Práctica 1
Búsqueda no informada

Asignatura: Inteligencia Artificial
Fecha: 01/10/2020
Autor: Diego Marco Beisty, 755232

ÍNDICE

Ejecución de 8-puzzle con algoritmos de búsqueda no informada

Algoritmos utilizados:
Búsqueda Primero en Anchura (BFS) tanto en árbol como en grafo. Búsqueda Primero en Profundidad (DFS) tanto en árbol como en grafo. Búsqueda Primero en Anchura Limitada (DLS) con implementación recursiva. Búsqueda en Profundidad Iterativa (IDS), Búsqueda de Coste Uniforme (UCS) tanto en grafo como en árbol.

Partiendo de que todas las acciones tienen el mismo coste, se han ejecutado todos estos algoritmos para la modelización del problema 8-puzzle con tres estados iniciales distintos.
El estado inicial “boardWithThreeMoveSolution” donde la solución óptima está a profundidad 3.
El estado inicial “random1” donde la solución óptima está a profundidad 9.
El estado inicial “extreme” donde la solución óptima está a profundidad 30.

<terminated> EightPuzzlePract1 [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/b

Problema	Profundidad	Expand	Q.Size	MaxQS	tiempo
BFS-G-3	3	5	4	5	9
BFS-T-3	3	6	9	10	1
DFS-G-3	59123	120491	39830	42913	810
DFS-T-3	---	---	---	---	(1)
DLS-9-3	9	10	0	0	1
DLS-3-3	3	4	0	0	1
IDS-3	3	9	0	0	0
UCS-G-3	3	16	9	10	1
UCS-T-3	3	32	57	58	1
BFS-G-9	9	288	198	199	23
BFS-T-9	9	5821	11055	11056	16
DFS-G-9	44665	141452	32012	42967	401
DFS-T-9	---	---	---	---	(1)
DLS-9-9	9	5474	0	0	8
DLS-3-9	0	12	0	0	0
IDS-9	9	9063	0	0	11
UCS-G-9	9	385	235	239	3
UCS-T-9	9	18070	31593	31594	45
BFS-G-30	30	181058	365	24048	507
BFS-T-30	---	---	---	---	(2)
DFS-G-30	62856	80569	41533	41534	341
DFS-T-30	---	---	---	---	(1)
DLS-9-30	0	4681	0	0	3
DLS-3-30	0	9	0	0	0
IDS-30	---	---	---	---	(1)
UCS-G-30	30	181390	49	24209	521
UCS-T-30	---	---	---	---	(2)

Como se observa en la tabla, para cada ejecución se han obtenido distintas métricas relevantes.
Estas métricas son:

La profundidad de la solución encontrada, el número de nodos expandidos totales, el tamaño de la frontera cuando se encontró la solución, el tamaño máximo de la frontera en la ejecución de la búsqueda y el tiempo total en milisegundos que ha tardado la búsqueda.

A continuación se comentan los resultados relevantes obtenidos:

BFS-G-3: Se aprecia que para una solución de 3 movimientos, expande 5 nodos y acaba encontrando la solución óptima de coste 3.

BFS-T-3: Al igual que en su versión en grafo, encuentra la solución óptima de coste 3. La diferencia principal es que al no mantener una lista de explorados, expande más nodos (6) y el tamaño de su frontera llega a ser mucho mayor que en su versión en grafo (10 vs 5).

DFS-G-3: La búsqueda en profundidad no garantiza que la solución sea la óptima y aquí se puede apreciar muy bien. La solución óptima que tiene coste 3, este algoritmo la ha encontrado en profundidad 59123. Como consecuencia, el número de nodos expandidos y guardados en la frontera, es muy superior al de los algoritmos de búsqueda en anchura anteriores.

DFS-T-3: La ejecución de la Búsqueda Primero en Profundidad en árbol ha tenido un coste excesivo en tiempo, bloqueando la ejecución de los algoritmos siguientes. Esto se debe a que no guarda una lista de explorados para poder descartar nodos que ya se hayan expandido y en consecuencia el algoritmo puede caer en bucles infinitos.
Se ha marcado este problema de tiempo con el identificador: (1).

DLS-9-3 y DLS-3-3: La Búsqueda en Profundidad Limitada no garantiza encontrar la solución óptima cuando se pone un límite superior a la profundidad de la solución. Es por ello que se observa que con un límite de 9, la solución encontrada es de coste 9. Sin embargo con un límite de 3, acaba encontrando la solución óptima en profundidad 3.
Destaca respecto a la Búsqueda en Profundidad en árbol, que este algoritmo sí es completo puesto que tiene un límite marcado que no puede superar y es imposible que entre en algún bucle infinito.
Por su implementación recursiva, la Búsqueda en Profundidad Limitada no necesita guardar una frontera para ir explorando los nodos, sino que la propia cola de llamadas sirve para explorar. Es por ello que el tamaño de la frontera siempre es nulo(0) para DLS.

IDS-3 : Se observa que la Búsqueda en Profundidad Recursiva tampoco guarda nodos en su frontera. Esto se debe a que internamente hace uso de la Búsqueda en Profundidad Limitada comentada en el párrafo anterior.
También se observa que encuentra la solución óptima a distancia 3.
Sin embargo, a diferencia del algoritmo DLS-3-3 que expande 4 nodos, este expande 9 nodos. Esto se debe a que IDS repite la expansión de los mismos nodos conforme incrementa el límite del árbol de búsqueda hasta encontrar la solución.

UCS-G-3 y UCS-T-3: La Búsqueda de Coste Uniforme tiene sentido utilizarla cuando el coste de cada acción es diferente. Sin embargo para este problema, el coste de las acciones es el mismo y por ello se observa que tanto la búsqueda en árbol como en grafo,

son similares a una Búsqueda Primero en Anchura. Por lo tanto también encuentran la solución óptima a profundidad 3.

Las siguientes ejecuciones de los algoritmos se realizan con la solución óptima a profundidad 9 y 30. Por tanto su comportamiento es idéntico al ya explicado para la solución a profundidad 3. Sin embargo, cabe destacar:

DFS-T-9 Y DFS-T-30: Este algoritmo sigue teniendo problemas para encontrar la solución en un tiempo razonable. Tanto a profundidad 9 como a 30 entra en bucles infinitos y nunca encuentra la solución.

IDS-30: La Búsqueda Iterativa en Profundidad, para una solución a profundidad 30 no es capaz de encontrarla en un tiempo razonable.

BFS-T-30: A diferencia de su versión en grafo para la profundidad de solución 30, en árbol, al no mantener una lista de nodos explorados, añade demasiados nodos a la frontera y esto acaba agotando la memoria.
Este problema de memoria se ha marcado con el identificador: (2).

UCS-T-30: Este algoritmo a profundidad 30 tiene el mismo problema que el ya mencionado en el párrafo anterior para el algoritmo bfs-t-30. Guarda tantos nodos en la frontera que al final consume toda la memoria disponible de mi máquina.

Definición del problema Misioneros y Caníbales

A continuación se presenta la modelización utilizada en el problema Misioneros y Caníbales.

- Representación del estado:
  Una tupla (m,c,b,m',c')
  m representa el número de misioneros en la orilla izquierda. [0,3]
  c representa el número de caníbales en la orilla izquierda [0,3]
  b representa la posición del bote. (1,0). (1) orilla izquierda. (0) orilla derecha.
  m' representa el número de misioneros en la orilla derecha. [0,3]
  c' representa el número de caníbales en la orilla derecha.[0,3]

- Estado inicial (3,3,1,0,0)

- Estado final (0,0,0,3,3)

- El modelo de estados seguido está basado en el realizado durante una clase de problemas de la asignatura cuando se modeló este problema.
  Básicamente las precondiciones se basan en evitar que en ningún momento existan más caníbales que misioneros en una orilla y que no se den situaciones incoherentes como mover un misionero de una orilla en la que no hay misioneros.
  Además la barca siempre tiene que estar en la orilla origen del desplazamiento de misioneros y caníbales.

Una vez implementado este modelo en java, se han ejecutado todos los algoritmos de búsqueda no informada para resolver este problema.
La siguiente imagen muestra las métricas y la traza obtenidas tras la ejecución de la Búsqueda Primero en Anchura en grafo:

```
<terminated> CanibalesPract1 [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (1 oct. 2020 12:10:
Misioneros y canibales BFS_grafo-->
pathCost : 11.0
nodesExpanded : 13
queueSize : 1
maxQueueSize: 3
Tiempo : 7mls

SOLUCION:
GOAL STATE
RIBERA-IZQ                --RIO-- BOTE M M M C C C  RIBERA-DCH
CAMINO ENCONTRADO
        INITIAL STATE   RIBERA-IZQ M M M C C C  BOTE --RIO--                 RIBERA-DCH
Action[name==Mover2Cd]  RIBERA-IZQ M M M     C       --RIO-- BOTE       C C  RIBERA-DCH
Action[name==Mover1Ci]  RIBERA-IZQ M M M   C C  BOTE --RIO--               C  RIBERA-DCH
Action[name==Mover2Cd]  RIBERA-IZQ M M M            --RIO-- BOTE     C C C  RIBERA-DCH
Action[name==Mover1Ci]  RIBERA-IZQ M M M     C  BOTE --RIO--             C C  RIBERA-DCH
Action[name==Mover2Md]  RIBERA-IZQ     M     C       --RIO-- BOTE   M M   C C  RIBERA-DCH
Action[name==Mover1M1Ci]   RIBERA-IZQ   M M   C C  BOTE --RIO--         M     C  RIBERA-DCH
Action[name==Mover2Md]  RIBERA-IZQ         C C       --RIO-- BOTE M M M     C  RIBERA-DCH
Action[name==Mover1Ci]  RIBERA-IZQ       C C C  BOTE --RIO--       M M M      RIBERA-DCH
Action[name==Mover2Cd]  RIBERA-IZQ           C       --RIO-- BOTE M M M   C C  RIBERA-DCH
Action[name==Mover1Ci]  RIBERA-IZQ         C C  BOTE --RIO--       M M M     C  RIBERA-DCH
Action[name==Mover2Cd]  RIBERA-IZQ                  --RIO-- BOTE M M M C C C  RIBERA-DCH
```

Definición del problema Fichas

A continuación se presenta la modelización utilizada en el problema Fichas para su posterior implementación en Java.

- Representación del estado:
  Un array {‘B’,’B’,’B’,’ ‘,’V’,’V’,’V’}
  ‘B’ representa una ficha de tipo B.
  ‘V’ representa una ficha de tipo V.
  ‘ ‘ representa un hueco(ausencia de ficha).

- Estado inicial {‘B’,’B’,’B’,’ ‘,’V’,’V’,’V’}

- Estado final {‘V’,’V’,’V’,’ ‘,’B’,’B’,’B’,’B’}

- Modelo de transiciones:

igap = Índice del hueco ‘ ‘ en el array estado

Acciones	Precondiciones	Resultado
GapRightJump0	igap < 6	pemutar(estado[igap], estado[igap + 1])
GapRightJump1	igap < 6 and igap > 0	pemutar(estado[igap], estado[igap + 1]) pemutar(estado[igap - 1], estado[igap - 2])
GapRightJump2	igap < 6 and igap > 1	pemutar(estado[igap], estado[igap + 1]) pemutar(estado[igap - 1], estado[igap - 2]) pemutar(estado[igap - 2], estado[igap - 3])
GapLeftJump0	igap > 0	pemutar(estado[igap], estado[igap - 1])
GapLeftJump1	igap >0 and igap < 6	pemutar(estado[igap], estado[igap - 1]) pemutar(estado[igap + 1], estado[igap + 2])
GapLeftJump2	igap >0 and igap < 5	pemutar(estado[igap], estado[igap - 1]) pemutar(estado[igap + 1], estado[igap + 2]) pemutar(estado[igap + 2], estado[igap + 3])

“GapRightJump0” = Cuando una ficha se desplaza al hueco moviéndose hacia la izquierda, el hueco se mueve a la derecha y además la ficha no salta ninguna ficha hacia la izquierda.

“GapRightJump1” = Cuando una ficha se desplaza al hueco moviéndose hacia la izquierda, el hueco se mueve a la derecha y además la ficha salta otra ficha hacia la izquierda.

“GapRightJump2” = Cuando una ficha se desplaza al hueco moviéndose hacia la izquierda, el hueco se mueve a la derecha y además la ficha salta otras dos fichas hacia la izquierda.

GapLeftJumpx son acciones equivalentes pero en sentido opuesto, es decir, el hueco se desplaza hacia la izquierda y por lo tanto la ficha contigua se desplaza hacia la derecha y además puede no saltar fichas(GapLeftJump0), saltar una ficha a la derecha(GapLeftJump1) o saltar dos fichas a la derecha (GapLeftJump2).

Cabe destacar que se han asociado distintos costes a cada acción para su posterior ejecución con todos los algoritmos no informados.
Esta diferencia de costes tiene sentido para el algoritmo Búsqueda de Coste Uniforme puesto que es el único cuyo criterio de elección del siguiente nodo a expandir se basa en el coste del nodo.

Se han asociado los siguientes costes a cada acción:

- La acción de mover una ficha al hueco = coste 1.0
- La acción de mover una ficha al hueco y que esta salte otra ficha = coste 2.0
- La ación de mover una ficha al gueco y que esta salte otras dos fichas = coste 3.0

La siguiente imagen muestra las métricas y la traza obtenidas de la ejecución de la Búsqueda de Coste Uniforme en grafo para este problema:

```
Fichas UCS_grafo-->
pathCost : 17.0
nodesExpanded : 139
queueSize : 0
maxQueueSize: 30
Tiempo : 1mls

SOLUCION:
GOAL STATE
|V|V|V| |B|B|B|
CAMINO ENCONTRADO
         INITIAL STATE   |B|B|B| |V|V|V|
Action[name==GapLeftJump1]   |B|B| |V|B|V|V|
Action[name==GapRightJump2]   |V|B|B| |B|V|V|
Action[name==GapRightJump0]   |V|B|B|B| |V|V|
Action[name==GapLeftJump2]   |V|B|B| |V|V|B|
Action[name==GapLeftJump2]   |V|B| |V|V|B|B|
Action[name==GapRightJump1]   |V|V|B| |V|B|B|
Action[name==GapLeftJump1]   |V|V| |V|B|B|B|
Action[name==GapRightJump0]   |V|V|V| |B|B|B|
```