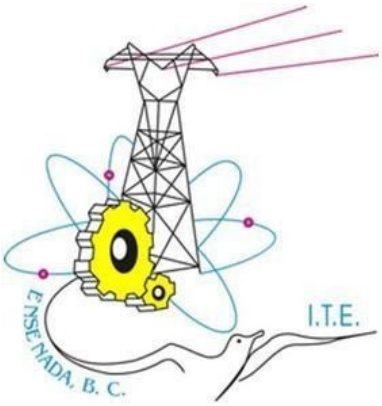
Tecnológico Nacional de México

Instituto Tecnológico de Ensenada

“Por la tecnología de hoy y del futuro”



Investigación:

Búsqueda por profundidad

Ingeniería en sistemas computacionales

Alumno: Jesus Luque Espinoza

Inteligencia artificial

Grupo 8SA

Profesor: Eddie Helbert Clemente Torres

Ensenada, B.C. 8 de octubre de 2020

Búsqueda por profundidad

En este proceso de búsqueda se genera sólo un sucesor del nodo en cada paso, es decir, cada vez que se obtiene un nuevo sucesor, se le aplica a este un nuevo operador y se obtiene un nuevo sucesor, y así sucesivamente.

En este tipo de búsqueda se avanza por una sola rama del árbol hasta que se encuentre una solución o hasta que se llegue a un callejón sin salida. En el caso de llegar a un callejón sin salida se retorna al nodo padre para iniciar una nueva búsqueda en los nodos hijos no visitados, en caso de que no existan nodos hijos no visitados, se retrocede aplicando el mismo proceso hasta llegar a la solución.

Las siguientes imágenes muestran el funcionamiento del algoritmo de búsqueda por profundidad.

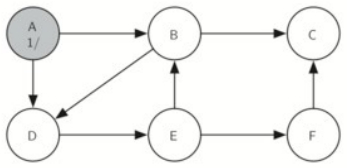


Figura 1

Figura 1. Inicia el algoritmo, visitando el vértice A. Marca este vértice de gris, indicando que fue visitado. Al tiempo de descubrimiento le asigna 1. Al tener 2 vértices adyacentes, se visita uno de ellos, en este caso B.

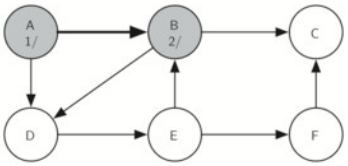


Figura 2

Figura 2. Se visita el vértice B y lo marca de gris. Al tiempo de descubrimiento le asigna 2. El vértice tiene como adyacentes otros dos vértices, el C y D. Se visita uno de ellos, en este caso C.

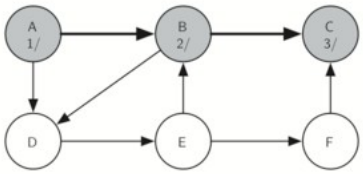


Figura 3

Figura 3. Se visita el vértice C y lo marca de gris. Al tiempo de descubrimiento le asigna 3. El vértice no tiene adyacentes, por lo tanto es el nodo final de una rama del árbol.

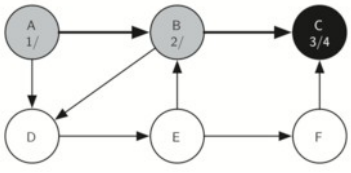


Figura 4

Figura 4. Al tener el vértice C como nodo final de una rama, se marca de gris. Al tiempo de descubrimiento se le asigna un 4. Por último se vuelve al vértice anterior, el vértice B.

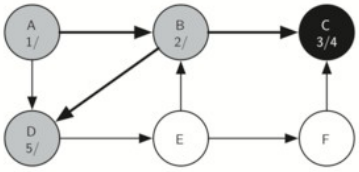


Figura 5

Figura 5. El vértice B solo tiene de adyacente no visitado al vértice D. Por lo tanto se visita el vértice D y ser marca de gris. Al tiempo de descubrimiento le asigna 5. El vértice tiene como adyacente el vértice E, por lo tanto se visita el vértice E.

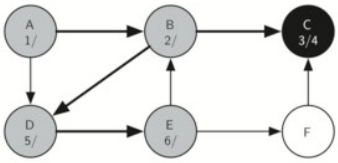


Figura 6

Figura 6. Se visita el vértice E y se marca de gris. Al tiempo de descubrimiento se le asigna 6. Los vértices adyacentes son B y F, pero en este caso B ya ha sido visitado, por lo tanto se visita el vértice F.

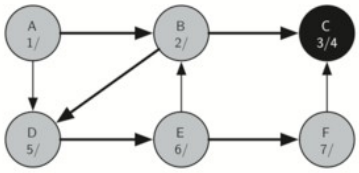


Figura 7

Figura 7. Se visita el vértice F y se marca de gris. Al tiempo de descubrimiento se le asigna 7. El único vértice adyacente es el vértice C, pero este vértice ya fue visitado y marcado en negro. Por lo tanto F es el fin de otra rama del arból.

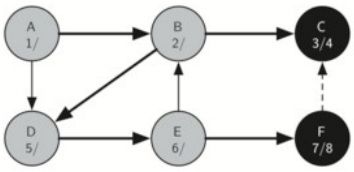


Figura 8

Figura 8. Al ser el vértice 7 otro fin de otra rama, se marca de negro y asigna al tiempo de descubrimiento 8. A partir de aquí el algoritmo va regresando por el camino que le llevo al vértice F, siendo F ← E ← D ← B ← A . Al terminar cada que se vuelve visitar los vértices grises, como no tienen ningún vértice adyacente sin visitar, se marcan en negro. Quedando el grafo como en la figura 9.

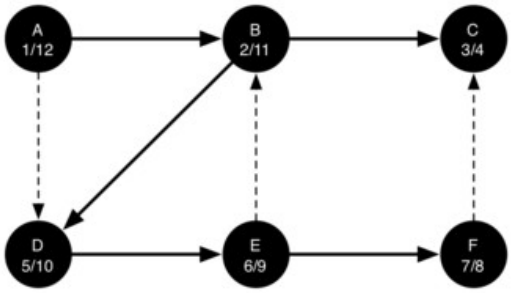


Figura 9

Comparación con el algoritmo de búsqueda por anchura

En el algorítmo de búsqueda en profundidad, la exploración del arbol funciona de tal forma que sólo necesita almacenar el recorrido hasta la solución o un punto sin sucesor, y en caso de que sea este último vuelve al nodo raíz, en cambio en el algoritmo de búsqueda de profundidad la exploración se realiza por niveles, no se revisa un nivel más profundo a menos que todos los nodos del nivel actual hayan sido visitados, de esta forma el arbol se va leyendo de manera uniforme hasta que se llegue a la solución.

Ejemplo de implementación de búsqueda por anchura y búsqueda por profundidad

Búsqueda por profundidad. El rompecabezas de la gira del caballo se juega en un tablero de ajedrez con una sola pieza de ajedrez, el caballo. El objetivo del rompecabezas es encontrar una secuencia de movimientos que permitan al caballo visitar cada cuadrado del tablero exactamente una vez. Para resolver el problema se utilizan dos pasos principales:

- Representar como un grafo los movimientos legales de un caballo en un tablero de ajedrez.
- Usar un algoritmo de grafos para encontrar una ruta de longitud $filas \times columnas - 1$ donde cada vértice del grafo se visite exactamente una vez.

La gira del caballo es un caso especial de una búsqueda en profundidad donde el objetivo es crear el árbol de búsqueda en profundidad más profundo, sin ramas.

Código

```
from pythoned.grafos import Grafo, Vertice
def giraCaballo(n,ruta,u,limite):
        u.asignarColor('gris')
        ruta.append(u)
        if n < limite:
            listaVecinos = list(u.obtenerConexiones())
            i = 0
            hecho = False
            while i < len(listaVecinos) and not hecho:
                if listaVecinos[i].obtenerColor() == 'blanco':
```

```
                    hecho = giraCaballo(n+1, ruta, listaVecinos[i], limite)
                i = i + 1
            if not hecho:  # prepararse para retroceder
                ruta.pop()
                u.asignarColor('blanco')
        else:
            hecho = True
        return hecho
```

Descripción del código

La función giraCaballo recibe cuatro parámetros:

- n, profundidad actual en el árbol de búsqueda.
- ruta, lista de vertices visitadas hasta el momento.
- u, vértice en el grafo que se desea explorar.
- limite, cantidad de nodos totales en la ruta.

Para saber cuales son los vecinos visitados y no visitados se indica con un color, siendo blanco un vértice no visitado y gris si visitado. Para conocer las conexiones que tiene un vértice se obtienen por el método obtenerConexiones y se guardan en una lista.

La función es recursiva, el caso base es si la cantidad de vértices de la ruta es 64 (cantidad de casillas totales del tablero de ajedrez), la función devuelve un True, indicando que se ha encontrado una gira exitosa. En cambio si la ruta aún no se completa, se sigue avanzando por un nuevo vértice. La forma de llegar a un callejón sin salida es si el caballo no tiene ninguna casilla no visitada para moverse y aún no se han visitado 64 casillas, es decir, si un vertice no tiene vecinos sin visitar y la longitud de la ruta no es 64. Cuando se llegue a un callejón sin salida se retrocede, la función devuelve un false, esto indica que se debe búscar por un vértice vecino no visitado.

Dentro del ciclo while al recibir un false y si la lista de vecinos no se ha recorrido en totalidad, se recorren los vecinos. Si no existen más vecinos no visitados y se ha recibido un false, se retrocede un vértice más quitandole el color gris al vértice actual, para que pueda ser visitado en una búsqueda por otro camino.

Búsqueda por anchura. El código de implementación de la búsqueda por anchura es el siguiente:

```
from pythoned.grafos import Grafo, Vertice
from pythoned.basicas import Cola

def bea(g,inicio):
  inicio.asignarDistancia(0)
  inicio.asignarPredecesor(None)
  colaVertices = Cola()
  colaVertices.agregar(inicio)
  while (colaVertices.tamano() > 0):
    verticeActual = colaVertices.avanzar()
    for vecino in verticeActual.obtenerConexiones():
      if (vecino.obtenerColor() == 'blanco'):
        vecino.asignarColor('gris')
        vecino.asignarDistancia(verticeActual.obtenerDistancia() + 1)
        vecino.asignarPredecesor(verticeActual)
        colaVertices.agregar(vecino)
    verticeActual.asignarColor('negro')
```

La implementación del algoritmo de búsqueda en anchura utiliza la representación de un grafo mediante una lista de adyacencia y una cola para decidir que vértice se debe explorar.

La función bea comienza en el vértice s. De igual forma que en la implementación de búsqueda por profundidad, los vértices no visitados tienen el color blanco, los si visitados pero aún no se evalúa su ruta más corta tienen el color gris y por último los si visitados con su ruta más corta tienen el color negro. Como el vértice de inicio es el primero en visitarse, se le pinta de gris, su distancia es 0 y su predecesor es None. Se agrega inicio a la cola, lo siguiente es avanzar uno en profundidad en la parte delantera de la cola. De esta forma se visita un vértice hijo y se procede a visitar a los vecinos

adyacentes, es decir del mismo nivel. Si el vértice vecino es color blánco, se asigna gris, se le asigna la distancia de el vertice actual + 1 y el predecesor se le asigna como el vértice actual. Por último se agrega el vértice vecino a agrega la cola, esto para dar una segunda exploración para conocer la ruta más corta, pero no antes de visitar todo el nivel actual.

Referencias

Miller, B et al., (2011). *Problem Solving with Algorithms and Data Structures using Python.* OR, United States: Franklin, Beedle & Associates Inc.

Molina, J, Torres, C, & Restrepo, C, (2008). *TÉCNICAS DE INTELIGENCIA ARTIFICIAL PARA LA SOLUCIÓN DE LABERINTOS DE ESTRUCTURA DESCONOCIDA.*. Scientia Et Technica, XIV(39),135-140. [fecha de Consulta 8 de Octubre de 2020]. ISSN: 0122-1701. Disponible en: https://www.redalyc.org/articulo.oa?id=849/84920503025