Tarea 1 - Inteligencia Artificial

Entrega: Miércoles 26 de Abril

Nombre: Dazhi Feng Zong

1.- (10 pts) De un ejemplo de un espacio de búsqueda (considerando factores como factor de ramificación, profundidad, ubicación del objetivo en profundidad) en el cual el método de búsqueda de profundidad iterativa tenga un peor desempeño que el de búsqueda en profundidad (básico). Puede usar un ejemplo artificial, es decir, el espacio de estados puede no corresponder a ningún problema real.

R: Un ejemplo, podría ser un grafo muy profundo con un factor de ramificación grande y la ubicación del objetivo esté profundo. Entonces, en este caso la búsqueda en profundidad (básica) tiene mejor desempeño al buscar la solución en profundidad, ya que, la búsqueda de profundidad iterativa tendrá que realizar varias iteraciones para expandir en profundidad antes de llegar a los niveles más profundos, y como el factor de ramificación es grande, necesitará mucho más tiempo y tendrá peor desempeño.

2.- Considere un problema en el cual tiene una mesa y cuatro bloques, del mismo tamaño. Cada bloque es identificado por una letra, A, B, C o D. Cada bloque puede estar en la mesa, o sobre otro bloque. Al comienzo, los bloques A, B y D están sobre la mesa, y el bloque C está sobre el bloque D. Un bloque puede moverse a la vez, sólo si no tiene otro bloque encima. Cada bloque puede moverse ya sea a la mesa o sobre otro bloque (si es que no hay nada sobre ese bloque). El objetivo es crear una torre con los bloques A, B y C, donde A es el bloque de más arriba y C el de más abajo (sobre la mesa).

2.1.- (10 pts) Describa la formulación del problema (cuáles son los estados, estado inicial, test objetivo y acciones)

R:

Estados:

- Bloque en la mesa.
- Bloque sobre otro bloque.

Estado inicial:

- Los bloques A, B y D están sobre la mesa.
- Bloque C está sobre el bloque D.

Test Objetivo:

- Una torre donde:
    - C esta sobre la mesa.
    - Bloque B está sobre el bloque C.
    - Bloque A está sobre el bloque B.
- D está sobre la mesa.

Acciones:

- Si no hay nada sobre un bloque, el bloque puede:
    - Si está en la mesa, moverse otro bloque.
    - Si está sobre un bloque, moverse a la mesa.
    - Si está sobre un bloque, moverse a otro bloque.

2.2.- (5 pts) Proponga una heurística no trivial (no h(n)=constante). ¿Es la heurística propuesta admisible?

R: h(n) = número de bloques debajo del bloque.

Esta heurística representa el mínimo costo de movimientos para poner un bloque. Por ejemplo, si quiero poner A sobre B (que está sobre C), el número mínimo de movimientos es 2 (poner B y después poner A sobre), entonces es admisible, porque es el mínimo y sería menor o igual al costo real.

3.- (5 pts) ¿Es el algoritmo de Hill Climbing apropiado para el problema de misioneros y caníbales? Justifique

R: No, ya que el algoritmo de Hill Climbing es de tipo greedy y puede encontrar un máximo local rápidamente. Pero, para el problema misioneros y caníbales, esto no es ideal, ya que se necesita una visión de todo el problema para tomar la siguiente decisión, en lugar de tomar la "mejor" decisión actual (máximo local), pero que después esto pueda afectar el problema y no encontrar la solución.

4.- (10 pts) Suponga que tiene dos heurísticas admisibles, h1 y h2. Ud. decide crear nuevas heurísticas de la siguiente manera:
h3(n) = max (h1(n), h2(n))

h4(n) = max (h1(n), 1.1*h2(n))

h5(n) = min(h1(n), 3*h2(n))

h6(n) = (h1(n) + h2(n))/2

Para cada una de estas nuevas heurísticas, especifique si son, o no, admisibles. Justifique su respuesta. ¿Usaría ud. alguna de estas nuevas heurísticas en lugar de h1 o h2?

R:

1. h3(n) = max (h1(n), h2(n)). **Admisible**, ya que, como h1 y h2 son admisibles, al seleccionar cualquiera de ellos dos, la heurística sigue siendo admisible.
2. h4(n) = max (h1(n), 1.1*h2(n)). **No admisible**, ya que h2, al multiplicarse por 1.1 puede sobreestimar el costo real. h4 elige este nuevo máximo que podría sobreestimar el costo y no sería admisible.
3. h5(n) = min(h1(n), 3*h2(n)). **Admisible**. Como h1 es admisible, si el nuevo h2 sobreestima el costo real, igual se escogerá h1, que es el mínimo y h5 seguiría siendo admisible.
4. h6(n) = (h1(n) + h2(n))/2. **Admisible**, ya que, se toma el promedio de h1 y h2, que es menor o igual a h1, h2, lo que sigue siendo admisible.

No usaría h4, ya que no es admisible. Dependiendo del problema, podría utilizar h3, h5 o h6, aunque h3 y h5 son más redundantes.

5.- Programación. Debe adjuntar un enlace a repositorio github con su código fuente documentado e instrucciones para su ejecución (se descontará puntaje de no contar con una documentación adecuada y/o instrucciones). Su código debe ser creado en C, C++ o Python, debe ser original (se comparará con el de sus compañeros y de otros repositorios), y no se pueden utilizar bibliotecas más allá de las que facilitan el uso de listas o arrays (por ej., numpy en Python). Cualquier evidencia de copia implicará una nota 1.0 en la evaluación.

5.1.- (25 pts) Implemente un programa (en C, C++ o Python) que, utilizando una estructura de búsqueda tipo árbol, encuentre una solución para el problema de encontrar una ruta entre A y H, diagramado en la Figura 1. Debe utilizar los siguientes métodos:

- Búsqueda en profundidad (escogiendo un sucesor al azar)
- Búsqueda por costo uniforme
- Búsqueda greedy
- A*

Para cada tipo de búsqueda, su código debe retornar:
- El camino encontrado y su costo (y si es la solución óptima, que ud. puede calcular a mano)
- Cantidad de nodos expandidos (veces por nodo y en total)

Su código debe leer el problema (grafo) desde un archivo .txt con el siguiente formato:
Init: <nodo_inicial>
Goal: <nodo_objetivo>
<Nodo1> <valor_heuristica1>
<Nodo2> <valor_heurística2>
…..
<Nodo1>, <nodo2>, <costo> // en el caso en que exista una arista entre nodo1 y nodo2 …..

Su código podría ser evaluado, además, cambiando los nodos de inicio y/o objetivo. El formato de salida debe ser:
Nodo_inicial → nodo1 → nodo2 …->nodo_objetivo
Costo: <costo>
<nodo_inicial>: número de veces que se expandió
<nodo1>: número de veces que se expandió ….

R: https://github.com/dazfz/IA-Tarea1

5.2 .- (10 pts) Responda:

- ¿Qué puede decir de la comparación entre los métodos implementados? En concreto, los que encontraron la solución óptima, a qué se debió? Y los que no, ¿por qué no la encontraron? - De los métodos vistos en clase, ¿hay alguno que NO retorne una solución (es decir, que se mantenga realizando búsqueda ad infinitum) en este problema? Si es así, cuál? Y si no, por qué no?

R: Para

- Búsqueda en profundidad: Como se escoge un sucesor al azar, a veces encuentra la solución óptima y a veces no, debido a la aleatoriedad del siguiente nodo a expandir, podría irse por el camino óptimo o no óptimo.
- Búsqueda por costo uniforme: Encuentra el camino óptimo, ya que compara y elige el camino con el menor costo acumulado para llegar a un nodo. Entonces siempre se elige el camino menos costoso para encontrar la solución. Además, si encuentra la solución, pero hay otros caminos sin explorar con costos acumulados menores, expande hacia esos nodos para comprobar/buscar el camino óptimo.
- Búsqueda greedy: No encuentra el óptimo, ya que elige el vecino con heurística menor actual, e ignora otros caminos que pueden tener mayor heurística actual, pero que a largo plazo podrían resultar en un camino menos costoso.
- A*: Encuentra el camino óptimo, ya que para los nodos pertenecientes al camino óptimo, se cumple el criterio de admisibilidad.

Dado que el grafo es dirigido y no contiene ciclos, la búsqueda siempre retornará, ya que no caerá en un ciclo infinito. Esto se debe a que la ausencia de loops en el grafo y el tipo de grafo, que es dirigido, asegura que la búsqueda no retrocederá (backtracking) y se detendrá al encontrar un nodo hoja (o solución, dependiendo del tipo de búsqueda).

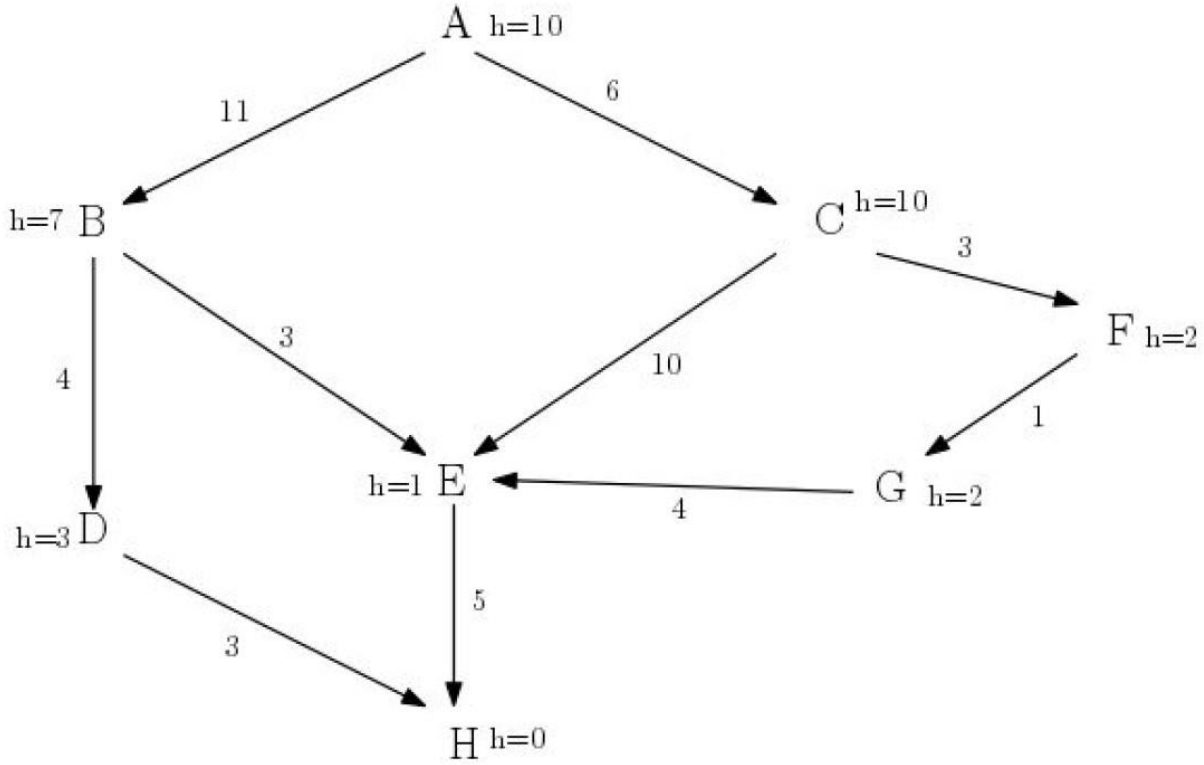

Figura 1