RELATÓRIO – SPRINT 1

Estruturas de Informação – LEI 2022/23

Gonçalo Ribeiro
1210792
Luís Monteiro
1211250
Miguel Marques
1201078
Pedro Mesquita
1211171
Ruben Vilela
1211861

2DG G064

Introdução

Neste projeto, foi-nos pedido o desenvolvimento de um conjunto de funcionalidades que permitam gerir a informação relativamente a uma rede de distribuição de cabazes entre agricultores e clientes.

Os agricultores/produtores disponibilizam diariamente à rede de distribuição os produtos e respetivas qualidades que têm para vender e os clientes (particulares ou empresas) colocam encomendas (cabazes de produção agrícola) à rede de distribuição.

Se a totalidade de um produto disponibilizado por um agricultor para venda, num determinado dia, não for totalmente expedido, fica disponível nos dois dias seguintes, sendo eliminado após esses dois dias. Os cabazes são expedidos num determinado dia, quer sejam totalmente satisfeitos ou não, sendo necessário registar para cada produto a quantidade encomendada, a quantidade entregue e o produtor que forneceu. Cada produto de um cabaz é fornecido por um só produtor, mas um cabaz pode ser fornecido por vários produtores.

A rede gere a distribuição dos produtos dos agricultores de modo a satisfazer os cabazes a serem entregues em hubs para posterior levantamento pelos clientes. Um hub é localizado numa empresa e cada cliente (particular ou empresa) recolhe as suas encomendas no hub mais próximo.

Todo o projeto foi realizado na linguagem de programação Java.

US301: Construir a rede de distribuição de cabazes

Neste User Story, a partir da informação fornecida nos ficheiros. O grafo deve ser implementado usando a representação mais adequada, que definimos ser um MUDAR ISTO, garantindo a manipulação indistinta dos clientes/empresas e produtores agrícolas, de código C, E e P, respetivamente.

Este US baseia-se em inserir as arestas no grafo. Inicialmente, passamos os vértices para uma árvore binária, de forma a ser mais fácil obter o objeto do local após ler apenas o nome no ficheiro das distâncias. Assim, ao ler o nome do local de origem e do local de destino é mais fácil obter os seus objetos para criar a aresta e adicioná-la ao grafo.

```
goncrib +2
public static void main(String[] arg) throws FileNotFoundException {

    CsvReader readFiles=new CsvReader();

    File file1=new File( pathname: "C:\\Users\\pnsri\\OneDrive\\Ambiente de Trabalho\\Nova pasta\\Sem3pi\\src
    File file2=new File( pathname: "C:\\Users\\pnsri\\OneDrive\\Ambiente de Trabalho\\Nova pasta\\Sem3pi\\src

    final Graph<Local,Integer> map=new MapGraph<>( directed: false);

    BST<Local> locais=readFiles.ReadClientesProdutores(file2, separatorRegex: ",");
    BST<Destinatário> destinatários=readFiles.getDestinatários();
    readFiles.ReadDistancias(file1,file2, separatorRegex: ",",map,locais);
```

US302: Verificar se o grafo é conexo e devolver número mínimo de ligações para haver contacto entre quaisquer clientes

Para implementar esta funcionalidade, criamos um método, denominado ligaçoesMinimas, algDist e connected, com a função de conseguir o número de ligações mínimas para qualquer cliente/produtor conseguir contactar qualquer um outro, um método de Dijkstra para conseguir o número de ligações mínimas de cada origem, só usado se for verificado se o grafo é conexo no último método, respetivamente.

```java
public class US302 {
    public Graph<Local,Integer> map;
    2 usages  Luis Monteiro *
    public US302(Graph<Local,Integer> map){
        this.map = map;
    }
    3 usages  Luis Monteiro *
    public Map<String,Map<String,Integer>> ligaçoesMinimas(){
        if (!connected())return null;
        Map<String,Map<String,Integer>> resultado=new HashMap<>();
        Map<String,Integer> dist=new HashMap<>();
        Graph<Local,ArrayList<Local>> minimas=new MapGraph<~>( directed: false);
        LinkedList<Local> lista=null;
        MapVertex<Local,ArrayList<Local>> mapvertex=null;

        for(Local local:map.vertices()){
            dist=algDist(local);
            resultado.put(local.getName(),dist);
        }
        return resultado;
    }
    1 usage  Luis Monteiro *
    public Map<String,Integer> algDist(Local origem){
        Map<String,Integer> resultado=new HashMap<>();
        List<Local> fila=new ArrayList<>();
        int dist[]=new int[map.numVertices()];
        String path[]=new String[map.numVertices()];
        boolean visitados[]=new boolean[map.numVertices()];
        Local original=origem;
        List<Local> result=new LinkedList<>();

        for (int d=0;d<map.numVertices();d++){
            dist[d]=999;
            visitados[d]=false;
        }
     if (!map.validVertex(origem))return null;

     dist[map.key(origem)]=0;
     visitados[map.key(origem)]=true;
     for (Local verts:map.adjVertices(origem)) {
         if (!fila.contains(verts) && visitados[map.key(verts)]==false) {
             if (dist[map.key(verts)]>dist[map.key(origem)]+1){
                 dist[map.key(verts)]=dist[map.key(origem)]+1;
                 path[map.key(verts)]=origem.getName();
             }
             fila.add(verts);
         }
     }

     while(true){
         result.add(origem);
         for (int d=0;d<fila.size();d++){
             if (visitados[map.key(fila.get(d))]==false){
                 origem=fila.get(d);
                 visitados[map.key(origem)]=true;
                 fila.remove(d);
                 break;
             }
             fila.remove(d);
         }

         for (Local verts:map.adjVertices(origem)) {
             if (!fila.contains(verts) && visitados[map.key(verts)]==false) {
                 if (dist[map.key(verts)]>dist[map.key(origem)]+1){
                     dist[map.key(verts)]=dist[map.key(origem)]+1;
                     path[map.key(verts)]=origem.getName();
                 }
                 fila.add(verts);
             }
         }
```

```
        if (fila.size()==0){
            if (!result.contains(origem)){
                result.add(origem);
            }
            break;
        }
    }

    if (result.size()==0)return null;

    for (Local loc: map.vertices()){
        resultado.put(loc.getName(),dist[map.key(loc)]);
    }
    return resultado;
}
```

1 usage Luis Monteiro *

```
private boolean connected(){

    LinkedList<Local> result=null;
    List<Local> lista=map.vertices();
    result= Algorithms.BreadthFirstSearch(map,map.vertex( key: 0));

    if (result.size()==lista.size()){
        return true;
    }
    else return false;
}
```

Classe US302

US303: Definir os hubs da rede de distribuição

Para este User Story, é-nos pedido encontrar as N empresas mais próximas de todos os pontos da rede, sendo N um número escolhido pelo utilizador. A distância é calculada como a média do comprimento do caminho mais curto de cada empresa a todos os clientes e produtores agrícolas.

```java
public class US303
{
    7 usages   mig567 +1
    public static List<Local> findHubs(Graph<Local, Integer> map, int n)
    {
        if(n <= 0)
        {
            return null;
        }

        ArrayList<Local> comps = new ArrayList<>();

        for (Local l : map.vertices())
        {
            if (l.getDestinatário().charAt(0) == 'E')
            {
                comps.add(l);
            }

        }

        if ((comps.size() == 0))
        {
            return null;
        }

        Map<Local, Double> company_sum = new HashMap<>();

        for (Local c : comps)
        {
            Double sum_dist = 0.0;
            Double counter = 0.0;

            ArrayList<LinkedList<Local>>  shortPaths = new ArrayList<>();
            ArrayList<Integer> distances = new ArrayList<>();

            Algorithms.shortestPaths(map, c, Integer::compare, Integer::sum,  zero: 0, shortPaths, distances);

            for(Integer value : distances){
                sum_dist += value;
                counter++;
            }

            if(counter != 0){
                Double average = sum_dist/counter;
                company_sum.put(c, average);
            }

        }

        List<Map.Entry<Local, Double>> compLst = new LinkedList<>(company_sum.entrySet());

        compLst.sort(Map.Entry.comparingByValue());

        List<Local> hubs = new ArrayList<>();

        if (compLst.size() < n)
        {
            n = compLst.size();
        }
        for (Map.Entry<Local, Double> entry : compLst.subList(0,n))
        {
            hubs.add(entry.getKey());
        }
        return hubs;
    }
}
```

US304: Determinar o hub mais próximo

De modo a implementar esta funcionalidade, criamos uma classe ParClienteHub, na pasta Domain, para organizar os clientes, hubs e distâncias entre eles, com métodos getters, setters e toString.

Primeiro, começamos por chamar o US303, onde são definidas os hubs, para poderem ser usados.

```
public class US304 {

    pedromesquita
    public static List<ParClienteHub> findNearestHubs(Graph<Local, Integer> map, Integer numberOfHubs){
        List<Local> hubs = US303.findHubs(map, numberOfHubs);

        return findNearestHubsAlgorithm(map, hubs);
    }

    1 usage   pedromesquita *
    public static List<ParClienteHub> findNearestHubsAlgorithm(Graph<Local, Integer> map, List<Local> hubs){

        List<Local> clients = getClients(map);
        Integer mindist = Integer.MAX_VALUE;
        LinkedList<Local> shortpath = new LinkedList<>();
        Local hubToAdd = null;

        List<ParClienteHub> hubClosestToEachClient = new ArrayList<>();

        if(clients.size() == 0)
            return null;

        for(Local client : clients){

            for(Local hub : hubs){
                Integer distance = Algorithms.shortestPath(
                        map,
                        client,
                        hub,
                        Integer::compareTo,
                        Integer::sum,
                         zero: 0,
                        shortpath
                );
```

```
                if(distance < mindist){
                    mindist = distance;
                    hubToAdd = hub;
                }
            }

            hubClosestToEachClient.add(new ParClienteHub(client, hubToAdd, mindist));
            mindist = Integer.MAX_VALUE;
        }

        if(hubClosestToEachClient.size() == 0){
            return null;
        }

        return hubClosestToEachClient;
    }

    1 usage   pedromesquita
    public static List<Local> getClients(Graph<Local, Integer> map){
        List<Local> clients = new ArrayList<>();

        for(Local member : map.vertices()){
            if(!(member.getDestinatário().charAt(0) == 'P'))
                clients.add(member);
        }

        return clients;
    }

}
```

US305: Determinar a rede que conecte todos os clientes e produtores com uma distância mínima

Para determinarmos a distância mínima de todos os clientes e produtores, usamos o algoritmo mstGraph das aulas Prático-Laboratoriais, ou seja, colocamos o gráfico no algoritmo, e é devolvido a rede que conecta todos os clientes e produtores com uma distância total mínima entre si.

```
public class US305 {
    2 usages
    private GrafoDistancia grafoDistancia;
    2 usages
    private Algorithms algorithms;

    2 usages   1211861
    public US305(GrafoDistancia grafoDistancia) {
        this.grafoDistancia = grafoDistancia;
        this.algorithms = new Algorithms();
    }

    3 usages   1211861
    public MapGraph<Local, Integer> minimumGraph(){

        MapGraph<Local, Integer> graph;

        graph = algorithms.mstGraph(grafoDistancia.getMapGraph());

        return graph;
    }

    1211861
    public void toStringMinimumGraph(){

        MapGraph<Local, Integer> graph;

        graph = minimumGraph();

        System.out.println(">>>>> VERTICES: " + graph.numVertices() + "\n>>>> EDGES: " + graph.numEdges() + "\n" + graph.edges());
    }

}
```