Programación 3 - 2023

Grafos

Recorridos de grafos

Recorrido en profundidad: DFS (Depth First Search)

```
public ListaGenerica<T> dfs(Grafo<T> grafo){
    ListaGenerica<T> resultado = new ListaGenericaEnlazada<T>();
    if(!grafo.esVacio()) {
        boolean[] visitados = new boolean[grafo.listaDeVertices().tamanio()];
        ListaGenerica<Vertice<T>> listaDeVertices = grafo.listaDeVertices();
        listaDeVertices.comenzar();
        while(!listaDeVertices.fin()) {
            Vertice<T> vInicial = listaDeVertices.proximo();
            if(!visitados[vInicial.posicion()]) {
                dfs(vInicial,visitados,resultado,grafo);
            }
        }
    }
    return resultado;
}

private void dfs(Vertice<T> vActual, boolean[] visitados, ListaGenerica<T> resultado, Grafo<T> grafo) {
    visitados[vActual.posicion()] = true;
    resultado.agregarFinal(vActual.dato());
    ListaGenerica<Arista<T>> listaDeAdyacentes = grafo.listaDeAdyacentes(vActual);
    listaDeAdyacentes.comenzar();
    while(!listaDeAdyacentes.fin()) {
        Vertice<T> vSiguiente = listaDeAdyacentes.proximo().verticeDestino();
        if(!visitados[vSiguiente.posicion()]) {
            dfs(vSiguiente, visitados, resultado, grafo);
        }
    }
}
```

Recorrido en amplitud: BFS (Breath First Search)

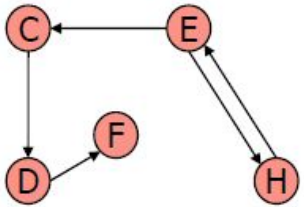
```java
public ListaGenerica<T> bfs(Grafo<T> grafo){
    ListaGenerica<T> resultado = new ListaGenericaEnlazada<T>();
    if(!grafo.esVacio()) {
        boolean[] visitados = new boolean[grafo.listaDeVertices().tamanio()];
        ListaGenerica<Vertice<T>> listaDeVertices = grafo.listaDeVertices();
        listaDeVertices.comenzar();
        while(!listaDeVertices.fin()) {
            Vertice<T> vInicial = listaDeVertices.proximo();
            if(!visitados[vInicial.posicion()]) {
                bfs(vInicial,visitados,resultado,grafo);
            }
        }
    }
    return resultado;
}

private void bfs(Vertice<T> vInicial, boolean[] visitados, ListaGenerica<T> resultado, Grafo<T> grafo) {
    ColaGenerica<Vertice<T>> cola = new ColaGenerica<Vertice<T>>();
    cola.encolar(vInicial);
    visitados[vInicial.posicion()] = true;
    while(!cola.esVacia()) {
        Vertice<T> vActual = cola.desencolar();
        resultado.agregarFinal(vActual.dato());
        ListaGenerica<Arista<T>> listaDeAdyacentes = grafo.listaDeAdyacentes(vActual);
        listaDeAdyacentes.comenzar();
        while(!listaDeAdyacentes.fin()) {
            Vertice<T> vSiguiente = listaDeAdyacentes.proximo().verticeDestino();
            if(!visitados[vSiguiente.posicion()]) {
                visitados[vSiguiente.posicion()] = true;
                cola.encolar(vSiguiente);
            }
        }
    }
}
```

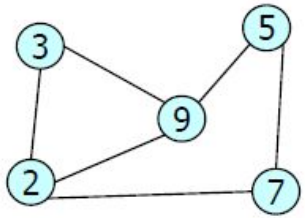
Caminos

Camino: Secuencia de vértices desde un origen a destino.

Longitud de un camino: número de aristas del camino.

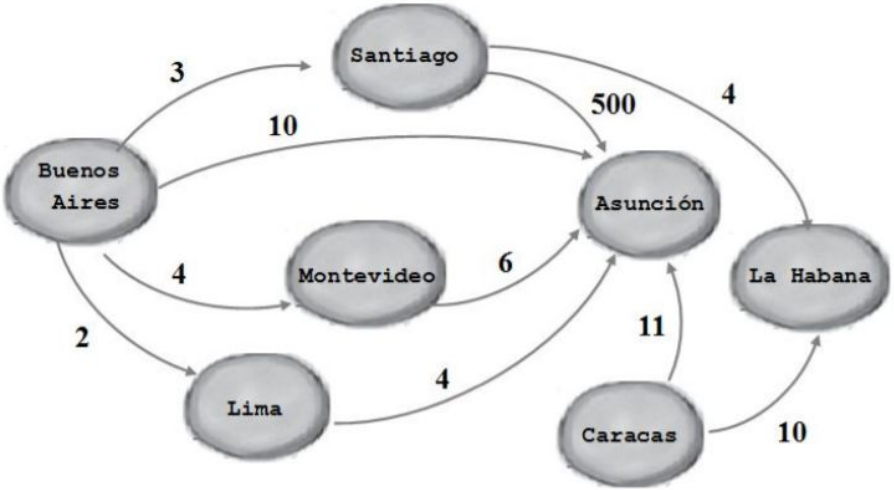
Ciclo: camino donde origen = destino





Ejemplo con caminos

Dado un Grafo orientado y valorado positivamente, implemente un método que retorne una lista el camino desde un origen a destino. En caso de que haya más de un camino, devolver el que pase por menos ciudades.



Ejemplo con caminos

```
public ListaGenerica<String> devolverCamino(Grafo<String> mapaCiudades, String origen, String destino) {
    Vertice<String> vInicial = null;
    Vertice<String> vFinal = null;
    boolean[] visitados = new boolean[mapaCiudades.listaDeVertices().tamanio()];
    ListaGenerica<String> camino = new ListaGenericaEnlazada<String>();
    if (mapaCiudades != null && !mapaCiudades.esVacio()) {
        ListaGenerica<Vertice<String>> vertices = mapaCiudades.listaDeVertices();
        vertices.comenzar();
        while (!vertices.fin()) {
            Vertice<String> vAux = vertices.proximo();
            if (vAux.dato().equals(origen)) {
                vInicial = vAux;
            }
            if (vAux.dato().equals(destino)) {
                vFinal = vAux;
            }
        }
        ListaGenerica<String> caminoActual = new ListaGenericaEnlazada<String>();
        if (vInicial != null && vFinal != null)
            dfsCamino(mapaCiudades, vInicial, vFinal, caminoActual, camino, visitados);
    }
    return camino;
}
```

Ejemplo con caminos

```
private void dfsCamino(Grafo<String> mapaCiudades, Vertice<String> vIni, Vertice<String> vFin,
        ListaGenerica<String> caminoActual, ListaGenerica<String> camino, boolean[] visitados) {
    caminoActual.agregarFinal(vIni.dato());
    visitados[vIni.posicion()] = true;
    if (vIni == vFin) {
        if (camino.esVacia() || camino.tamanio() > caminoActual.tamanio())
            copiarLista(caminoActual, camino);
    } else {
        ListaGenerica<Arista<String>> ady = mapaCiudades.listaDeAdyacentes(vIni);
        ady.comenzar();
        while (!ady.fin()) {
            Vertice<String> vAux = ady.proximo().verticeDestino();
            if (!visitados[vAux.posicion()]) {
                dfsCamino(mapaCiudades, vAux, vFin, caminoActual, camino, visitados);
            }
        }
    }
    caminoActual.eliminarEn(caminoActual.tamanio()-1);
    visitados[vIni.posicion()] = false;
}

private void copiarLista(ListaGenerica<String> caminoActual, ListaGenerica<String> camino) {
    while (!camino.esVacia())
        camino.eliminarEn(0);
    caminoActual.comenzar();
    while (!caminoActual.fin()) {
        camino.agregarFinal(caminoActual.proximo());
    }

}
```