Pseudocódigo para la cuarta entrega:

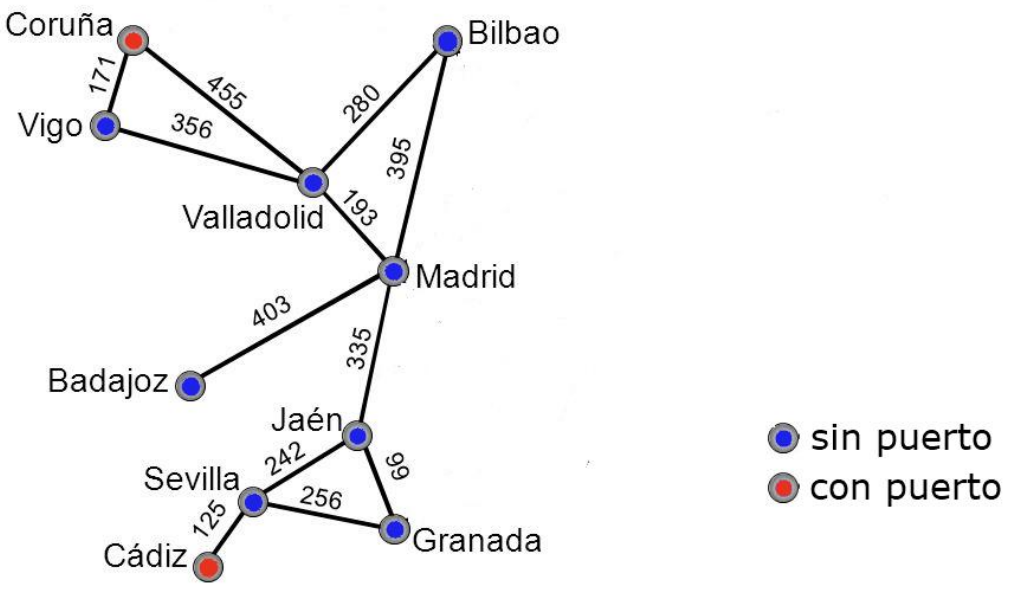
```
//C es el conjunto de elementos a los que se quieren llegar y O es el punto de origen de la busqueda

Función greedy(conjunto<candidatos> C, origen O){

conjunto<candidatos> S

distancias[] D = seleccionar(C,O)

conjunto<candidatos> S = factible(D,C)

If(!S.vacio())

        return solución(S,D)

else

        return “no_hay_solucion”

}


//seleccionar se encarga de usar dijkstra para crear una tabla con las distancias más cortas entre ciudades

Función seleccionar(conjunto<candidatos> C, origen O){

conjunto[][] resultado

For i = 0 to C.size()-1

        If(i == O.posicion)

                resultado[I]=0

        else

                resultado[i] = 10000

dijkstra(O,resultado)

Return resultado

}
```

```
//algoritmo dijkstra

Función dijkstra(origen O,conjunto[] resultado){

Int pos = O.posicion

Int sig

For i = 0 to resultado.size()-2{

        Resultado[pos] = 1

        Valor= resultado[pos]

        Adyacentes = conjunto.get(pos)

        For j = 0 to adyacentes.size()-1{

                If(j = 0)

                        Sig = buscarPoscion(adyacente.get(j)

                Tmp = valor + adyacente.get(j).getDistancia()

                If(tmp< resultado[buscarPoscion(adyacente.get(j)]){

                        resultado[buscarPoscion(adyacente.get(j)]) = tmp

                        if(resultado[sig]>tmp)

                                sig= buscarPoscion(adyacente.get(j)

                }

        }

        Pos=sig

}

}
```

```
//
Función factible(conjunto<candidatos> C, conjunto[] D){
conjunto<candidatos> S
For i = 0 to C.size()-1{
        Pos = buscarPoscion(C.get(j))
        If(resultado[pos]<10000)
                S.add(c.get(i))
}
}

//
Función solución(conjunto<candidatos> S, conjunto[] D){
Int valor
Int pos
For(i = 0 to S.size()-1{
        If(i ==0){
                Pos  = buscarPoscion(C.get(j))
                Valor  =  resultado[buscarPoscion(pos)]
        }
        Else{
                valorTemporal = resultado[buscarPoscion(C.get(j))]
                if(valor>valorTemporal){
                pos = buscarPoscion(C.get(j))
                valor = valorTemporal
        }
}
}
```

Seguimiento:

Tomamos el diagrama de referencia



Si partimos de una ciudad como madrid para llegar a un puerto cuando ejecutemos lo que primero va a hacer es tratar de obtener un array con la distancia minima para llegar al resto de las ciudades.

La tabla del seguimiento seria

Se toma a Madrid como origen

Madrid	bilbao	valladolid	virgo	coruña	jaén	granada	sevilla	cádiz	Badajoz
0	395	193	10000	10000	335	10000	10000	10000	403

Valladolid se toma com siguiente punto para el dijkstra

Madrid	bilbao	valladolid	virgo	coruña	Jaén	granada	sevilla	cádiz	Badajoz
0	395	193	356	455	335	10000	10000	10000	403

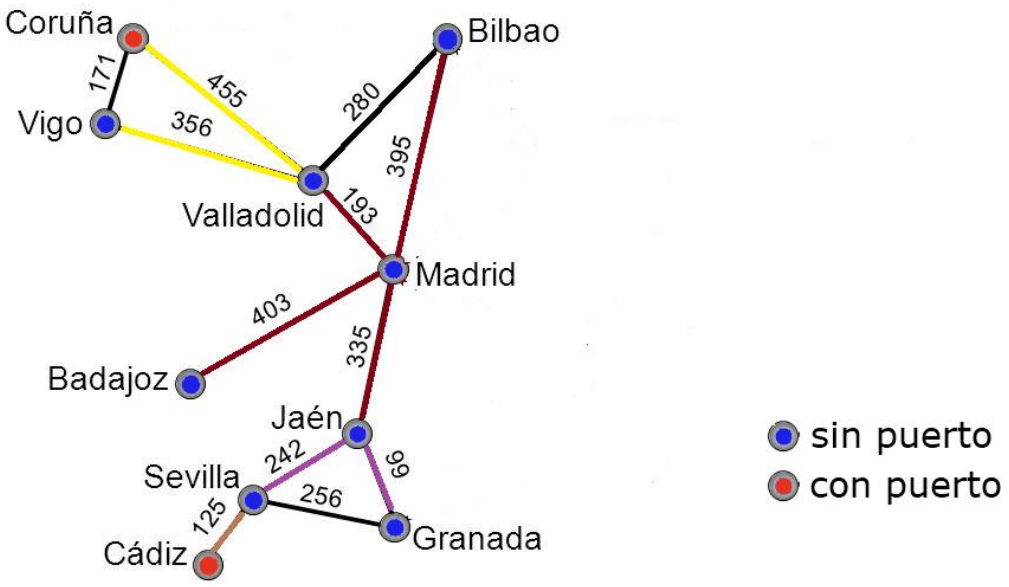
Jaén es el siguiente que modifica los valores de la tabla

Madrid	bilbao	valladolid	virgo	coruña	jaén	granada	sevilla	cádiz	Badajoz
0	395	193	356	455	335	434	677	10000	403

Sevilla es el último de los que genera un cambio en la tabla

Madrid	bilbao	valladolid	virgo	coruña	jaén	granada	sevilla	cádiz	Badajoz
0	395	193	356	455	335	434	677	802	403

Para el resto de las ciudades la distancia de la tabla no cambia



Factible decidiría si los valores de distancia para entran en un rango valido. En este caso coruña y cádiz son factibles.

Solución toma la ciudad con la menor distancia del origen que pertenece a el resultado de factible. En este caso coruña es la solucion