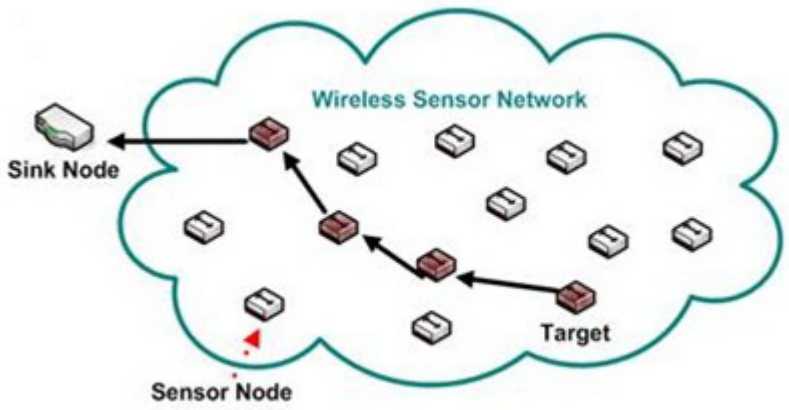
Práctica 3
Algoritmos Voraces

Joaquín Sergio García Ibáñez
Juan Navarro Maldonado

Índice:

Ejercicio 1

Una red de sensores inalámbrica está compuesta por múltiples nodos sensores desplegados en un entorno (invernadero, campo de cultivo, instalación industrial, perímetro de vigilancia de seguridad, ...), cada uno equipado con un transmisor de datos inalámbrico de un alcance reducido. Cada cierto tiempo, cada sensor debe enviar los datos recolectados del entorno a un servidor de datos central (sink). Sin embargo, la distancia entre el nodo sensor y el servidor central puede ser elevada como para enviar todos los datos directamente, por lo que será necesario, en ocasiones, enviar los datos por otros nodos sensores que hagan de enlace intermedio (ver Figura 1). Nos interesa enviar los datos con la máxima velocidad posible por lo que, para cada par de nodos sensores de la red ni, nj (entre los que se incluye el servidor central), conocemos el tiempo de envío entre ambos nodos como t(ni, nj) – el tiempo que se tarda en enviar los datos desde el nodo ni al nodo nj) -. El valor t(ni, nj) podría tener valor infinito si la red inalámbrica no permite enviar datos directamente desde el nodo ni hasta el nodo nj. Se pide: desarrollar un algoritmo que nos permita conocer por cuáles nodos sensores intermedios debe enviar los datos cada nodo sensor, hasta llegar el servidor central, de modo que se tarde el mínimo tiempo en la transmisión desde cada nodo hasta el servidor central.



Diseño de componentes

La idea general de este ejercicio es obtener un conjunto de secuencias de nodos/aristas que definen un camino mínimo entre un nodo origen que en nuestro caso será el servidor central a todos los nodos sensores que hacen de enlace intermedio.

Suponemos que los nodos están numerados entre el 0 y n-1
La solución serán dos vectores N y T:

- N[i] contiene el elemento anterior por el que hay que pasar en el camino mínimo entre el nodo inicial S dado y el nodo i
- T[i] contiene el tiempo que tardaría en recorrer entre el nodo inicial S dado y el nodo i

Existe una matriz L, donde cada componente M[i][j] indica el coste de viajar desde el nodo i al nodo j directamente

- **Lista de candidatos:** Los diferentes sensores que componen la red de sensores
- **Lista de candidatos utilizados**: Todos los sensores que se han considerado para insertarse en la solución
- **Criterio de selección:** Se seleccionara los nodos cuyo tiempo de propagación sea mínimo
- **Criterio de factibilidad:** Un sensor se insertará en la solución si, al insertarla, su tiempo de propagación es mínimo
- **Función solución:** Una solución será solución final al problema cuando los sensores producen un camino entre todos ellos hasta el sensor central
- **Función objetivo:** Minimizar la suma de los tiempos de las aristas de modo que minimicen el tiempo de propagación entre todos los nodos

Diseño del algoritmo

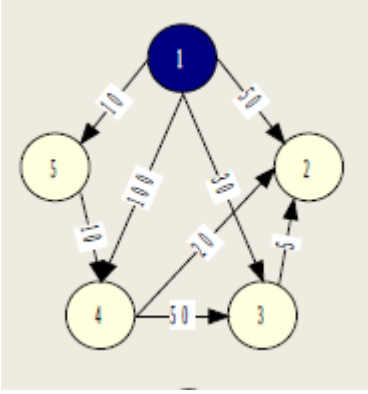
Algoritmo [N,T] = Dikjstra(G=(V,A),M,S)

```
n=|V|
C = V / {S}
Para i=1 hasta n, hacer: N[i] = L[S][i]; P[i] = S
Repetir n-2 veces:
    V=elemento de C tal que T[v] es mínimo
    C=C\{v}
    Para cada w en C, hacer:
        Si T[w] < T[v]+L[v][w], entonces:
            T[w]=T[v]+L[v][w] v;
        Fin-Si
    Fin-Para
Fin-Repetir
Devolver N,T
```

Estudio de optimalidad

Ejemplo paso a paso de la explicación del funcionamiento del algoritmo para una instancia pequeña propuesta por el estudiante

Sea el siguiente grafo:



En el cual sabemos que el nodo central es el 1, tenemos que hallar el camino mínimo que nos una todos los nodos. Para ello usaremos el algoritmo de Dijkstra.
Sabemos también que existen 5 nodos y 8 aristas ponderadas. Por lo tanto el algoritmo el algoritmo se desarrolla de la siguiente forma:

Paso	**V**	**C**	**D**
0	-	2,3,4,5	0,50,30,100,10
i=1	5	2,3,4	0,50,30,20,10
i=2	4	2,3	0,40,30,20,10
i=3	3	2	0,35,30,20,10

Por lo tanto el coste de la solución es 95

Correcto funcionamiento de la implementación

```
juan@DESKTOP-UELB1G3:/mnt/c/Users/Wavin/Documents/Universidad/Tercero/ALG/Greedy$ ./Ejercicio1.bin 5 8 1 2  50 1 3 30 1 4 100 1 5 10 5 4 20 4 2 20
4 3 50 3 2 5
Algoritmo de Dijkstra en C++
Ingrese Cantidad de Vertices:
5 8 1 2  50 1 3 30 1 4 100 1 5 10 5 4 20 4 2 20 4 3 50 3 2 5
Ingrese Cantidad de Aristas:
Ingrese Origen:
Ingrese Destino:
Ingrese Peso de la Arista:
Ingrese Origen:
Ingrese Destino:
Ingrese Peso de la Arista:
Ingrese Origen:
Ingrese Destino:
Ingrese Peso de la Arista:
Ingrese Origen:
Ingrese Destino:
Ingrese Peso de la Arista:
Ingrese Origen:
Ingrese Destino:
Ingrese Peso de la Arista:
Ingrese Origen:
Ingrese Destino:
Ingrese Peso de la Arista:
Ingrese Origen:
Ingrese Destino:
Ingrese Peso de la Arista:
Ingrese Origen:
Ingrese Destino:
Ingrese Peso de la Arista:
Ingrese Vertice Inicial:
1
Ingrese Vertice Final:
2
Longitud del Camino mas Corto: 95
```

Ejercicio 2

Un autobús realiza una ruta determinada entre su origen y su destino (n kilómetros en total). Con el tanque de gasolina lleno, el autobús puede recorrer k kilómetros sin parar. El conductor dispone de un listado con las gasolineras existentes en su camino, y el punto kilométrico donde se encuentran. Se pide: Diseñar un algoritmo greedy que determine en qué gasolineras tiene que repostar el conductor para realizar el mínimo número de paradas posible.

Diseño de componentes

La idea general del algoritmo es que debe de encontrar los contenedores cuyos pesos sean los mayores del resto de contenedores que aún no hayan sido cargados en el buque.

- **Lista de candidatos:** Las gasolineras para que el autobús reporte las mínimas veces.
- **Lista de candidatos utilizados**: Las gasolineras insertadas en la solución.
- **Criterio de selección:** Comprobaremos que gasolineras más alejadas podemos llegar a partir de nuestra posición para poder llegar al destino.
- **Criterio de factibilidad:** El candidato seleccionado cc se insertará en la solución siempre que sea una estación a la cual esté lo más alejada posible y podamos llegar con nuestra gasolina.
- **Función solución:** El algoritmo parará cuando se llegue al destino.
- **Función objetivo:** Encontrar las gasolineras más alejadas pudiendo llegar con la gasolina que tenemos y así hacer el mínimo número de paradas.

Diseño del algoritmo

Algoritmo S= Greedy(distancia: distancia al objetivo, num_gasolineras_ num de gasolineras que hay en el camino, gasolineras: gasolineras con su distancia, combustible: combustible que tenemos en el autobus)

```
    i = 0
    Mientras combustible <= distancia
        Si combustible] <= la distancia a la gasolinera iesima
        num_paradas++
        distancia = distancia - distancia a la gasolinera iesima
        Seleccionamos el contenedor i y añadimos a S
        i++
        Fin Si
        En otro caso
            i++
        Fin En Otro Caso
    Devolver S
Fin Algoritmo
```

Estudio de optimalidad

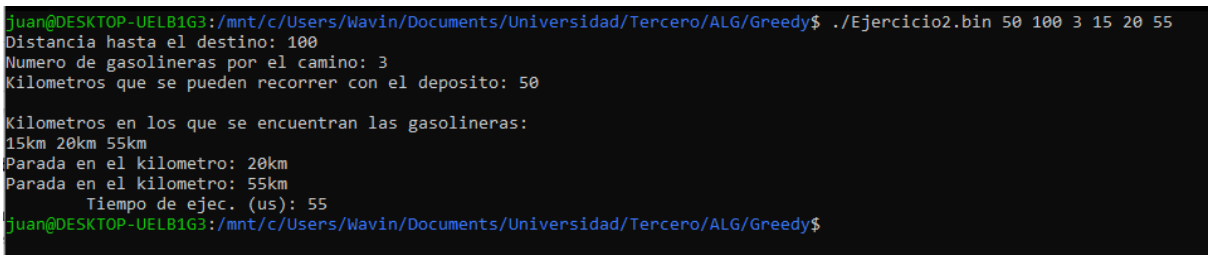
En este caso intentamos hallar un contraejemplo para ver si nuestra solución que venía dada por encontrar una gasolinera más alejada de nuestro punto y que podamos llegar con la gasolina que tenemos sin parar y así hacer el mínimo número de paradas era óptima, pero nos dimos cuenta que si encontramos otra solución que sea diferente a la nuestra se va a tener que hacer el mismo número de paradas si ponemos otro criterio de recorrer el mapa con lo cual llegamos a la conclusión de que nuestro algoritmo es óptimo.

Ejemplo paso a paso de la explicación del funcionamiento del algoritmo para una instancia pequeña propuesta por el estudiante

En el ejemplo propuesto, sabemos que se trata de un grafo dirigido, donde no se realizan bucles y solamente existe un camino. Los nodos serían las distintas gasolineras que existen en el camino y las aristas la distancia a cada una de ellas. Un ejemplo de esto seria que se desea viajar desde un punto A a un punto B en el cual la distancia entre ambos es 100km, existen 3 gasolineras entre ambos puntos en los kilómetros 15km 20km y 55km del punto A.

Para ello nuestro algoritmo propuesto comprueba si se puede realizar la parada desde el punto A al B sin necesidad de realizar ninguna parada, si no se puede, pararía en la gasolinera más lejana al punto en el que se encuentra, es decir, primero pararía en el kilómetro 20 y luego en el kilómetro 55.

Correcto funcionamiento de la implementación



Ejercicio 3

Se tiene un buque mercante cuya capacidad de carga es de k toneladas, y un conjunto de contenedores c1, ... , cn cuyos pesos respectivos son p1 ,..., pn (expresados también en toneladas). Teniendo en cuenta que la capacidad del buque podría ser menor que la suma total de los pesos de los contenedores, se pide: Diseñar un algoritmo que permita decidir qué contenedores hay que cargar para maximizar la suma de los pesos de los contenedores a transportar en el barco.
MOCHILA

Diseño de componentes

La idea general del algoritmo es que debe de encontrar los contenedores cuyos pesos sean los mayores del resto de contenedores que aún no hayan sido cargados en el buque.

- **Lista de candidatos:** Los contenedores del conjunto inicial C.
- **Lista de candidatos utilizados**: Los contenedores que ya han sido insertados en la solución.
- **Criterio de selección:** Ordenaremos de mayor a menor según el peso de los contenedores y seleccionaremos como candidato aquel contenedor que hace que su peso sea mayor que los contenedores existentes.
- **Criterio de factibilidad:** El candidato seleccionado cc se insertará en la solución siempre que sea un contenedor cuyo peso sea mayor que el del resto que quede aun.
- **Función solución:** El algoritmo parará cuando la suma de los pesos de los contenedores sea igual o superior sin superar a la capacidad de carga del buque k.
- **Función objetivo:** Encontrar los contenedores con mayor peso.

Diseño del algoritmo

Algoritmo S = Greedy(p:pesos de cada contenedor, n: num de contenedores, k: carga maxima, tam_sol: tamaño de la solucion)

```
Ordenar de mayor a menor el vector de pesos p[]
i = 0
sum = 0
Mientras i <=n
    Si sum + p[i] <= k
    Seleccionamos el peso del contenedor i y añadimos a S
    sum = sum + p[i]
    i++
    tam_sol++
    Fin Si
```

En otro caso
i++
Fin En Otro Caso
Devolver S
Fin Algoritmo

Estudio de optimalidad

Sea $T = \{c_1, \dots, c_n\}$ y suponemos que los pesos de los contenedores están ordenadas de mayor a menor $p_1 <= p_2 <= \dots <= p_n$

Si todos los $p_i = 1$, entonces la solución óptima es trivial.

En el caso contrario, la solución que proporciona el algoritmo greedy se formará por el conjunto $S = \{c_1, c_2, \dots, c_n\}$ de modo que:

$$\sum_{c_i} = \sum_{i=1}^{m} p_i <= k \; y \sum_{i=1}^{m+1} p_i <= k$$

Por tanto, el beneficio de la solución de P es máximo global

Ejemplo paso a paso de la explicación del funcionamiento del algoritmo para una instancia pequeña propuesta por el estudiante

Para un ejemplo de 5 contenedores cuyos pesos son respectivamente 1, 5, 3, 20, 15 toneladas respectivamente y cuya carga máxima es 25. Lo primero que haría nuestro algoritmo sería ordenar ese vector de orden mayor a menor, después lo que haría sería comprobar si se ha alcanzado ya a comprobar todos los contenedores y si no en el bucle comprobaría si la suma acumulada de los pesos más el peso del i ésimo contenedor es menor o igual a la carga máxima que puede llevar el buque, en ese caso seleccionaremos el el contenedor i y sumaremos a nuestra suma acumulada de pesos el peso de ese contenedor, después sumaria uno al contenedor para que la siguiente iteración del bucle comprobaria el siguiente contenedor, en otro caso sumaria uno al contenedor para que la siguiente iteración compruebe el siguiente contenedor

Correcto funcionamiento de la implementación

Por último para comprobar el correcto funcionamiento del problema propuesto, vamos a ejecutar el problema anterior en el ordenador. De esta forma deberíamos obtener que los contenedores que se almacenan en el buque sean los de 20 y 15 toneladas.