Pedro Escoboza
A01251531

11 de noviembre de 2020
TC1031.501

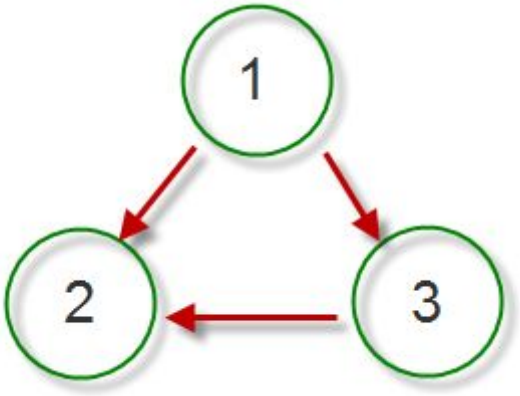
Actividad Integral de Grafos

Un grafo es una estructura de datos no lineal que está conformada por nodos interconectados entre ellos por aristas. Este tipo de organización y cálculo de relaciones entre los nodos, tales como caminos entre ellos y ordenaciones topológicas por distintos tipos de recorridos.

Existen distintos tipos de implementaciones para una estructura programática de grafo. Dos altamente populares son la lista la matriz de adyacencia y la lista de adyacencia. Ambas metodologías de representación permiten la instanciación de grafos dirigidos y bidirigidos. En los primeros, la relación de arista entre dos nodos indica una direccionalidad, mientras que en el segundo, la relación de arista permite recorridos en ambas direcciones.

En la solución al presente reto se optó por la segunda implementación que funciona de la siguiente manera: Se tiene una lista de elementos donde cada uno guarda una referencia a un grupo de nodos de los cuales este se considera el padre y tiene aristas en dirección a ellos. La estructura se representa a continuación:

```
1 - { 2, 3 }
|
2 - { }
|
3 - { 2 }
```



Funcionalidad común de un grafo

Búsqueda de amplitud
Complejidad computacional: O(n)

Recorre el grafo desde una posición inicial y visita cada nodo encontrado dando prioridad a visitar todos los nodos hijos del nodos presente antes de pasar a los nietos.

En su implementación más común y seleccionada para la presente solución, se utiliza una cola como estructura de datos asistente. Se inserta en nodo raiz a la cola marca como visitado. Después, mientra la cola no este vacía, se insertan los hijos del primero nodo extraído del frente y se actualiza un iterador para el siguiente nodo de la cola. Solo se

insertan nodos que no han sido marcados como visitados y se marcan al ser insertados en la cola de visita.

Búsqueda de profundidad
Complejidad computacional: `O(n)`

Recorre el grafo desde una posición inicial y visita cada nodo, con prioridad en visitar los nodos en el siguiente nivel de profundidad antes que los vecinos inmediatos del nodo actual.
Comenzando en el nodo raíz, se marca el presente nodos como visitado y se llama recursivamente la operación de búsqueda de profundidad en cada uno de sus hijos. Esto mientras no se encuentre un nodo ya visitado, caso en cual se regresa la pila de ejecución un nivel arriba y se busca otro nodo hijo por el cual visitar a profundidad.

Agregar nodo
Complejidad computacional: `O(1)`

Se inserta un nuevo nodo en el grafo. En el caso de la lista de adyacencia, esto se hace insertando en nodo en la lista principal.

Agregar arista
Complejidad computacional: `O(1)` (utilizando un hashmap como tabla de nodos)

Se reciben los valores de dos nodos y se representa la relación de conexión entre ellos. En el caso de la lista de adyacencia, se insertan a la lista principal y se agrega un apuntador al nodo hijo en la lista de adyacencias del nodo padre.

Determinar existencia de nodo
Complejidad computacional: `O(1)` (utilizando un hashmap como tabla de nodos)

Determina si un nodo existe como parte del grafo. Para la lista de adyacencia, se realiza una búsqueda por valor en lista principal.

Identificación de ciclos
Complejidad computacional: `O(n)`

Determina la presencia de ciclos en un grafo al realizar un recorrido, ya sea de profundidad o amplitud. En caso de detectar un intento de visita a un nodo previamente marcado, se determina la existencia de un ciclo en el camino recorrido. Esta misma operación se puede utilizar para determinar si un camino de recorrido es un árbol.

Identificar si es un grafo bipartito
Complejidad computacional: `O(n)`

Pedro Escoboza
A01251531

Un grafo bipartito es un tipo de grafo que tiene la característica de que los nodos se pueden clasificar en dos conjuntos en los cuales sus miembros no tienen conexiones entre ellos mismos. Suele hacerse la analogía de que los nodos tienen dos colores opuestos.

Se recorre el grafo desde uno nodo inicial por el método designado. Al visitar cada nodo, se marcan con una de dos denominaciones o colores en orden alternado. Es decir, si un nodo es rojo, el siguiente nodo será marcado azul. En el caso de encontrar un nodo previamente marcado por el recorrido, se comparan los colores del nodo presente y anterior. Si ambos son del mismo color, significa que no es un grafo bipartito. En caso de terminar el recorrido sin producirse el caso anterior, el recorrido corresponderá a un grafo bipartito.

Ordenamiento topológico
Complejidad computacional: `O(n)`

Es un método de ordenamiento de los valores almacenados en los nodos que corresponde al resultado de un recorrido. Se recorre la estructura desde un nodo inicial, por profundidad o amplitud, y se insertan todos los nodos, en el orden en que son visitados, en una estructura de pila, que permita su extracción ordenada en el orden inverso al recorrido.

Solución

En el presente reto se trabajó con datos leídos de un documento de entrada con registros de intentos de acceso desde distintos puertos y direcciones ip. Para determinar programáticamente la dirección con mayor número de intentos de acceso se optó por utilizar un grafo implementado a través de una lista de adyacencias. Esta a su vez fue representada utilizando un hashmap de nodos como llaves y sets de punteros a las llaves como valores.

Se creó un sistema de herencia de clases con una base NetNode, que representa el nodo de una red. Las clases que heredan de esta base, son Listener y Client. Listener representa una dirección de puerto de acceso en el servidor, mientras que Client identifica a una dirección ip que intenta acceder. De esta manera, se obtiene de cada uno de los registros del documento de entrada, una arista del grafo.

Al utilizar a NetNode como argumento de templete para la clase Graph, y sobrecargar structs de ordenamiento e igualdad, fue posible utilizar punteros polimórficos como llaves en el hashmap de la lista de adyacencias, lo que permite generalizar el comportamiento de todos los tipos de nodo y optimizar la complejidad computacional del acceso directo a un nodo a una valoración Big O de O(1).

En el flujo de la solución, se lee el archivo y se insertan todos las aristas en el grafo, incrementando el número de de cada nodo según sea necesario. Al finalizar, se realiza un recorrido completo por la tabla de nodos del grafo para determinar el cliente con mayor número de intentos de acceso. Esto resulta en una complejidad computacional total de O(n).

Pedro Escoboza
A01251531
11 de noviembre de 2020
TC1031.501

Reflexión

Los grafos son estructuras de datos que por sus características estructurales se prestan para distintas aplicaciones. Entre ellas están la solución a problemas de navegación y transporte, sin embargo, también son preferidos en el manejo de relaciones abstractas entre conjuntos de datos, como por ejemplo para mapear relaciones entre usuarios en redes sociales o relaciones estructurales entre directorios de un ordenador. En el presente trabajo se hizo uso de grafos para representar nodos de una red y mapear las rutas y frecuencia de acceso a distintos puertos de un servidor. La estructura es apropiada para este caso, ya que permite modelar de manera compleja y dinámica un conjunto de datos interrelacionados de una manera en la que no es posible en otras estructuras, conservando la complejidad de interconexión entre todos los nodos. Esta y otras aplicaciones son evidencia de la importancia del entendimiento y manejo adecuado de estructuras de datos para resolución de problemas y generación de valor a través de las disciplinas ingenieriles.

Referencias

Anggoro, W. (2018). C++ Data Structures and Algorithms : Learn How to Write Efficient

Carey, J., Doshi, S. & Rajan, P. (2019). C++ data structures and algorithm design principles : Leverage the Power of Modern C++ to Build Robust and Scalable Applications. Birmingham, UK: Packt Publishing.

Code to Build Scalable and Robust Applications in C++. Packt Publishing.

Dale, N. (2003). C++ plus data structures. Boston, MA: Jones and Bartlett.

Drozdek, A. (2001). Data structures and algorithms in C. Pacific Grove, CA: Brooks/Cole.