Posgrado de Ingeniería de Sistemas

Búsqueda en anchura y profundidad

Alberto Benavides
2 de abril de 2020

1. Introducción

En grafos, se denomina *búsqueda* a la operación de visitar nodos del grafo con la finalidad de encontrar un nodo específico. En el peor de los casos, se deben recorrer todos los nodos del grafo con una complejidad $\mathcal{O}(n)$ con la posibilidad de que el nodo de interés no sea encontrado. Estos algoritmos se utlizan cuando se desconoce la ubicación del nodo, en contraste con los algoritmos para colas prioritarias, por ejemplo , en que los montículos de Fibonacci permiten el acceso al menor elemento en tiempo $\mathcal{O}(n)$.

Para esta práctica se desarrollaron dos algoritmos de búsqueda, uno llamado *búsqueda en profundidad* y el otro *búsqueda en anchura*. Estos algoritmos fueron desarrollados en `javascript` con la ayuda de la librería `greuler` [1] para mostrar una representación visual de los nodos.

2. Definición de los algoritmos

Ambos algoritmos comparten el principio de recorrer un grafo $G = (V, E)$ a partir de un vértice inicial $v \in V$, pero la manera de proceder es distinta en cada uno. Las búsquedas en anchura visitan nodo a nodo a través de las jerarquías inmediatas inferiores. Es decir, que tras partir del nodo inicial n, se busca primero en todos sus nodos vecinos $\{u, v\} in V_n$, posteriormente en todos los vecinos de los vecinos, y así sucesivamente.

El algoritmo de búsqueda en profundidad explora los nodos extendiéndose en en camino de vecindades hasta agotarlo y luego regresa hasta algún camino no visitado para explorarlo a continuación, y se prosigue de este modo hasta que encuentra el nodo buscado o se visitan todos los nodos.

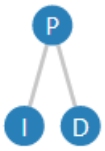

Figura 3.1: Representación de un árbol binario con tres nodos. La raíz o padre marcado por una P, el hijo izquierdo por una I y el derecho por una D.

3. Árbol binario

Con finalidades ilustrativas, se eligió como grafo una estructura de datos de *árbol binario*. Estos árboles tienen la propiedad de que cuentan con un *nodo inicial o raíz* a partir del cual se deben establecer hasta un máximo de dos vecindades. Por simplicidad, se suele denominar *hijos* a los dos nodos que acompañan a otro nodo, a su vez nombrado *padre*. Además, por ser dos los hijos, se les refiere como *hijo izquierdo* e *hijo derecho*. Esto puede verse en la imagen 3.1.

La operación de inserción para un árbol binario se puede realizar de diversas maneras. En esta práctica se ha optado por utilizar un vector de nodos N identificados cada uno por un índice $i \in 0, 1, 2, ...$ Así, el nodo con índice $i = 0$ es el nodo raíz y para todos los demás nodos se asigna un entero consecutivo conforme al orden de la inserción. Los nodos con $i > 0$ son por defecto hijos de algún otro nodo con índice j dado por $j = \lfloor (i-1)/2 \rfloor$ en el entendido de que si $i \mod 2 = 0$ se trata del hijo derecho, mientras que en otro caso, es el hijo izquierdo.

4. Implementaciones

La búsqueda de un valor k en anchura para un árbol binario definido de esta forma consiste en recorrer el vector de nodos como se muestra en el algoritmo 1.

Algoritmo 1: Algoritmo de búsqueda en anchura.

```
para i ∈ N hacer
    si n_i = v entonces
        terminar;
    fin
fin
devolver i
```

En el caso del algoritmo de búsqueda en profunidad, se optó por una función recursiva que toma como parámetros el valor k buscado y un nodo m como se constata en el algoritmo 2.

Algoritmo 2: Algoritmo de búsqueda en profundidad.

Función `Profundidad`(k, m)**:**
  si $k = m$ **entonces**
    devolver $\top$;
  en otro caso
    $e \leftarrow \bot$;
    si $\exists m_{\textit{hijo izquierdo}}$ **entonces**
      `Profundidad`$(k, m_{\text{hijo izquierdo}})$;
    fin
    si $\exists m_{\textit{hijo derecho}}$ **entonces**
      `Profundidad`$(k, m_{\text{hijo derecho}})$;
    fin
    devolver $\bot$;
  fin
  devolver $\bot$;
Fin Profundidad
`Produndidad`(k, n);

Estas implementaciones pueden consultarse en `https://jbenavidesv87.github.io/algoritmos/search.html`.

Referencias

[1] Mauricio Poppe. Greuler. Graph Theory Visualization. `https://mauriciopoppe.github.io/greuler/`. [Accedido 24/marzo/2020].