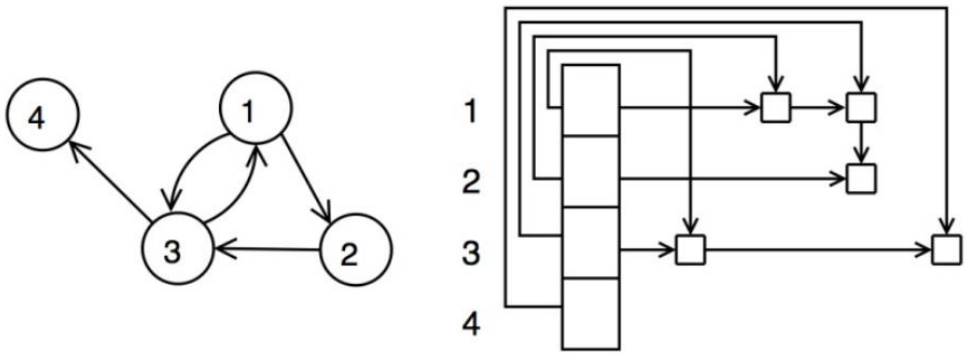
Pràctica 2 de Estructura de Dades

ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA
Universitat Rovira i Virgili

Estructures de Dades 2022

Multillistes d'adjacència (Millora)



Pedro Mendoza Fernandez

15-6-2022

Índex:

Contenido

1- Aspectes més rellevants de les vostres implementacions:

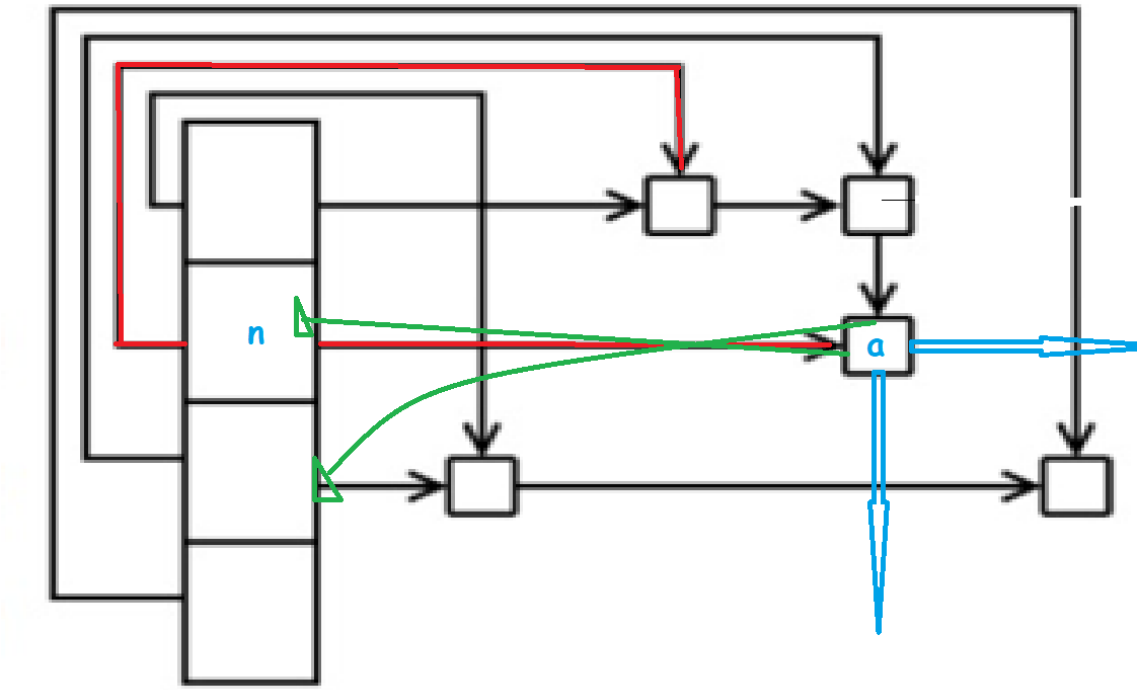
1.1 Fitxer json i array potencies.

Per començar primer que tot, he llegit el arxiu json on he anat mirant si la id_estacio ja era existent i si era així afegir la nova potencia a la ArrayList definida dins dins del nodeH, a mesura que he anat fent la pràctica me adonat que es possible que la implementació que he fet servir per implementar la multi llista d'adjacència ha esta prou enrevessada però una vegada començada hi amb el temps que he tingut per fer-la he fet el que he pogut , encara així he estat treballant molt amb ella i revisant si anava i si no anava perquè, que no funcionava i on.

Ha estat difícil ja que era laboriós corroborar la adjacència entre estacions, ja que havies de utilitzar altres mètodes implementats els quals podien tindre alguna falla.

1.2 Node hash i Node aresta i punters respectius

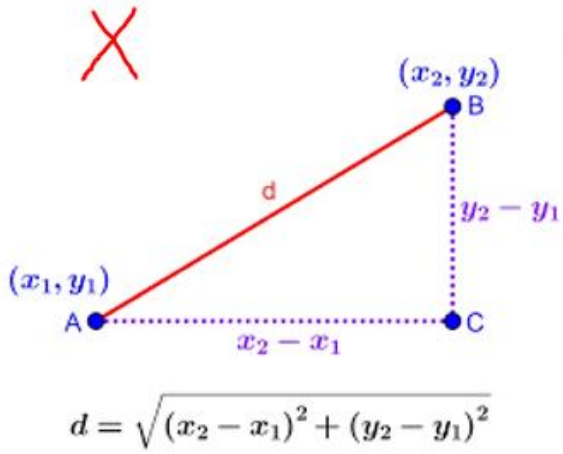
He decidit crear dos nodes, NodeH que serà els vèrtexs dels grafs on a cadascun estarà una estació amb la seva llista de endolls, pel que fa a les dades que al estar en genèrics serà la <V>, també consta de dos punters cap a arestes amb les quals ens indicarà cap a quina aresta esta connectat. Pel que fa al node_aresta, te 4 punters, 2 que mostren les següents arestes, i dos que indiquen si aquesta aresta esta connectada a un altre node i per finalitzar també guarda el valor entre aquestes dues.

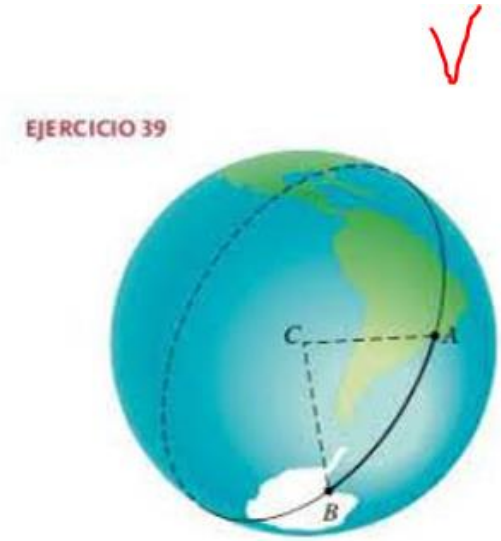


També quan creo el graf el que faig es passar-li per paràmetre un hashmap on estiguin ja tots els vèrtexs, es clar sense arestes com diu la pràctica.

1.3 Distància entre dos punts sobre la superfície de la Terra.

També he decidit fer la distància entre dues estacions agafant la distancia entre dos punts sobre la superfície de la terra ja que he pensat que seria més realista.





En conclusió la meva distància serà una mica mes llarga ja que la curvatura de la terra fa que sigui menys directa.

Buscant per internet he trobat que la formula era utilitzar els teoremes dels sinus i cosinus per tal de obtindré el resultat.

1.4 Funcions Graf

Més coses que he decidit com a decisions de disseny, són passar Strings a les funcions del graf

Ja que només necessito el id_estació per poder obtindré el valor.

Pel que fa a les pròpies funcions del graf, resumint ràpidament el funcionament per cada funció ha estat principalment recorre les arestes per obtindré alguna cosa, és a dir.

AfegirAresta: Volem recórrer les arestes fins trobar alguna que fos nul·la i llavors mitjançant els punters fer que es connecti al altre node.

ExisteixAresta: Volem recórrer per poder comparar el node que apunta la aresta amb el que ens passen per paràmetre.

Adjacents: Volem recórrer per poder trobar totes les connexions on tenia un node fins que fos null (no en te mes) i retornar-les en un ArrayList.

ValorAresta: Volem recórrer els nodes per tal de poder compara per observar si hem arribat al node passat per paràmetre i si es així poder obtindré el valor <E> que en aquesta cas es un double distancia. També hem tingut en conte si existia una aresta per poder evitar retornar valors nulls ja que entre dos arestes no veïnes no hi ha distancia.

3- Joc de Proves:

```
15167086
13382168
25.705127178202623
15167086
32316854
9.110795532909508
15167086
32316853
9.184769820066899
15167086
22276270
29.902352441706647
15167087
15167088
13.773352330818383
15167087
17544159
32.33971815499929
15167087
9008984
35.49510096202068
```

Podem observar com hem printat per pantalla un parell de estacions quan estan entre menys de 40km i consegüentment aquestes han de afegires una aresta entre elles.

```
System.out.println(grafFinal.existeixAresta("15167086", "13382168"));
System.out.println(grafFinal.valorAresta("15167086","13382168"));
```

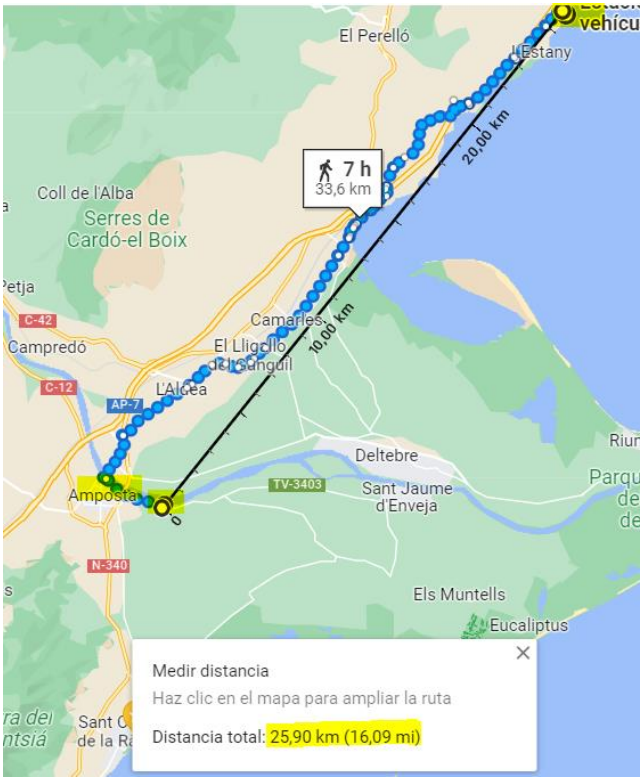
```
true
25.705127178202623
```

Observem com funciona i ens diu que hi ha aresta i el seu valor, podem comprovarho entre aquestes dues.

Com es mostra a google maps podem corroborar la exactitud del programa.


```
"id": "15167123",
"id_estacio": "15167086",
"nom": "EDAR Amposta - INGETEAM EdRSR 1",
"data": "2020-09-30 10:31:40",
"consum": "",
"carrer": "Amposta",
"ciutat": "Amposta",
"estat": "error",
"temps": "0",
"potencia": "22",
"tipus": "Mennekes (Type 2)",
"latitud": "40.704179",
"longitud": "0.6101"
```

```
"id": "13382209",
"id_estacio": "13382168",
"nom": "Port de l'Ametlla de Mar - INGETEAM EdRSR 7",
"data": "2022-04-19 16:45:09",
"consum": "",
"carrer": "Ctra. Puerto, s/n",
"ciutat": "L'Ametlla de Mar",
"estat": "lliure",
"temps": "0",
"potencia": "22",
"tipus": "Type-2 (Type 2)",
"latitud": "40.882774",
"longitud": "0.803978"
```

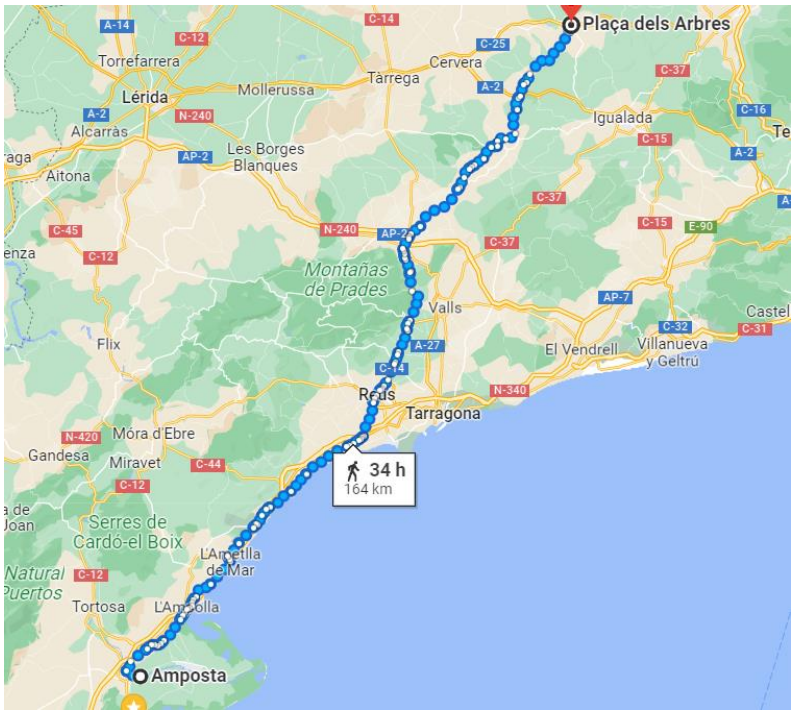
3.1 Joc de proves Algorisme de Dijkstra (camí optim)

He de dir que hagués estat bé aplicar el algoritme de A* vist a classe ja que seria una bonica aplicació per demostrar la variant de dijkstra però tinc més consolidat el normal.

```
ArrayList<String> hola = camiOptim("15167086","19090955" , 50, grafFinal, HashEstacio);
System.out.println(hola);
```


```
"id": "15167123",
"id_estacio": "15167086",
"nom": "EDAR Amposta - INGETEAM EdRSR 1",
"data": "2020-09-30 10:31:40",
"consum": "",
"carrer": "Amposta",
"ciutat": "Amposta",
"estat": "error",
"temps": "0",
"potencia": "22",
"tipus": "Mennekes (Type 2)",
"latitud": "40.704179",
"longitud": "0.6101"
```

```
"id": "18976851",
"id_estacio": "18976785",
"nom": "Calaf - LAFON EdRR 1",
"data": "2022-04-19 16:45:09",
"consum": "",
"carrer": "Plaça dels arbres s/n",
"ciutat": "Calaf",
"estat": "lliure",
"temps": "0",
"potencia": "55",
"tipus": "Type-2 (Type 2)",
"latitud": "41.7321543",
"longitud": "1.5137804"
```

Fem el algoritme i ens retorna el primer en 50 de autonomia (menys llocs de carrega)

I en 20 més llocs de carrega

He fet servir com a inspiració el pseudocodi del propi algorisme de dijsktra.

```
[19090955, 34886091, 19591404, 14849659, 7088644, 13382168, 15167086]
[19090955, 3386242, 7088710, 7088605, 12690152, 10643125, 12690203, 15016574, 7088636, 7088619, 13361452, 15167086]
```

Pel que fa al segon algorisme , aquest crec que he entès malament el funcionament i l'he encarat a fins a quina estació puc arribar en cas de que mirant la autonomia pugui anar contínuament de estació en estació.

```
ArrayList<String> hola2 = zonesDistMaxNoGarantida("15167086", 20, HashEstacio);
System.out.println(hola2);
```

```
[3386242, 14849659, 3386246, 3386244, 7088609, 10126083,
7088607, 15932118, 31683849, 3386248, 7088608, 14335614,
32641615, 7088605, 7088606, 7088603, 7088604, 7088612, 29068411,
29068410, 14849660, 7088611, 14849661, 12690220, 12690221,
14849666, 14849667, 14849668, 14849669, 12690228, 14849662,
14849663, 14849664, 14849665, 14222706, 7088619, 7088617,
7088623, 7088622, 7088620, 12690210, 320645, 12690213, 16655837,
16655836, 27562799, 16655835, 26560628, 12690216, 16655834,
12690217, 16655838, 16285113, 17551296, 32361384, 7088627,
7088625, 7088626, 7088634, 7088635, 7088632, 7088633, 15932584,
7088630, 7088631, 20106781, 12690202, 12690203, 27562667,
12690207, 15351619, 15351744, 15016780, 15000194, 9404311,
26628638, 7088638, 33400166, 7088636, 33400167, 35228586,
7088637, 26628639, 7088645, 27241694, 7088646, 27241693,
7088643, 7088644, 7088642, 9357225, 11319736, 11916669,
11916668, 13142489, 11916667, 22369408, 33016505, 13142488,
9464889, 26273820, 12690143, 33852300, 12690147, 12690148,
14997390, 34886092, 15209104, 34886091, 34886090, 13236356,
14849691, 14849692, 12690130, 14849693, 14849694, 14849690,
35688504, 15600689, 29110141, 20103021, 7088708, 7088709,
7088706, 36084694, 14849680, 14849681, 1771418, 14849682,
1771417, 7088710, 14849683, 34257894, 14849688, 14849689,
12690127, 14849684, 14849685, 14849686, 14849687, 33049981,
31541270, 13379714, 33049982, 11916670, 28469816, 23144754,
7088601, 7088602, 14849670, 14849671, 14849672, 12690114,
14849677, 14849678, 15016574, 7516161, 14849679, 14849673,
14849674, 14849675, 14849676, 30528693, 14345022, 18736142,
13238472, 30528459, 13238473, 15006003, 30528458, 12690190,
11916587, 26957077, 6136765, 6136766, 14742581, 9456394,
12690187, 19504077, 18111159, 14742580, 23597594, 13763193,
36095402, 29362332, 19296574, 19296573, 19296572, 13826718,
13213064, 13774192, 19504078, 32446592, 1297069, 14345023,
32446594, 32446593, 32446595, 19580959, 10124085, 13774184,
13774183, 13774185, 7088599, 13238489, 30214570, 7088597,
12690165, 22400287, 12690168, 22400288, 13636662, 30213004,
3701832, 12690152, 1114241, 35228859, 14500967, 9820183,
19898486, 7088649, 19898487, 7088647, 7088648, 7088652, 7088653,
7088650, 7088651, 32317257, 34886237, 34886236, 10349627,
13763014, 31540480, 14519594, 12690097, 26945069, 14624245,
14624246, 36096716, 16309250, 12690091, 15150831, 16309249,
12690085, 14827812, 14827814, 14827813, 9680024, 12690194,
12690196, 8177080, 12690078, 12690199, 6129709, 10824126,
13954521, 8946724, 30372438, 10124220, 3887525, 16117360,
```

13485757, 30372437, 3887522, 3887523, 3887524, 23881579,
31252402, 16862024, 13254325, 30625880, 31950524, 15147242,
33292133, 15939770, 13692838, 15939771, 13312860, 17544159,
18976785, 13692715, 19090955, 21596486, 21596485, 24497551,
33852299, 18481272, 35349594, 35349595, 15515358, 15515359,
29945442, 19808528, 29945441, 15196847, 15354661, 19808529,
17993294, 19808530, 13692773, 14091866, 18483265, 15515362,
15515363, 18111072, 15354672, 9904796, 13195461, 24791035,
9125721, 13632499, 14078903, 14078902, 31545188, 10832150,
8031989, 10832151, 25089403, 9008984, 18510906, 33845336,
15354657, 15354658, 15354656, 13860588, 24791135, 18146939,
27564117, 15515279, 21853884, 10666690, 34252185, 34252186,
2495435, 1299938, 13600679, 9907183, 10708508, 1300339,
30581311, 15354755, 31954734, 61735, 22356918, 4346732,
19591404, 4346731, 23131836, 22816321, 9903705, 13361309,
28326810, 33845386, 61741, 26272917, 61743, 15005930, 61745,
61747, 32641320, 61739, 32639122, 19355070, 24790358, 13865548,
19504419, 33845378, 10622506, 13816473, 23140516, 61752, 61755,
3296805, 3296806, 15560995, 61758, 23881089, 3887394, 5751546,
19895657, 61761, 15434782, 13331649, 24497472, 31953577, 61764,
9142, 9143, 13600518, 13828275, 13600516, 13600517, 20759732,
30372989, 13612731, 33344034, 33344035, 8271407, 20968323,
33344033, 21419359, 2253713, 21419360, 19895550, 29786118,
29786117, 35664708, 35664709, 19741092, 31684442, 24265595,
10643125, 10643124, 13382168, 19895549, 30528801, 30528923,
13236531, 22276270]

4Codi

```
public static void main(String[] args) throws IOException, ParseException, ErrorV_NoTrobat,
ErrorExisteixAresta, ErrorNoExisteixAresta  {


                double potencia =0;
                JSONParser parser = new JSONParser();
        FileReader reader = new FileReader("icaen.json");
        Object ob = parser.parse(reader);
        JSONArray array = (JSONArray)ob;


        hash<String,estacio<Double>,Double> HashEstacio = new hash<String,
estacio<Double>,Double>();


        for(int i = 0; i<array.size(); i++) {

        JSONObject objecte = (JSONObject)array.get(i);
        String key = (String) objecte.get("id_estacio");
        double latitud = Double.parseDouble((String) objecte.get("latitud"));
        double longitud = Double.parseDouble((String) objecte.get("longitud"));

        try {
        potencia = Double.parseDouble((String) objecte.get("potencia"));
        }
        catch(java.lang.NumberFormatException e) {
            potencia = 0;
        }
        estacio<Double> e1 = new estacio<Double>(key, latitud,longitud);


                if (!HashEstacio.teK(e1.getId_estacio())){

                e1.setALlista_endolls(potencia);
                nodeH<estacio<Double>,Double> nodeh = new nodeH<estacio<Double>,Double>(e1);
                HashEstacio.put(key,nodeh);


                }else {
                estacio<Double> eaux= HashEstacio.get(key).getV();
                eaux.setALlista_endolls(potencia);
                nodeH<estacio<Double>,Double> nodeh = new nodeH<estacio<Double>,Double>(eaux);
                HashEstacio.put(key,nodeh);


                }
                //System.out.println( HashEstacio.get(key).getV());
        }


        double d = 0;
        graf_generic<String,estacio<Double>,Double> grafFinal = new
graf_generic<String,estacio<Double>,Double>();
       //meter doble bucle de iterators para poder "afegir_aresta"
        grafFinal.crearGraf(HashEstacio.retornaTaula());
        Iterator<String> i_est1= HashEstacio.setK().iterator();
        Iterator<String> i_est2= HashEstacio.setK().iterator();
        double distMinima=111111111.0;
        estacio<Double> estAux = null;
        int p1= 0;

                while(i_est1.hasNext()) {

                        distMinima=1111111111;
                        String clau1 = i_est1.next();
                        estacio<Double> est1 = HashEstacio.get(clau1).getV();
                        i_est2 = HashEstacio.setK().iterator();
                        for (int i=0; i<=p1; i++) {
                                i_est2.next();
                        }

                        while(i_est2.hasNext()) {
```

```
                                String clau2 = i_est2.next();
                                estacio<Double> est2 = HashEstacio.get(clau2).getV();
                                d =
HashEstacio.calcularDist(est1.getLatitud(),est1.getLongitud(),est2.getLatitud(),
est2.getLongitud());

                                if(d<distMinima) {
                                        distMinima=d; // distancia minima que no sea menor a 40
ya que sino sera una aresta normal
                                        estAux = est2;
                                }
                                if (d<= 40 && d!=0 ) {  // si es menor a 40 haz aresta

                                        grafFinal.afegirAresta(est1.getId_estacio(),
est2.getId_estacio(), d);




                                        }
                        }
                        if (distMinima > 40 && est1.getId_estacio()!=estAux.getId_estacio()) {
                        grafFinal.afegirAresta(est1.getId_estacio(), estAux.getId_estacio(),
distMinima);

                        //System.out.println(est1.getId_estacio());
                        //System.out.println(estAux.getId_estacio());
                        //System.out.println(distMinima);

                        }
                        p1++;

                }
                System.out.println(grafFinal.existeixAresta("15167086", "13382168"));
                System.out.println(grafFinal.valorAresta("15167086","13382168"));
                grafFinal.valorAresta("32316853","22276270");


                ArrayList<String> hola = camiOptim("15167086","19090955" , 50, grafFinal,
HashEstacio);
                System.out.printldn(hola);
                ArrayList<String> hola1 = camiOptim("15167086","19090955" , 20, grafFinal,
HashEstacio);
                System.out.println(hola1);




                ArrayList<String> hola2 = zonesDistMaxNoGarantida("15167086", 20, HashEstacio);
                System.out.println(hola2);



        }

        // algorisme 1
        public static ArrayList<Stridng> camiOptim(String id1, String id2, double autonomia,
graf_generic<String,estacio<Double>,Double> grafFinal,hash<String,estacio<Double>,Double>
HashEstacio) throws ErrorV_NoTrobat, ErrorNoExisteixAresta{
                ArrayList<String> cami = new ArrayList<String>();
                if(HashEstacio.get(id1) != null && HashEstacio.get(id2) != null) {

                        ArrayList<estacio<Double>> veins = grafFinal.adjacents(id1);
                        ArrayList<estacio<Double>> aux = new ArrayList<estacio<Double>>();
                        HashMap<String, Double> HashDist = new HashMap<String, Double>();
                        HashMap<String, String> HashAnterior = new HashMap<String, String>();
                        boolean trobat=false;

                        Iterator<String> i_est1= HashEstacio.setK().iterator();


                        while(i_est1.hasNext()) {
```

```
                                String clau1 = i_est1.next();
                                aux.add( HashEstacio.get(clau1).getV());
                                HashDist.put(clau1, Double.MAX_VALUE );

                        }
                        HashDist.put(id1, 0.0);

                        double dmin;
                        String id_min = null;
                        double distAux;

                        while(!trobat && !aux.isEmpty()) {
                                sdmin = Double.MAX_VALUE;
                                for( estacio<Double> est : aux){
                                        distAux=HashDist.get(est.getId_estacio());
                                        if(distAux < dmin) {
                                                dmin= distAux;
                                                id_min= est.getId_estacio();
                                        }
                                }
                                if(id_min.equals(id2)) {
                                        trobat=true;
                                }else {
                                        aux.remove( HashEstacio.get(id_min).getV());
                                        veins= grafFinal.adjacents(id_min);
                                        for (estacio<Double> vei : veins) {
                                                distAux = HashDist.get(id_min)+
grafFinal.valorAresta(id_min, vei.getId_estacio());
                                                Double distEntreNodes =
grafFinal.valorAresta(id_min, vei.getId_estacio());
                                                if( HashDist.get(vei.getId_estacio()) > distAux
&& distEntreNodes <= autondomia && aux.contains(vei)) {
                                                        HashDist.put(vei.getId_estacio(),
distAux);
                                                        HashAnterior.put(vei.getId_estacio() ,
id_min);
                                                }
                                        }
                                }
                        }

                        do {
                                cami.add(id2);
                                id2=HashAnterior.get(id2);
                        }while(!id2.equals(id1));
                        cami.add(id1);


                }else {
                        throw new ErrorV_NoTrobat();
                }

                return cami;
        }

        //algorisme 2
        static ArrayList<String> zonesDistMaxNoGarantida(String id_ori,int autonomia,
hash<String,estacio<Double>, Double> HashEstacio){
                //INICIALITZEM
        hash<String,estacio<Double>, Double> hash=HashEstacio;
                Iterator<String> estacio1 = HashEstacio.setK().iterator();
                nodeH<estacio<Double>, Double> origen = hash.get(id_ori);
                estacio<Double> aux = origen.getV();
                estaciodw<Double> aux2;
                estacio<Double> aux1;

        double dist = 0;
        double distl = 0;d
        String key1=null;
        ArrayList<estacio<Double>> visitats = new ArrayList<>();
        ArrayList<String> noVisitats = new ArrayList<>();


        while(estacio1.hasNext()) {
```

```
        key1=null;
        key1= estacio1.next();
        //AFEGIR
        noVisitats.add(key1);
        }


                Iterator<String> e1 = noVisitats.iterator();



        while(e1.hasNext()) {
        key1=e1.next();
        //
        aux2 = hash.get(key1).getV();
                dist = calcularDist(aux.getLongitud(), aux.getLatitud(), aux2.getLongitud(),
aux2.getLatitud(  ));
                if(dist<autonomia) {
                        visitats.add(aux2);
                        hash.Esborrar(aux2.getId_estacio());
                }

                for(int i=0; i<visitats.size(); i++) {
                        auxl = visitats.get(i);
                distl = calcularDist(auxl.getLongitud(), auxl.getLatitud(), aux2.getLongitud(),
aux2.getLatitud());
                if(distl<autonomia) {
                        if(!visitats.contains(auxl)) {
                                visitats.add(aux2);
                                hash.Esborrar(aux2.getId_estacio());
                        }
                }
                }
         }

        estacio1 = hash.setK().iterator();
        noVisitats=new ArrayList<>();
        while(estacio1.hasNext()) {
                key1=null;
        key1=estacio1.next();
                noVisitats.add(key1);
        }

                return noVisitats;

        }


        //-----------------------------------------------------Calcular dist con la formula -----
---------------------------------
                public static double calcularDist(double l1,double long1, double l2, double
long2) {
                        l1= Math.toRadians(l1);
                        long1= Math.toRadians(long1);
                         l2= Math.toRadians(l2);
                        long2= Math.toRadians(long2);

                        final double Radi_Terra=6371.01;
                        double dist = Radi_Terra * Math.acos(Math.sin(l1) * Math.sin(l2) +
Math.cos(l1) * Math.cos(l2) * Math.cos(long1-long2));
                        return dist;
                }


}
```

```java
package aresta;

import hash.nodeH;

public class node_aresta<V, E> {
        private node_aresta<V, E> dreta;
    private node_aresta<V, E> baix;
    private nodeH<V,E> PNodeDreta;
    private nodeH<V,E> PNodeBaix;
    private E dist;

        public node_aresta(nodeH<V, E> PNodeDreta, nodeH<V, E> PNodeBaix,E dist) {
                super();
                this.PNodeDreta = PNodeDreta;
                this.PNodeBaix = PNodeBaix;
                this.dist = dist;
        }

        public node_aresta<V, E> getDreta() {
                return dreta;
        }

        public void setDreta(node_aresta<V, E> dreta) {
                this.dreta = dreta;
        }

        public node_aresta<V, E> getBaix() {
                return baix;
        }

        public void setBaix(node_aresta<V, E> baix) {
                this.baix = baix;
        }

        public nodeH<V, E> getPNodeDreta() {
                return PNodeDreta;
        }

        public void setPNodeDreta(nodeH<V, E> PNodeDreta) {
                this.PNodeDreta = PNodeDreta;
        }

        public nodeH<V, E> getPNodeBaix() {
                return PNodeBaix;
        }

        public void setPNodeBaix(nodeH<V, E> PNodeBaix) {
                this.PNodeBaix = PNodeBaix;
        }

        public E getDist() {
                return dist;
        }

        public void setDist(E dist) {
                this.dist = dist;
        }







}
package hash;

import aresta.node_aresta;

public class nodeH<V,E> {
         private node_aresta<V,E> areD;
         private node_aresta<V,E> areB;
         private V v;
```

```
	public nodeH(V v) {
		this.v = v;
	}

	public node_aresta<V, E> getAreD() {
		return areD;
	}

	public void setAreD(node_aresta<V, E> areD) {
		this.areD = areD;
	}

	public node_aresta<V, E> getAreB() {
		return areB;
	}

	public void setAreB(node_aresta<V, E> areB) {
		this.areB = areB;
	}

	public V getV() {
		return v;
	}

	public void setV(V v) {
		this.v = v;
	}



}
```