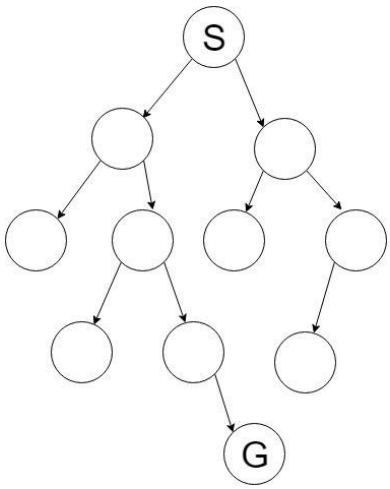
Tarea 1

Inteligencia Artificial

Francisco Flores

1. En este espacio de búsqueda (figura de la derecha) *b=2*, *d=4* y *m=4*. Para el algoritmo búsqueda en profundidad iterativa, llegar desde el nodo inicial S al nodo objetivo G le cuesta expandir mayor cantidad de nodos, pues *l=4* (debe iterar cuatro veces), lo que le hace visitar todos los nodos por cada nivel. Es decir tiene que hacer 1+3+7+10+6 = 27 operaciones en su ejecución hasta encontrar el objetivo. Mientras que el algoritmo búsqueda en profundidad expande el nodo más profundo (y más a la izquierda) por lo que tiene que hacer solo 7 operaciones de expandir nodo, lo que implica que no recorre el árbol de búsqueda completo. Finalmente, en este caso, búsqueda en profundidad iterativa tiene un peor desempeño que búsqueda en profundidad.



2. Problema de la mesa y los cuatro bloques

2.1. Formulación del problema

- Estados: Cada una de las posibles configuraciones
  Se define la estructura {b1, b2, b3, b4} como un estado tal que los bloques b1, b2, b3, b4 están sobre la mesa y {(bi, bj)} significa que el bloque bi está sobre la mesa y bj sobre bi. Luego siguiendo estas definiciones, algunos de los estados del problema son:
  {A, B, C, D} ; {A, B, (D, C)} ; {(A, B), (D,C)} ; {(B, A), (D, C)} ;
  {A, (B, C, D)} ; {D, (C, B, A)} ; {A, D, (B, C)} etcétera.
- Estado inicial: Bloques A, B y D sobre la mesa y C sobre D
  = {A, B, (D, C)}
- Test objetivo: Retorna Verdadero si A está sobre B, B sobre C y C sobre la mesa, y D sobre la mesa = {D, (C, B, A)}. En cualquier otro caso el test objetivo retorna Falso.

- Acciones: para el estado {A, B, C, D} las posibles acciones son: {C, D, (B, A)} ; {B, D, (C, A)} ; {B, C, (D, A)} es decir el bloque A sobre cualquier otro y los restantes sobre la mesa. Lo mismo para B, C y D.

  En general en cualquier estado, el estado siguiente será poner un bloque que está sobre la mesa encima de cualquier otro, o quitar un bloque que está encima de otro y ponerlo sobre la mesa. Cumpliendo con la regla de mover un bloque a la vez siempre que no tenga otro encima.

2.2. h(n) = cantidad de bloques sobre la mesa - cantidad de bloques sobre otro bloque. Esta heurística no es admisible, pues sobreestima el costo de alcanzar el objetivo. por ejemplo h({(A, B), (C, D)}) = 2 - 2 = 0.

3. En A* la función a evaluar para ir de un nodo n hasta un nodo sucesor es
f(n) = g(n) + h(n), donde
g(n) = costo del camino y
h(n) = valor de heurística.
Por lo que f(n) es el costo más barato estimado de la solución.
Cuando se enfatiza g(n) o h(n) como por ejemplo en f(n) = (1-w)g(n) + wh(n), sucede que si w es tal que pondera más en h(n) sería equivalente a una búsqueda greedy, pues el valor del costo de camino sería tan bajo que casi no habría diferencia entre f(n) y wh(n). Por otro lado si w pondera más en g(n) la bùsqueda sería equivalente a búsqueda por costo uniforme, pues el algoritmo elegiría el nodo con el menor costo y h(n) no afectaría al resultado.

4. Suponiendo h_1 y h_2 admisibles

4.1. h_4(n) = max{ h_1(n), 1.1*h_2(n) }
En este caso 1.1*h_2(n) domina a h_1(n) por lo que no se puede asegurar que h_4(n) sea admisible ya que estaría sobreestimando el costo de llegar al objetivo.

4.2. h_5(n) = max{ h_1(n), h_2(n)/2 }
en este caso se elige h_1(n) y como h_1 es admisible, en consecuencia h_5 también lo es. En caso de escoger h_2, h_5 también sería admisible.

4.3. h_6(n) = min{ h_1(n), h_2(n)/2 }
aquí h_1 y (h_2)/2 son admisibles (h_2 disminuye: no sobreestima) por lo que escoger una u otra hará que h_6 sea admisible.

De las 3 elegiría h_5 pues no se alteraría el valor al mantenerla igual a h_1.

5. Programación

5.1. Ver código en archivo adjunto. Es un Jupyter Notebook. Para ejecutarlo basta abrirlo en jupyter y ejecutar celda a celda. O bien en hacer clic en cell, luego en Run All.

5.2.

5.2.1. Búsqueda en profundidad encuentra la solución óptima en algunos casos (no siempre), debido a que escoge un nodo hijo al azar. Greedy y A* no encuentran la solución óptima pues la heurística no es admisible. Por último el único algoritmo que encuentra el camino óptimo es búsqueda por costo uniforme, pues solo considera los pesos entre los nodos.

5.2.2. No, ningún algoritmo de los vistos en clase se mantiene haciendo una busqueda ad infinitum, pues el grafo no tiene ciclos.