



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO INGENIERIA EN SISTEMAS COMPUTACIONALES

Ejercicio de laboratorio 3

“DIFERENCIAS ENTRE BFS Y DFS”

Presentan

GOMEZ HERNANDEZ ALAN JAVIER
HERNÁNDEZ PÉREZ JUAN MANUEL
JIMÉNEZ CRUZ DANIEL
RIVAS CARRERA DIANA LAURA

Profesor

FLORIANO GARCÍA ANDRÉS

Septiembre 2023



INTRODUCCION

- Búsqueda en Profundidad (DFS):

La Búsqueda en Profundidad (Depth-First Search o DFS en inglés) es un algoritmo de búsqueda utilizado para explorar y recorrer estructuras de datos como grafos y árboles. La característica principal de DFS es que explora tan profundamente como sea posible a lo largo de una rama antes de retroceder. Esto significa que sigue explorando un camino hasta que llega al final, luego retrocede al nodo anterior y continúa explorando otro camino. A menudo se implementa utilizando una pila (o la recursión en el caso de la implementación recursiva).

DFS es útil para encontrar soluciones en problemas donde el objetivo es llegar lo más profundo posible antes de retroceder, como la resolución de laberintos, la generación de todas las permutaciones o combinaciones de un conjunto y la búsqueda en árboles de decisión.

- Búsqueda en Amplitud (BFS):

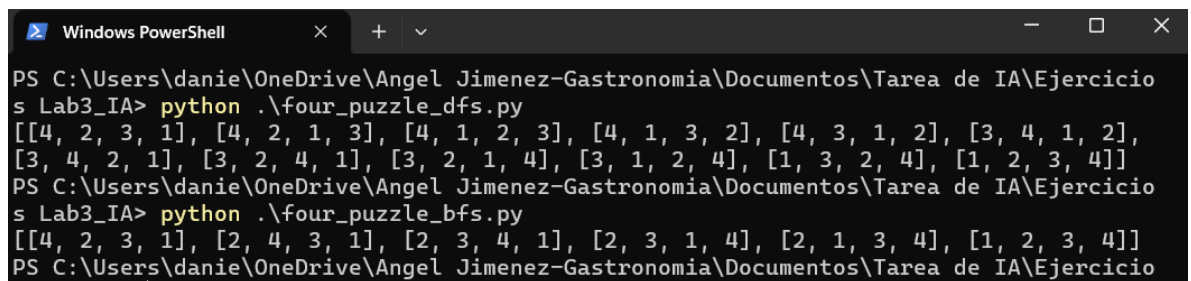
La Búsqueda en Amplitud (Breadth-First Search o BFS en inglés) es otro algoritmo de búsqueda utilizado para explorar estructuras de datos como grafos y árboles. A diferencia de DFS, BFS explora todos los nodos a un nivel dado antes de avanzar al siguiente nivel. Esto significa que primero se exploran todos los nodos a una profundidad de 1, luego a una profundidad de 2, y así sucesivamente. Se implementa utilizando una cola.

BFS es útil en problemas donde se busca la solución más corta o la solución a menor profundidad, ya que garantiza que se encuentre la solución óptima en términos de la cantidad de movimientos o pasos necesarios para llegar a ella. Se utiliza comúnmente en la búsqueda de rutas en mapas, la búsqueda en árboles de juegos y la resolución de rompecabezas como el Sudoku.

DESARROLLO

1. Para el ejemplo del 4-puzzle ejecuta el código proporcionado y determina si la solución en BFS fue mejor que la solución propuesta con DFS.

Podemos observar en la figura 1 que efectivamente la solución en BFS es mejor, principalmente porque los pasos para BFS fueron la mitad comparándolo con DFS. Esto se debe a que BFS evita bucles infinitos de manera natural, ya que explora todos los nodos a una profundidad antes de avanzar.



```
PS C:\Users\danie\OneDrive\Angel Jimenez-Gastronomia\Documentos\Tarea de IA\Ejercicio
s Lab3_IA> python .\four_puzzle_dfs.py
[[4, 2, 3, 1], [4, 2, 1, 3], [4, 1, 2, 3], [4, 1, 3, 2], [4, 3, 1, 2], [3, 4, 1, 2],
[3, 4, 2, 1], [3, 2, 4, 1], [3, 2, 1, 4], [3, 1, 2, 4], [1, 3, 2, 4], [1, 2, 3, 4]]
PS C:\Users\danie\OneDrive\Angel Jimenez-Gastronomia\Documentos\Tarea de IA\Ejercicio
s Lab3_IA> python .\four_puzzle_bfs.py
[[4, 2, 3, 1], [2, 4, 3, 1], [2, 3, 4, 1], [2, 3, 1, 4], [2, 1, 3, 4], [1, 2, 3, 4]]
PS C:\Users\danie\OneDrive\Angel Jimenez-Gastronomia\Documentos\Tarea de IA\Ejercicio
```

Figura 1. Ejecución de los códigos DFS y BFS.

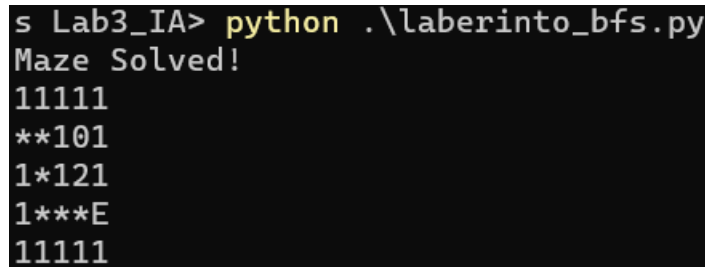
- a. Analiza la propuesta de backtracking proporcionada en este código ¿es eficiente en memoria?

Es eficiente debido a que se sigue el enfoque típico de BFS al utilizar una cola (en este caso, `nodos_frontera`) para mantener los nodos a explorar. Esto permite que el algoritmo explore primero todos los nodos a una profundidad dada antes de pasar a la siguiente profundidad. Dado que se utiliza una cola, los nodos se exploran en el orden en que se agregaron, lo que garantiza que se encuentre la solución óptima en términos de pasos.

Sin embargo, cabe mencionar que la eficiencia en memoria depende en gran medida del espacio de búsqueda real y de la estructura de datos utilizada para representarlo, así que, puede haber situaciones donde este código no sea el más eficiente.

2. Ejecuta el código que soluciona el laberinto con BFS.

En la figura 2, se muestra la ejecución del código.



```
s Lab3_IA> python .\laberinto_bfs.py
Maze Solved!
11111
**101
1*121
1***E
11111
```

Figura 2. Laberinto con BFS

- a. ¿Qué diferencias tiene con respecto al método DFS?
 - La primera diferencia es referente a la estructura de datos, BFS emplea una cola (deque) para mantener los nodos a explorar, mientras que DFS utiliza una pila (stack).
 - En cuanto a la exploración del laberinto, utilizando BFS se exploran primero todos los nodos a una profundidad dada antes de avanzar a la siguiente profundidad. Esto garantiza que encuentre la solución más corta en término de pasos. Por otro lado, DFS explora el camino lo más profundo posible antes de retroceder. No garantiza encontrar la solución más corta en términos de pasos, ya que podría encontrar una solución más profunda antes de encontrar una solución menos profunda.
- b. Al igual que en el ejercicio de Laboratorio 2, propón laberintos de mayor tamaño y dificultad y resuélvelos empleando los dos métodos; mide el tiempo de ejecución de ambos casos y determina qué método es el adecuado para los casos más complejos que hayas propuesto.

En la figura 3 y 4 se muestra la ejecución de ambos códigos empleando los dos métodos. Para ambos casos, el programa se ejecuto con el mismo tamaño (15) del laberinto. Podemos notar que en el caso del laberinto usando DFS (figura 3), el tiempo de ejecución es de 0.001699 segundos. Por otra parte, para el laberinto usando BFS (figura 4), el tiempo de ejecución fue de 0.002070 segundos.

En ambos casos, el tiempo no es muy distante uno del otro, sin embargo, se debe considerar lo siguiente.

BFS es adecuado cuando:

- Se necesita encontrar la solución más corta en términos de pasos o nivel de profundidad.
- Se tiene suficiente memoria disponible, ya que BFS tiende a requerir más memoria que DFS debido a la naturaleza de su cola. En casos extremadamente grandes, el uso de memoria podría ser una limitación.
- El laberinto no es demasiado profundo. En casos en los que el laberinto tiene una profundidad excesiva, BFS podría no ser práctico debido a la cantidad masiva de nodos que se deben almacenar en la cola.

DFS es adecuado cuando:

- La memoria es limitada, ya que DFS utiliza menos memoria en comparación con BFS. Si el espacio de memoria es una preocupación, DFS podría ser más viable.
- No es necesario encontrar la solución más corta. En algunos casos, encontrar la solución más corta no es una prioridad y se puede aceptar cualquier solución válida.
- El laberinto es muy profundo pero la solución se encuentra relativamente cerca del punto de inicio. DFS puede ser más eficiente en términos de tiempo cuando la solución está cerca de la raíz del árbol de búsqueda.

```
Enter the size of the maze (n m): 15 15
[+] Maze Solved!

# # # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
S . . # # # # # # # #
# . # # # # # # # # #
# # . . . . # # # # #
# # # # # E # # # # #
Tiempo de ejecución: 0.001699 segundos
```

Figura 3. Laberinto usando DFS

```

Enter the size of the maze (n m): 15 15
[+] Maze Solved!

# # # # # # # # # # # # # #
#   #       #       #   #   #
#       #       #       #   #
# #       # #       #       #
#       #   #       #       # # #
#       # #       #   # # # #
# #   #       #       #   # #
#   #       # # # # # # # #
# #       #       #   # # #
#       #       #       #   #
#   # #       #       # # #
# # # # # E S # # # # # # #
Execution Time: 0.002070 seconds

```

Figura 4. Laberinto usando BFS

CONCLUSIÓN

Después de analizar los diferentes problemas vistos en esta práctica, podemos concluir que las diferencias clave entre la Búsqueda en Profundidad (DFS) y la Búsqueda en Amplitud (BFS) se pueden resumir de la siguiente manera:

Estrategia de Exploración:

- DFS explora un camino lo más profundo posible antes de retroceder, mientras que BFS explora todos los nodos a una profundidad dada antes de avanzar a la siguiente profundidad.

Eficiencia en Memoria:

- DFS tiende a utilizar menos memoria que BFS, ya que utiliza una pila en lugar de una cola. Es adecuado cuando la memoria es limitada.
- BFS utiliza más memoria debido a su cola, lo que puede ser un inconveniente en problemas con un espacio de búsqueda masivo.

Optimalidad:

- DFS no garantiza encontrar la solución más corta en términos de pasos; puede encontrar una solución, pero no necesariamente la óptima.
- BFS garantiza encontrar la solución más corta en términos de pasos, lo que lo hace ideal para problemas de búsqueda de rutas más cortas.

Profundidad vs. Anchura:

- DFS es ideal para encontrar soluciones cercanas al punto de inicio o en grafos profundos.
- BFS es preferible cuando se busca la solución más corta, independientemente de la profundidad en la que se encuentre.

En general, la elección entre DFS y BFS depende de las características específicas del problema, los requisitos de tiempo y memoria, y el objetivo principal. Cada uno de estos algoritmos tiene sus propias fortalezas y debilidades, por lo que es importante seleccionar el que mejor se adapte a las necesidades de la tarea que se está abordando.

Referencias

Encora. (2020) *DFS vs BFS*. Recuperado el 23 de septiembre de 2023, de <https://www.encora.com/es/blog/dfs-vs-bfs>

Geeks for Geeks. (2020) *Difference between BFS and DFS*. Recuperado el 23 de septiembre de 2023, de <https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/>

Techie Delight (2016) *Búsqueda primero en profundidad (DFS) frente a búsqueda primero en amplitud (BFS)*. Recuperado el 23 de septiembre de 2023, de <https://www.techiedelight.com/es/depth-first-search-dfs-vs-breadth-first-search-bfs/>