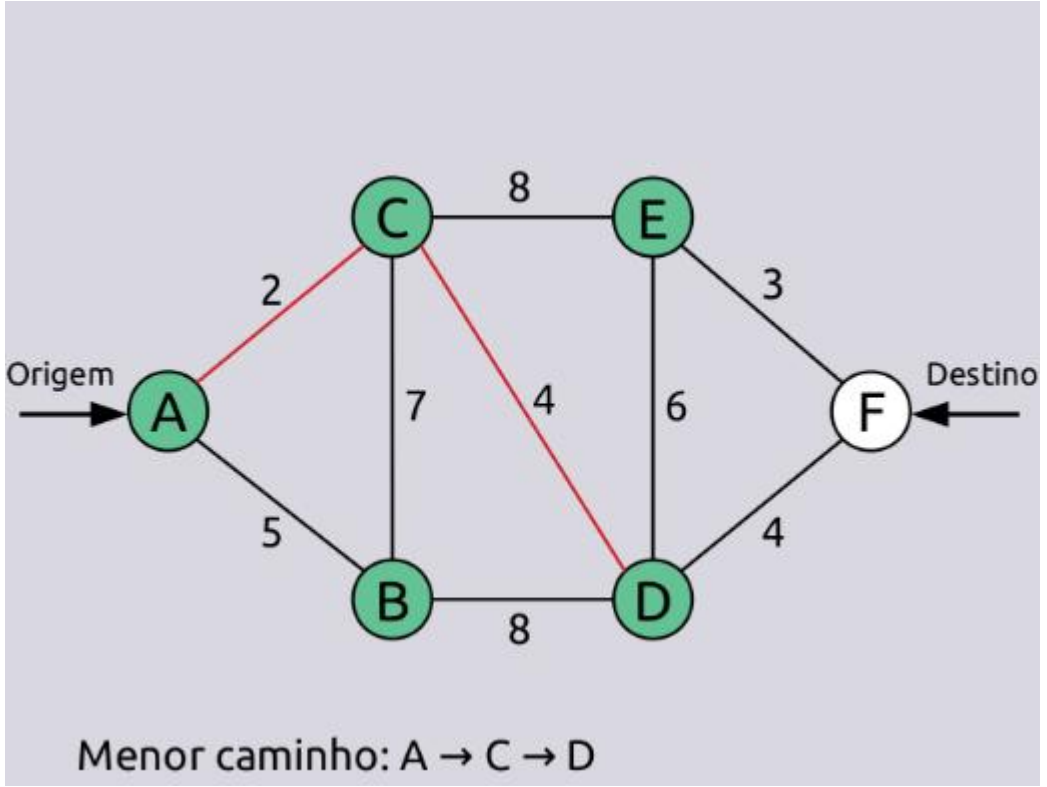
1. Búsquedas por costo uniforme (UCS)

ALGORITMO DIJKSTRA - En Python:

```
def dijkstra(G, s):
  n = len(G)
  visited = [False]*n
  path = [-1]*n
  cost = [math.inf]*n

  cost[s] = 0
  pqueue = [(0, s)]
  while pqueue:
    g, u = hq.heappop(pqueue)
    if not visited[u]:
      visited[u] = True
      for v, w in G[u]:
        if not visited[v]:
          f = g + w
          if f < cost[v]:
            cost[v] = f
            path[v] = u
            hq.heappush(pqueue, (f, v))

  return path, cost
```



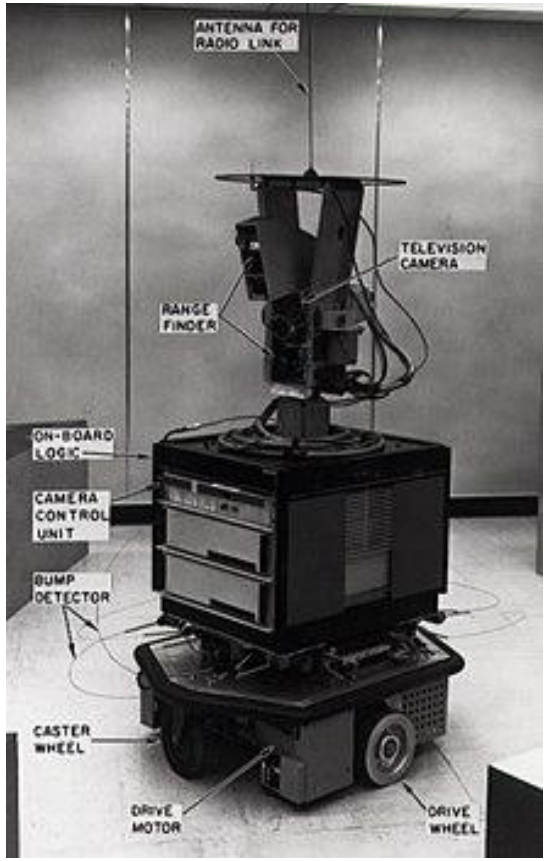
1. Búsquedas por costo uniforme (UCS)

ALGORITMO A*

- Fue publicado por primera vez en 1968 por Peter Hart, **Nils Nilsson** y **Bertram Raphael.**

- **A*** fue inventado por investigadores que trabajaban en la planificación del camino de Shakey the Robot.

- Inicialmente fue considerado una extensión del algoritmo de Dijkstra, pero resulta ser más rápido que este debido a que utiliza la función heurística.



1. Búsquedas por costo uniforme (UCS)

ALGORIMO A*: UTILIZADO PARA LA SOLUCION DE PROBLEMAS

Objetivo: lograr la **optimización** y la **completitud**, dos propiedades valiosas de los algoritmos de búsqueda.

Optimización: significa la garantía de encontrar la mejor solución posible.

Completitud: significa que si existe una solución a un problema dado, el algoritmo garantiza encontrarla.

1. Búsquedas por costo uniforme (UCS)

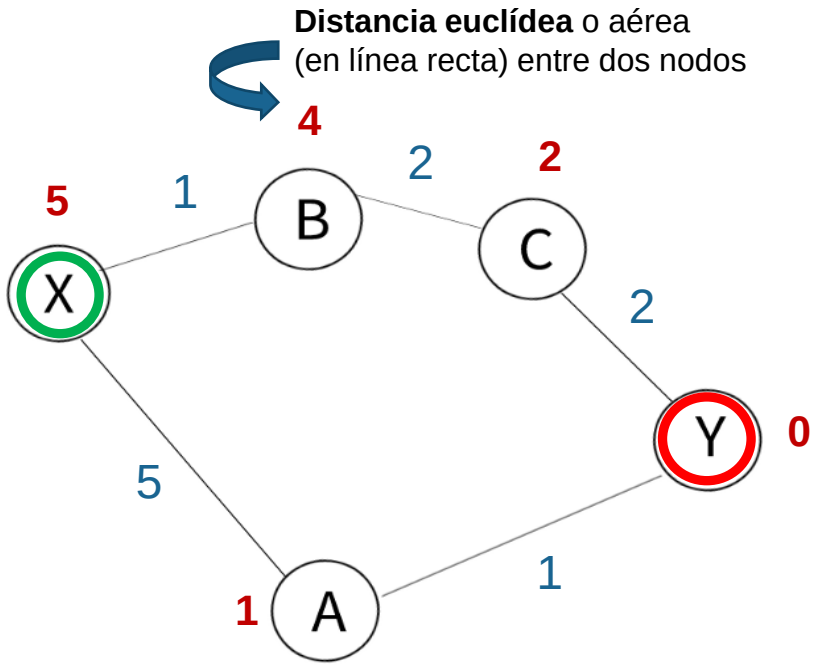
ENTENDAMOS COMO ES LA ESTRUCTURA DE UN GRAFO PONDERADO

Costo: valor numérico (por ejemplo, distancia, tiempo o gasto financiero) para la ruta de un nodo a otro nodo.

g (n) : el costo de la ruta entre el primer nodo y el nodo actual (dato del problema)

h (n) : función heurística (costo estimado heurístico desde el nodo n hasta el nodo objetivo)

f (n) : Costo total calculado de la ruta (elegiremos el costo menor)



1. Búsquedas por costo uniforme (UCS)

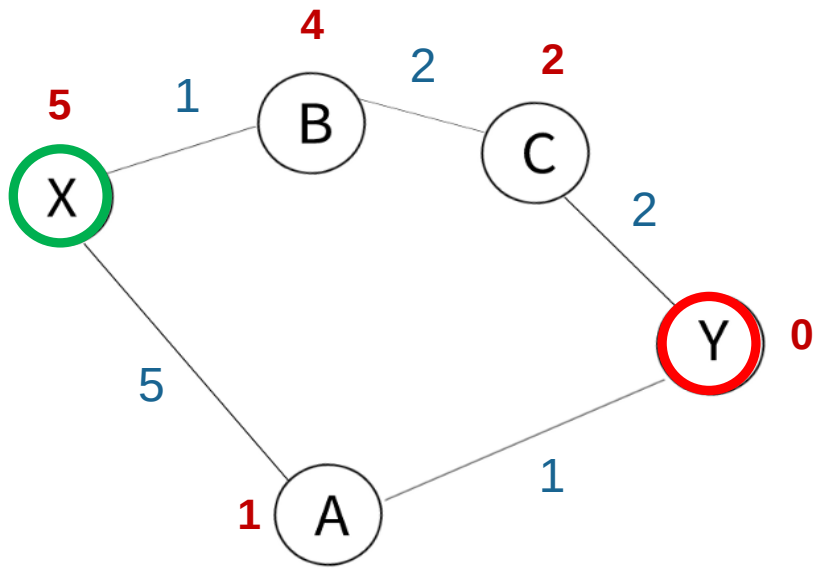
Costo total calculado de la ruta

f (n) = g (n) + h (n) = Costo total calculado de la ruta

Donde:

g (n): el costo de la ruta entre el primer nodo y el nodo actual (dato del problema)

h (n): función heurística (costo estimado heurístico desde el nodo n hasta el nodo objetivo)

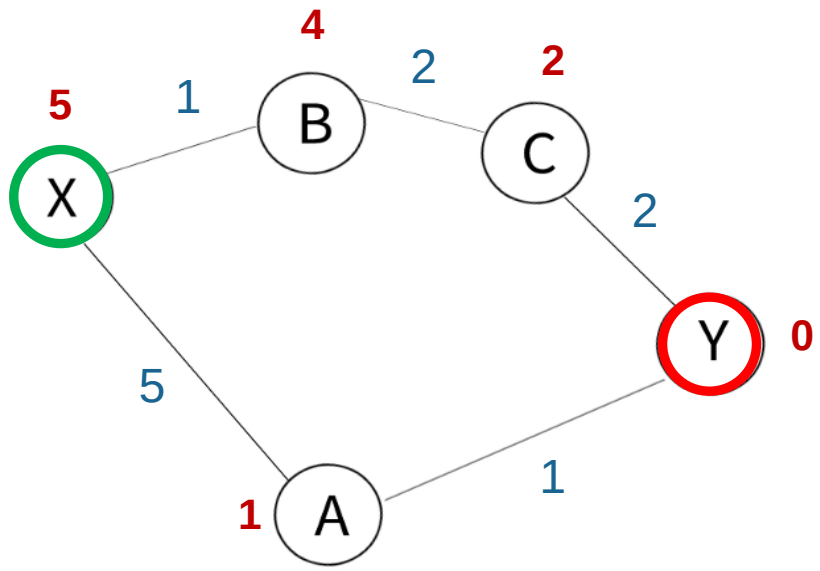


Pregunta: Usando la **función f(n)** para calcular el costo de la ruta ¿Cuál sería la ruta más corta para ir de X-Y?

1. Búsquedas por costo uniforme (UCS)

EJEMPLO #1: Utilizando la distancia euclidiana.

Pregunta: Usando la **función f(n)** para calcular el costo de la ruta ¿Cuál sería la ruta más corta para ir de **X=>Y**?



Ruta 1: XAY

X - A => g (A) + h(A) = 5 + 1 = 6

A - Y => g (Y) + h(Y) = 6+ 0 = 6

Ruta 2: XBCY

X— B => g (B) + h(B) = 1+ 4 = 5

B - C => g (C) + h(C) = 3+ 2 = 5

C - Y => g (Y) + h (Y) = 5 + 0 = 5

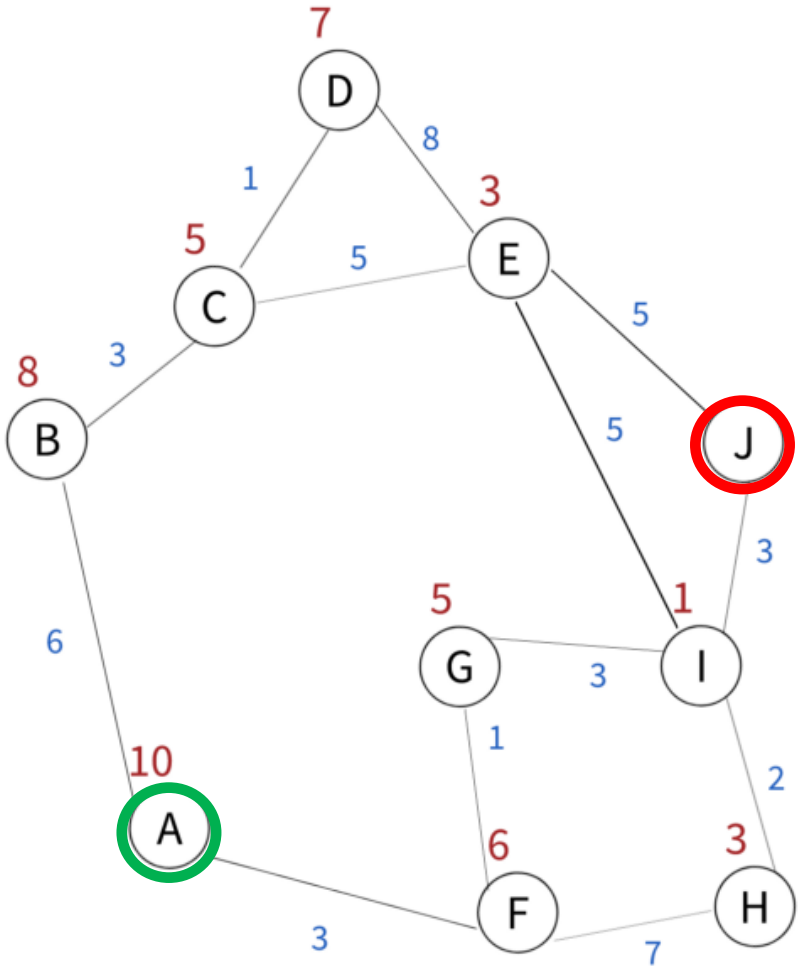
Solución: la ruta más corta es la ruta 2 = **XBCY**. El costo de esta vía es de 5 unidades, mientras que el costo de la ruta alternativa XAY es de 6 unidades.

g: recordar que se calcula el costo desde el nodo inicial (X)
h: recordar que se calcula el costo desde el nodo actual (n)

1. Búsquedas por costo uniforme (UCS)

Pregunta:

Usando la **función f(n) = g(n) + h(n)** para calcular el costo de la ruta ¿Cuál sería la ruta más corta para ir de **A=>J**?



Solución:

- Nos dan el costo **g(n)** de cada borde y el **h(n)** para cada nodo
- Desde el nodo A hay 2 puntos (B y F) para llegar a J.
- Calculamos los costos generales para elegir el de menor costo:

A – B => f(B) = g(B) + h(B) = 6 + 8 = 14

A – F => f(F) = g(F) + h(F) = 3 + 6 = 9

A* continúa desde aquí por ser el de menor costo.

1. Búsquedas por costo uniforme (UCS)

Continuación:

- Desde el **nodo F** hay 2 puntos (G y H) para llegar a J.

  F – G => f(G) = g(G) + h(G) = 4 + 5 = 9 ✓ **A *** continúa desde aquí por ser el de menor costo.

  F – H => f(H) = g(H) + h(H) = 10 + 3 = 13

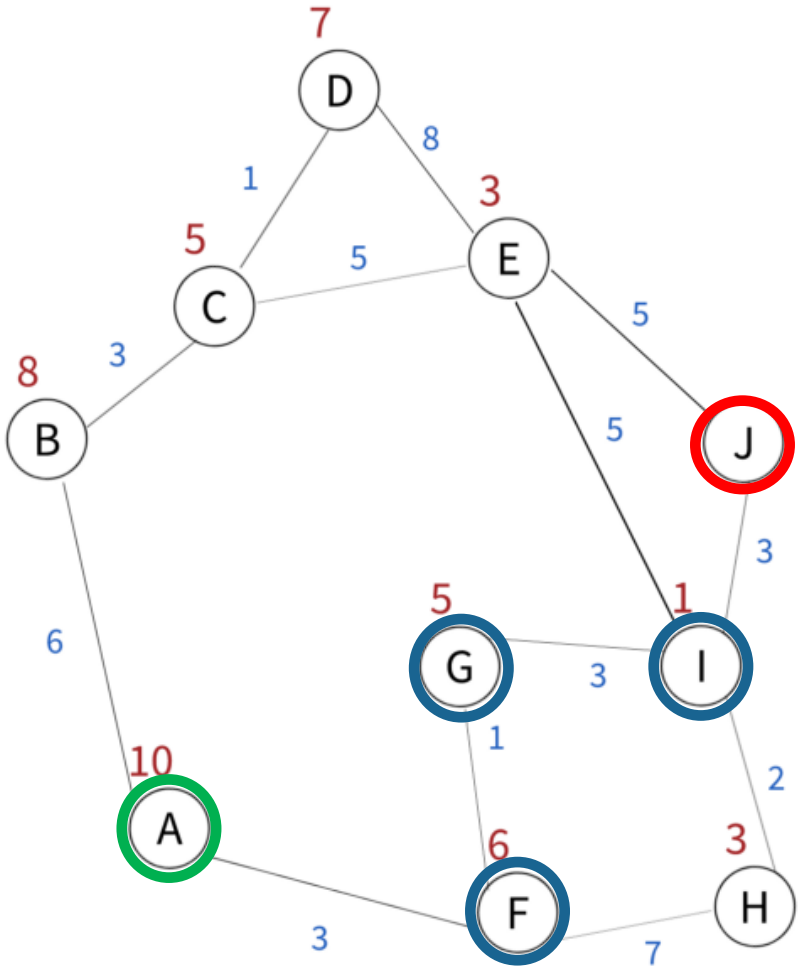
- Faltaría seguir de G – I para llegar a J

  G – I => f(I) = g(I) + h(I) = 7 + 1 = 8

  I – J => f(J) = g(J) + h(J) = 10 + 0 = 10

- Dado que todos los valores obtenidos después de ir al nodo F resultaron menores que f (B) = 14, no regresamos al nodo B.

- Bastaría que f(G) o f(I) sean mayores a f(B) = 14 para que el algoritmo se interrumpa. En este caso, según el algoritmo A *, el proceso se interrumpe aquí y la ruta continúa con el nodo B. Aquí, tan pronto como f (C)> f (I).



1. Búsquedas por costo uniforme (UCS)

APLICACIONES DEL ALGORITMO A*

A-star (A *) es un poderoso algoritmo en Inteligencia Artificial con una amplia gama de usos.

- **A *** es la opción más popular para **la búsqueda de rutas (mapas)**.

  En el planeamiento de rutas, estimar el costo del camino más barato puede ser la distancia en línea recta entre dos ciudades.

  Aplicado a problemas de caminos/trayectorias se utiliza la heurística:

  - ✓ **Distancia Euclídea (o aérea):** distancia en línea recta entre dos nodos
  - ✓ **Distancia Manhattan:** distancia entre dos puntos es la suma de las diferencias de sus coordenadas

- En **Machine Learning** y optimización de búsqueda.
- En el **desarrollo de juegos** donde los personajes navegan a través de terrenos complejos y obstáculos para llegar al jugador.

2. Búsquedas por profundidad

Hasta el momento ya conocemos:

- La búsqueda en profundidad (DFS)
- La búsqueda en amplitud (BFS)
- La búsqueda de coste uniforme (BCU)

¿Qué más sigue ?

De la **búsqueda en profundidad (DFS)**, podemos aplicarle algunas modificaciones, para:

- Limitar la profundidad **-> Búsqueda por profundidad limitada (DLS)**
- Hacerla iterativa **-> Búsqueda por profundidad iterativa (IDS)**

Veamos a que se refieren estos cambios…

2. Búsquedas por profundidad

Búsqueda por profundidad limitada (DLS)

¿Por que limitamos la profundidad de la búsqueda?

Porque ante un espacio de estados (nodos) infinito

Podríamos entrar en una búsqueda infinita.

Y estaríamos en un problema de callejón sin salida…

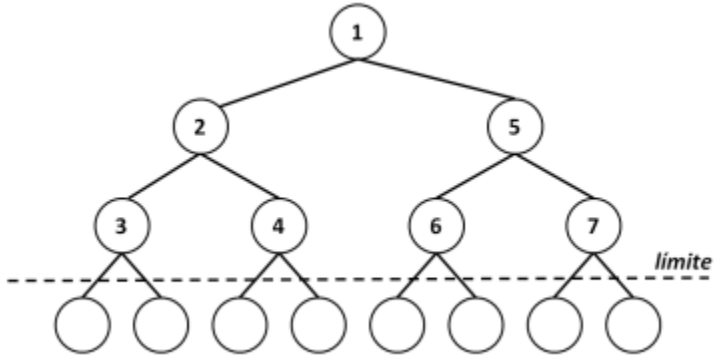
Ante esa situación, debemos limitar nuestras búsquedas en profundidad hasta recorrer cierto nivel de estados.

2. Búsquedas por profundidad

Búsqueda por profundidad limitada (DLS)

¿Que logramos aplicando una búsqueda DLS?

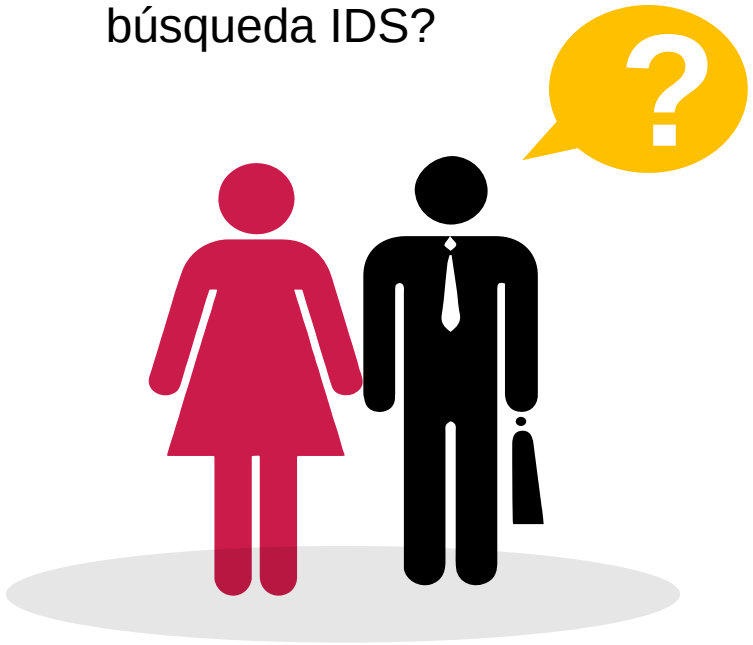
- Limitar la profundidad máxima de una ruta para la **búsqueda por profundidad (DFS)**.
- Establecer un límite sin conocer mucho sobre el espacio de estados.
- Obtener un resultado que puede no ser completa ni óptima:
  - Un límite muy pequeño puede no contener la solución.
  - Un limite muy grande puede contener muchas soluciones no óptimas.



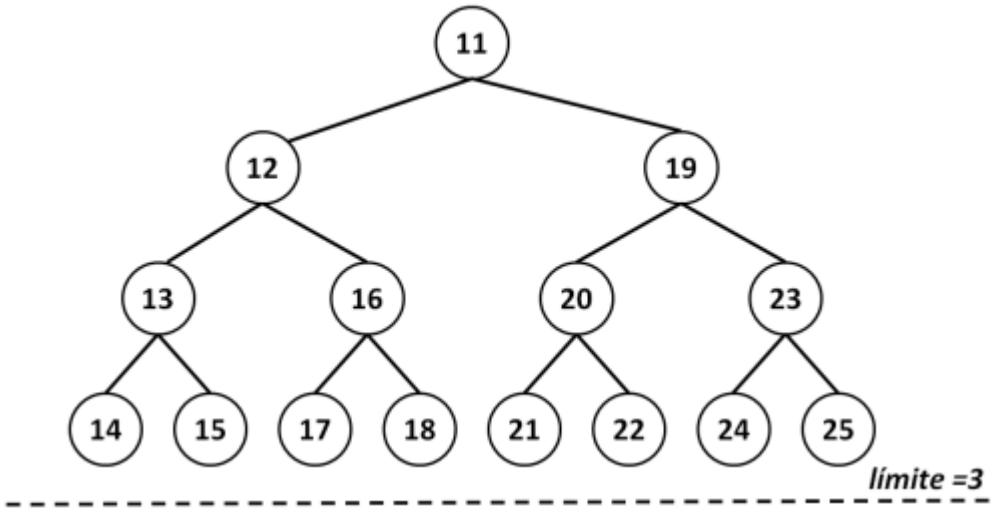
2. Búsquedas por profundidad

Búsqueda por profundidad iterativa (IDS)

¿En que consiste la búsqueda IDS?



- Consiste en iterar la búsqueda de profundidad limitada (DLS) incrementando gradualmente el límite, hasta encontrar un objetivo.

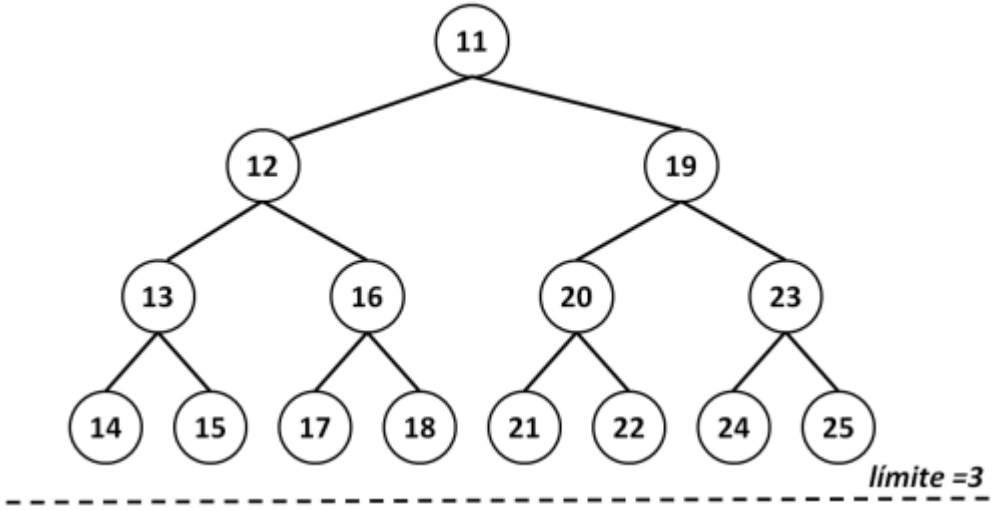


2. Búsquedas por profundidad

Búsqueda por profundidad iterativa (IDS)

¿Que logramos aplicando una búsqueda IDS?

- Eliminar la dificultad de elegir un límite adecuado de profundidad en el DLS.
- Probar todos los límites de profundidad posibles, primero la profundidad 0, luego la 1, luego la 2, etc.
- Combinar las ventajas de las búsquedas por profundidad y por amplitud.



PREGUNTAS

Dudas y opiniones