Informe algoritmos DFS y BFS

Para hacer el algoritmo de DFS lo primero que hicimos fue buscar información sobre cómo funcionaba el algoritmo y recopilar las implementaciones de las estructuras necesarias para ponerlo en práctica. Las estructuras que usamos son las de Grafos y la de Conjuntos que es en donde se guardan los vértices de un grafo.

Para realizar el algoritmo BFS nuevamente buscamos información y notamos el uso de las estructuras para DFS y además una estructura de Cola.

Al ver la información recopilada nos dimos cuenta que la implementación que teníamos no tenía un atributo estado en los nodos del grafo, el cual es necesario para poder recorrerlo de forma vertical. También es necesario para recorrerlo de forma horizontal, pero en esta última utiliza solo 2 estados, cuando en la vertical utiliza 3. Al principio pusimos validaciones de que el estado sólo pueda tener 3 string determinados, pero al darnos cuenta de esto, decidimos que estado pueda tener cualquier string que se desee utilizar, para poder poner BFS y DFS en el mismo directorio y que utilicen la misma implementación de grafos.

El funcionamiento es el siguiente:

La función DFS recibe:

- GrafoTDA grafo : El grafo que se recorrerá
- NodoGrafo origen: El nodo desde el cual empieza a recorrer
- int Tiempo: Una variable que siempre será 0, la cual se debe recibir por parámetro ya que no podriamos declararla en la función por utilizar una estructura recursiva.
- int[] arrayTiempos: Un array de enteros en donde se guardará el tiempo que se tardó en llegar a cada nodo
- int[] nodosPadres: Un array de enteros en donde se guarda el numero de los nodos padre de cada nodo
- ArrayList<Integer> recorrido: Un objeto array de enteros en donde se guarda el recorrido que es la solución final

Pseudocódigo

```
DFS(){
  //Sumo un tiempo por la ejecución para añadirlo al array tiempos
  tiempo = tiempo+1;

  //Marco el origen como "gris"
  NodoGrafo nodo = grafo.RetornarNodo(origen.getNodo());
  grafo.CambiarEstado("gris",nodo);
  /* Esto hace RetornarNodo
    public NodoGrafo RetornarNodo(int v){
        NodoGrafo nodo = Vert2Nodo(v);   //Vert2Nodo busca el nodo que tiene v como valor
        return nodo;
  */

    // Obtener los nodos adyacentes
    ConjuntoTDANodo nodosAdyacentes = grafo.NodosAdyacentes(arista,"blanco");

  //Recorro por cada nodo adyacente
  while (!nodosAdyacentes.ConjuntoVacio()){
          //Elijo un nodo y lo saco de nodosAdyacentes
       NodoGrafo nodoAdyacente = nodosAdyacentes.Elegir();
       nodosAdyacentes.Sacar(nodoAdyacente);
          //Si el nodoAdyacente tiene como estado null o "blanco"
       if (nodoAdyacente.getEstado() == "blanco"|| nodoAdyacente.getEstado() == null){

          //Agrego el nodo padre de ese nodo al array nodosPadres
          nodosPadres[nodoAdyacente.getNodo()] = origen.getNodo();

             //Ejecuto de nuevo DFS con el nodo adyacente que saqué en la iteración
              DFS(grafo, nodoAdyacente, tiempo, arrayTiempos, nodosPadres,recorrido);
             //Añado el nodo al recorrido
             recorrido.add(nodoAdyacente.getNodo());
       }

    }
//Cambio el estado de el nodo en el cual se encuentra en esta ejecución recursiva, esto
hace que vaya desde el final hacia al principio
origen.setEstado("negro");

//Añado el tiempo a el arrayTiempos asignandolo al nodo en donde me encuentro
arrayTiempos[origen.getNodo()] = tiempo;

}
```

Para el algoritmo BFS:

La función recibe:

- GrafoTDA grafo : El grafo que se recorrerá
- NodoGrafo origen: El nodo desde el cual empieza a recorrer
- int Tiempo: Una variable que siempre será 0, la cual se debe recibir por parametro ya que no podriamos declararla en la función por utilizar una estructura recursiva.
- int[] arrayTiempos: Un array de enteros en donde se guardará el tiempo que se tardó en llegar a cada nodo
- int[] nodosPadres: Un array de enteros en donde se guarda el numero de los nodos padre de cada nodo
- ArrayList<Integer> recorrido: Un objeto array de enteros en donde se guarda el recorrido que es la solución final
- Cola cola: recibe una cola para poder acolar los nodos adyacentes, para luego ir desacolando y procesando los siguientes nodos adyacentes.

Pseudocódigo

```
//Sumo un tiempo por la ejecución
      tiempo = tiempo+1;
      arrayTiempos[origen.getNodo()] = tiempo;
      //Marco el origen como "visitado"
      NodoGrafo nodo = grafo.RetornarNodo(origen.getNodo());
      grafo.CambiarEstado("visitado",nodo);

      //obtener nodos adyacentes
      NodoArista arista = nodo.getArista();
      ConjuntoTDANodo nodosAdyacentes = grafo.NodosAdyacentes(arista,"no visitado");
      //iteramos por cada nodo adyacente
      while(!nodosAdyacentes.ConjuntoVacio()){
         //Elijo un nodo y lo saco de nodosAdyacentes
         NodoGrafo nodoAdyacente = nodosAdyacentes.Elegir();
         nodosAdyacentes.Sacar(nodoAdyacente);
         //si el estado del nodo es no visitado o null
         if (nodoAdyacente.getEstado() == "no visitado" || nodoAdyacente.getEstado() ==
null){

            //Agrego el nodo padre de ese nodo al array nodosPadres
            nodosPadres[nodoAdyacente.getNodo()] = origen.getNodo();
```

```
            //Acolo el nodo para luego ir desacolando por nivel
            cola.Acolar(nodoAdyacente)
            //Añado el nodo al recorrido
            recorrido.add(nodoAdyacente.getNodo());
        }
    }
    //mientras que la cola no este vacía y nos queden elementos para procesar
    //obtenemos uno de esos elementos y desacolamos
    if (!cola.ColaVacia()){
        NodoGrafo siguienteNodo = cola.Primero();
        cola.Desacolar();
        //llamada recursiva con el siguiente nodo
        BFS(grafo, siguienteNodo, tiempo, arrayTiempos, nodosPadres,cola, recorrido);

    }

}
```

Para ejecutar cualquiera de estos dos algoritmos solo hay que cambiar
String estado = "blanco";
Si queremos DFS, Cambiar estado a "blanco" .
Para BFS cambiar estado a "no visitado".

Integrantes: Piehl Luciano, Romano lucas