ENTREGA III

Breve introducción…

Se nos pidió solucionar un problema en el contexto de algoritmos voraces que nos diera para cada ciudad de una provincia el menor camino que se debería recorrer para transportar cargamento hacia alguno de los puertos ubicados en la misma. Por lo que dado un grafo no dirigido y ponderado de ciudades de una provincia y un conjunto de ciudades a evaluar, debemos implementar un algoritmo en pseudocódigo que nos devuelva para cada una de estas últimas ciudades, el camino hacia el puerto de menor costo.

Para esto, nos fue sugerido usar el algoritmo voraz de Dijkstra. Este algoritmo calcula la ruta más económica en este caso, siempre en líneas generales calcula la ruta óptima desde un vértice origen a todos los demás vértices de un grafo. Se basa en la idea de mantener la mejor distancia conocida de cualquier vértice al origen, y a medida que se descubren distancias mejores se actualizan estos valores.
Es un enfoque voraz ya que en un momento dado hay un conjunto de vértices de los cuales se conoce la ruta óptima al origen y además son los más cercanos a él. Del resto de los vértices no conocemos la ruta óptima, sino la mejor ruta encontrada hasta el momento que no tiene porque ser la óptima.
En su esquema este algoritmo usa dos vectores/arreglos, ambos de tamaño n (n= cantidad de elementos en el grafo) que van a almacenar la solución, uno tiene las distancias mínimas de cada uno de los vértices del grafo al vértice de origen y otro tiene los padres/predecesores de cada uno de los vértices, que marcan el camino que se deberá recorrer hacia el vértice de origen.

Resolución

Para la resolución del problema tuve en cuenta lo siguiente:

1) La funcion principal (caminoXciudad) recibe por parámetro el grafo en cuestion (provincia) y un conjunto de ciudades de las cuales quiero calcular la distancia al puerto. A partir de ahí se obtienen los puertos que tiene la provincia y por cada uno de los puertos se calcula la menor distancia desde ellos hacia el resto de las ciudades, siendo más fácil esta implementación desde los puertos que tenga la provincia que calculando los caminos por cada una de las ciudades y guardando esta información en un conjunto de Registros de Puertos el cual tendrá un arreglo de distancias y otro de padres. Una vez generado el Djikstra desde cada uno de los puertos, se imprime por cada ciudad el camino que deberá recorrer hacia el puerto de menor distancia.
2) La función Djikstra la cual fue explicada anteriormente recibe como parámetro un arreglo vacio de distancias y otro de padres, el cual deberá llenar con la información correspondiente para su posterior uso en el algoritmo general. Para seleccionar cada una de las ciudades que pasan por la evaluación del camino nos fijamos que sea la

de menor distancia hasta el momento en el arreglo de distancias y que no haya sido evaluada previamente (funcion Seleccionar).

3) La función getCamino recibe la ciudad a evaluar y los Registros de Puertos para calcular para esa ciudad el puerto que esta a menor distancia, una vez encontrado llama a la función obtenerCamino
4) La funcion armarCamino dada la ciudad en cuestion, el puerto más cercano y el arreglo de padres de ese puerto, lo que hace es armar el camino pidiendo los padres desde la ciudad hasta encontrarse con el puerto, una vez ocurrido esto devuelve un conjunto de ciudades que representan el camino.

Pseudocódigo

```
funcion caminoXciudad(Grafo g, ArrayList<Ciudad> ciudades)

    ArrayList<RegistroPuerto>caminoPuertos= VACIO
    puertos[]= g.getPuertos()

    por cada p en puertos hacer
        distancias[]= VACIO
        padres[]=VACIO

        Djikstra(g, p, distancias, padres)

        caminoPuertos.add(new RegistroPuerto(distancias, padres))
    fin por

    por cada c en ciudades hacer
        camino= getCamino(g, c, caminoPuertos)
        iterator it= camino.iterator()
        while(it.hasNext())
            System.out.println(it.next())
        fin while
    fin por
fin funcion


funcion Djikstra (Grafo g, Ciudad p, distancias[ ], padres[ ])

    solucion[]
    vertices[]= g.getVertices()

    por cada v de vertices hacer
        distancias[v]= INFINITO
        padres[v]= VACIO
```

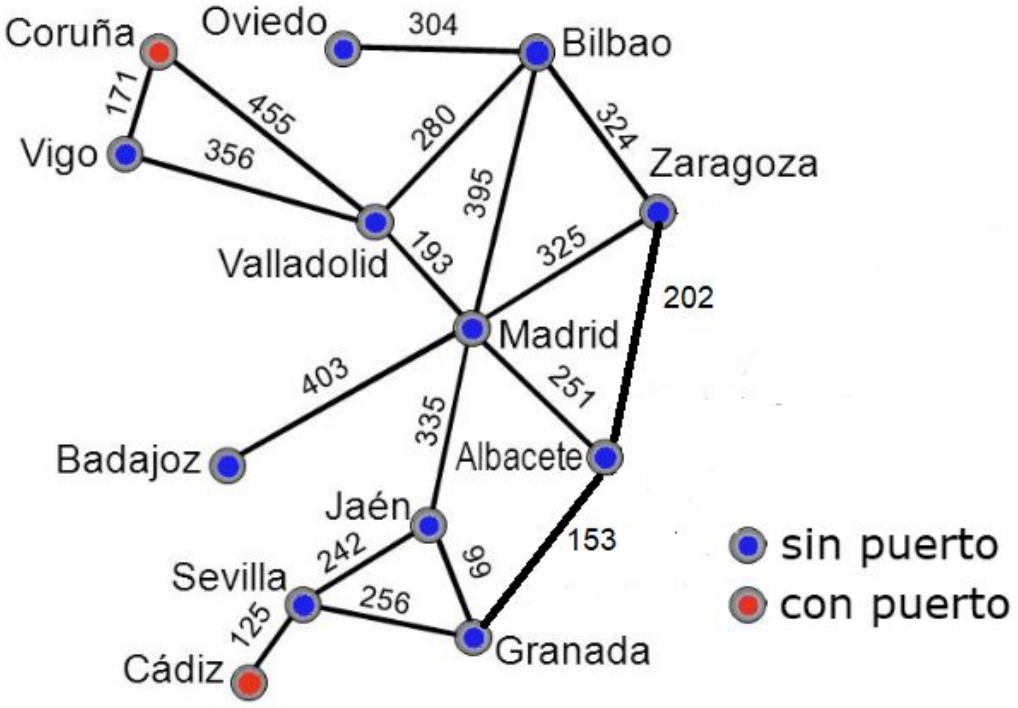
```
        fin por

        distancias[p]=0;

        mientras que ( solucion.size() < vertices.size() ) hacer
                x= seleccionar(distancias, solucion)
                solucion= {p}
                adyacentes[]= x.getAdyacentes()

                por cada v de adyacentes hacer
                        si ( !solucion.contains(v) ) hacer
                                si ( distancia[v] > distancia[x] + g.getPeso(x, v) ) hacer
                                        distancia[v]= distancia[x]+ g.getPeso(x,v)
                                        padre[v]= x
                                fin si
                        fin si
                fin por
        fin mientras
fin funcion
```

```
funcion getCamino(Grafo g, Ciudad c, ArrayList<RegistroPuerto>caminoPuertos)

        distanciaPuerto=INFINITO
        puertoCercano=NULL
        padresCamino[]=VACIO

        por cada r en caminoPuertos hacer
                distancias[]= r.getDistancias()
                si (distancias[c]<distanciaPuerto) hacer
                        distanciaPuerto= distancias[c]
                        puertoCercano=p
                        padresCamino= r.getPadres()
                fin si
        fin por


        si (puertoCercano!=NULL) hacer
                camino[]= armarCamino(puertoCercano, c, padresCamino)
        fin si

        return camino[]

fin funcion
```

```
funcion seleccionar ( distancias[ ], solucion [ ] )

        menor= INFINITO
        ciudadMenor= NULL

        por cada c en distancias hacer
                si (distancias[c]<menor) AND (!solucion.contains(c)) hacer
                        menor=distancias[c]
                        ciudadMenor= c
                fin si
        fin por

        return ciudadMenor

fin funcion


function armarCamino ( Ciudad p, Ciudad c, padres[ ] )

        retorno[]= VACIO
        Ciudad aux = padres[c]
        boolean corte= FALSE

        mientras que !corte hacer
                retorno.agregar(aux)
                aux= padres[aux]
                si (aux==p) hacer
                        retorno.agregar(aux)
                        corte=true
                fin si
        fin mientras

        return retorno

fin funcion
```

Seguimiento

Para no extender mucho el seguimiento tengo en cuenta el Dijkstra para uno de los puertos y no para todos (Coruña).



Iteracion 0: distancias[]= INFINITO / padres[]=NULL

S: {}

C	Coruña	Vigo	Vallado	Oviedo	Bilbao	Madrid	Sevilla	Jaén	Badajo	Cádiz
D	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
P	null	null	null	null	null	null	null	null	null	null

Iteracion 1: Coruña

S:{Coruña}

C	Coruña	Vigo	Vallado	Oviedo	Bilbao	Madrid	Sevilla	Jaén	Badajo	Cádiz
D	0	171	455	∞	∞	∞	∞	∞	∞	∞
P	Coruña	Coruña	Coruña	null	null	null	null	null	null	null
Ady		171	455							

Iteracion 2: Vigo

S:{Coruña, Vigo}

C	Coruña	Vigo	Vallado	Oviedo	Bilbao	Madrid	Sevilla	Jaén	Badajo	Cádiz
D	0	171	455	∞	∞	∞	∞	∞	∞	∞
P	Coruña	Coruña	Coruña	null	null	null	null	null	null	null
Ady	en S		527							

Iteracion 3: Valladolid

S:{Coruña, Vigo, Valladolid}

C	Coruña	Vigo	Vallado	Oviedo	Bilbao	Madrid	Sevilla	Jaén	Badajo	Cádiz
D	0	171	455	∞	535	648	∞	∞	∞	∞
P	Coruña	Coruña	Coruña	null	Vallado	Vallado	null	null	null	null
Ady	en S	en S			535	648				

Iteracion 4: Bilbao

S:{Coruña, Vigo, Valladolid, Bilbao}

C	Coruña	Vigo	Vallado	Oviedo	Bilbao	Madrid	Sevilla	Jaén	Badajo	Cádiz
D	0	171	455	839	535	648	∞	∞	∞	∞
P	Coruña	Coruña	Coruña	Bilbao	Vallado	Vallado	null	null	null	null
Ady			en S	839		930				

Iteracion 5: Madrid

S:{Coruña, Vigo, Valladolid, Bilbao,Madrid}

C	Coruña	Vigo	Vallado	Oviedo	Bilbao	Madrid	Sevilla	Jaén	Badajo	Cádiz
D	0	171	455	839	535	648	∞	983	1051	∞
P	Coruña	Coruña	Coruña	Bilbao	Vallado	Vallado	null	Madrid	Madrid	null
Ady			en S		en S			983	1051	

Iteracion 6: Oviedo

S:{Coruña, Vigo, Valladolid, Bilbao,Madrid, Oviedo}

C	Coruña	Vigo	Vallado	Oviedo	Bilbao	Madrid	Sevilla	Jaén	Badajo	Cádiz
D	0	171	455	839	535	648	∞	983	1051	∞
P	Coruña	Coruña	Coruña	Bilbao	Vallado	Vallado	null	Madrid	Madrid	null
Ady					en S					

Iteracion 7: Jaén

S:{Coruña, Vigo, Valladolid, Bilbao,Madrid, Oviedo, Jaén}

C	Coruña	Vigo	Vallado	Oviedo	Bilbao	Madrid	Sevilla	Jaén	Badajo	Cádiz
D	0	171	455	839	535	648	1225	983	1051	∞
P	Coruña	Coruña	Coruña	Bilbao	Vallado	Vallado	Jaén	Madrid	Madrid	null
Ady						en S	1225			

Iteracion 8: Badajoz

S:{Coruña, Vigo, Valladolid, Bilbao,Madrid, Oviedo, Jaén, Badajoz}

C	Coruña	Vigo	Vallado	Oviedo	Bilbao	Madrid	Sevilla	Jaén	Badajo	Cádiz
D	0	171	455	839	535	648	1225	983	1051	∞
P	Coruña	Coruña	Coruña	Bilbao	Vallado	Vallado	Jaén	Madrid	Madrid	null
Ady						en S	1356			

Iteracion 9: Sevilla

S:{Coruña, Vigo, Valladolid, Bilbao,Madrid, Oviedo, Jaén, Badajoz, Sevilla}

C	Coruña	Vigo	Vallado	Oviedo	Bilbao	Madrid	Sevilla	Jaén	Badajo	Cádiz
D	0	171	455	839	535	648	1225	983	1051	1350
P	Coruña	Coruña	Coruña	Bilbao	Vallado	Vallado	Jaén	Madrid	Madrid	Sevilla
Ady								en S	en S	1350