GRADO EN INGENIERÍA INFORMÁTICA

CURSO 2023-2024

UNIVERSIDAD
DE GRANADA

ALGORÍTMICA

Práctica 3 - Algoritmos Voraces

Miguel Martinez Azor

Ángel Rodríguez Faya

Alejandro Botaro Crespo

Alberto Parejo Bellido

Alejandro Ocaña Sánchez

16/04/2024

ÍNDICE

Problema 1

Supongamos que tenemos n estudiantes en una clase y queremos crear con ellos equipos formados por parejas (podemos suponer que n es un número par). Se dispone de una matriz p de tamaño n×n en la que p(i, j) indica el nivel de preferencia que el estudiante i tiene para trabajar con el estudiante j. El valor del emparejamiento del estudiante i con el j es $p(i, j) * p(j, i)$. Se trata de encontrar un emparejamiento para todos los estudiantes de forma que se maximice la suma de los valores de los emparejamientos.

Diseño de componentes:

- Lista de candidatos: Los estudiantes de la clase no emparejados
- Lista de candidatos usados: Los estudiantes que se han ido emparejando
- Función solución: El número de parejas es n/2
- Criterio de selección: El estudiante se empareja con el estudiante que tenga más valor de emparejamiento dentro de los que todavía no se han emparejado,
- Criterio de factibilidad: Un estudiante solo se puede emparejar una vez
- Función objetivo: Maximizar la suma de los valores de los emparejamientos

Diseño del algoritmo

- Idea general:

- Se parte de una lista de estudiantes pares.
- En cada paso, se empareja un estudiante con otro con el que tenga más valor de emparejamiento de los que están disponibles (el criterio de selección usado) y se suma el valor de cada pareja.
- Se repite el procedimiento hasta que no queden más estudiantes por emparejar, lo que viene a ser n/2 parejas.

- Pseudocódigo:

```
function AsignarPareja(C[1..n][1..n])
        U ← {2, . . . , n}        // Estudiantes a emparejar
        X ← {0}n·n
        E[1..n]          // Estudiantes emparejados
        for all i ∈ [1, . . . , n] do
                if i no emparejado then
                        j ← estudiante j donde C[i ][j ] sea máximo e j ∈ U
                        X[i][j] = C[i][j] * C[j][i]
                        E[i] = emparejado
                        E[j] = emparejado
                end if
        end for
        return X
end function
```

Ejemplo paso a paso del funcionamiento para un caso pequeño

- Supongamos que cogemos como ejemplo a los 5 estudiantes de nuestro grupo y al profesor de Algorítmica.
- Cada uno tiene una relación con el resto de personas del 1 al 10 siendo 1,

X	Alberto	Míchel	Alejandro	Ángel	Alejandro^2	Profesor
Alberto	-	4	7	9	3	10
Míchel	3	-	10	5	8	10
Alejandro	8	6	-	3	4	10
Ángel	6	5	6	-	9	10
Alejandro^2	5	9	7	2	-	10
Profesor	10	10	10	10	5	-

no quiero verlo ni en fotos, y un 10, me caso con la persona.

- En esta tabla podemos ver la relación de cada una de las personas con el resto. Podemos observar que todos los alumnos tienen una relación muy buena con el Profesor y el Profesor con ellos menos con uno que ha perdido su simpatía ahora que ve cómo se va a resolver este problema.

1. El algoritmo lo primero que hará será emparejar al primer estudiante de nuestro grupo con la persona con la que tenga mejor relación, en este caso, Alberto se emparejará con el (querido) Profesor que es la persona con la que se lleva mejor.

   Alberto*Profesor = 10*10 = 100

2. Alberto y Profesor pasan a estar emparejados y por lo tanto no pueden volver a emparejarse con nadie más.
3. Pasamos ahora a Míchel que se emparejará con Alejandro que le cae de 10 por lo que no seguimos mirando.

   Míchel*Alejandro = 10*6 = 60

4. Ahora tocaría Alejandro pero ya está emparejado por lo que continuamos con Ángel que se tendrá que emparejar obligatoriamente con Alejandro² que es el único que queda sin pareja.

   Ángel*Alejandro² = 9*2 = 18

5. La suma de todas las parejas sería entonces: 100+60+18 = 178

Estudio de optimalidad

Si analizamos de nuevo la tabla podemos ver un resultado más óptimo, por lo que no es un algoritmo óptimo siempre.

X	Alberto	Míchel	Alejandro	Ángel	Alejandro²	Profesor
Alberto	-	4	7	9	3	10
Míchel	3	-	10	5	8	10
Alejandro	8	6	-	3	4	10
Ángel	6	5	6	-	9	10
Alejandro²	5	9	7	2	-	10
Profesor	10	10	10	10	5	-

Alberto*Ángel

Míchel*Alejandro²

Alejandro*Profesor

Alberto podría sacrificar su especial relación con el Profesor para emparejarse con Ángel que le cae un poquito peor, aunque a Ángel sí le molesta más el cambio. Luego Míchel tendría que sacrificarse también porque hace muy buena pareja con Alejandro² que mejora bastante su pareja de antes. Y ya finalmente Alejandro es el afortunado que puede hacer pareja con el Profesor.

Tendríamos esta vez una suma de: 54+72+100 = 226

Problema 2

Se va a celebrar una cena de gala a la que asistirán n invitados. Todos se van a sentar alrededor de una única gran mesa circular, de forma que cada invitado tendrá sentados junto a él a otros dos comensales (uno a su izquierda y otro a su derecha). En función de las características de cada invitado (por ejemplo categoría o puesto, lugar de procedencia,...) existen unas normas de protocolo que indican el nivel de conveniencia de que dos invitados se sienten en lugares contiguos (supondremos que dicho nivel es un número entero entre 0 y 100). El nivel de conveniencia total de una asignación de invitados a su puesto en la mesa es la suma de todos los niveles de conveniencia de cada invitado con cada uno de los dos invitados sentados a su lado.

Se desea sentar a los invitados de forma que el nivel de conveniencia global sea lo mayor posible.

- Diseño de componentes:

Lista de candidatos: Lista que contiene a todos los invitados que son susceptibles de ser sentados en la mesa, es decir, que aún no han sido asignados/sentados en la mesa.

Lista de candidatos usados: Lista que contiene los invitados a los cuales ya se les ha asignado un lugar en la mesa, es decir, ya están sentados.

Función solución: Determina si a todos los n invitados se les ha asignado un puesto en la mesa, es decir, si la mesa está completamente llena y no queda ningún invitado por sentar en ella.

Criterio de selección: Determina cuál de los invitados i que quedan por sentar en la mesa es más conveniente a sentar en la mesa seleccionandolo mediante el criterio de que este tenga la mayor suma de niveles de conveniencia con respecto a los invitados contiguos (izquierda y derecha) que ya están sentados en la mesa.

Criterio de factibilidad: Determina si un invitado puede o no sentarse junto a los invitados que ya estén sentados en la mesa sin que se deje de cumplir la restricción de que tiene que haber dos invitados por comensal.

Función objetivo: Determina el novel de conveniencia total de los invitados asignados a sitios en la mesa, el objetivo será maximizar esta cantidad.

- Diseño del algoritmo (en base a la plantilla greedy):

- Idea General

-Partimos de la base de que la mesa se encuentra inicialmente vacía.

-Seleccionamos un invitado de la lista de candidatos, en este caso los invitados, basándonos en el que tenga la mayor suma de niveles de conveniencia con los invitados ya sentados (el "más prometedor").

-Eliminamos al invitado de la lista de candidatos.

-Intentamos asignarle un sitio en la mesa cumpliendo con que el nivel de conveniencia sea el mayor posible.

-Añadimos al invitado a la lista de candidatos seleccionados

-Comprobamos si se ha cumplido que todos los invitados hayan sido asignados maximizando el nivel de conveniencia, si se cumple se ha hallado la solución y sino se escoge otro candidato y se repite el proceso.

- Pseudocódigo

```
function invitados(n, conveniencia):
    S ← { }  // Lista de candidatos seleccionados
    C ← { …. }  // Lista de candidatos inicial
    mientras haya elementos en C y no EsSolucion(S) :
        x ← Seleccionar(C)
        C ← C - {x}
        si EsFactible( {x}) entonces:
```

S ← {x}

si EsSolucion(S) entonces:

devuelve S

sino:

devuelve "No hay solución"

// Función de selección

función Seleccionar(C):

devuelve el candidato de C con la mayor suma de niveles de conveniencia con los invitados ya sentados

// Función de factibilidad

función EsFactible(S):

devuelve verdadero si se puede asignar un asiento al último invitado seleccionado sin violar las restricciones de la mesa

// Función solución

función EsSolucion(S):

devuelve verdadero si todos los invitados han sido asignados a la mesa y se ha maximizado el nivel de conveniencia total

- Ejemplo paso a paso del funcionamiento para un caso pequeño

En este ejemplo tenemos 5 invitados y 5 sitios en la mesa.

El nivel de conveniencia de sentar un invitado con otro es el siguiente:

	Invitado 1	Invitado 2	Invitado 3	Invitado 4	Invitado 5

Inv 1	X	10	20	30	40
Inv 2	10	X	15	25	35
Inv 3	20	15	X	10	20
Inv 4	30	25	10	X	15
Inv 5	40	35	20	15	X

1.- Partimos de la mesa vacia-> Candidatos (C) = [1,2,3,4,5]

Seleccionados (S) = [ ]

2.- Escogemos el invitado que maximiza la suma de niveles de conveniencia con los invitados ya asignados. Como la mesa está vacía, seleccionamos cualquier invitado. Seleccionamos al invitado 1

3.- Al estar la mesa vacia el invitado 1 se puede sentar sin problema, es factible.

4.- Añadimos al invitado 1 a los candidatos seleccionados y lo eliminamos de la lista de candidatos a seleccionar:

S = [1]

C = [2,3,4,5]

Suma Total = 0

5.- Como la lista de candidatos no esta vacia por lo tanto quedan asientos por asignar asi que continuamos escogiendo otro candidato:

Buscamos el candidato cuyo nivel de conveniencia es mayor con respecto al candidato 1 -> Inv 2, Suma de conveniencia -> 0+10 = 10

Inv 3, Suma de conveniencia -> 0+20 = 20

Inv 4, Suma de conveniencia -> 0+30 = 30

Inv 5, Suma de conveniencia -> 0+40 = 40

Seleccionamos el invitado 5 ya que su nivel de conveniencia (Suma) con respecto al invitado 1 es el mayor de todos (fila 5 columna 1)

6.- Comprobamos que el asignar al invitado 5 junto al invitado 1 no provoca conflictos (es factible) por lo tanto sentamos al invitado 5 junto al 1.

7.- Actualizamos la lista de candidatos pendientes de sentar (eliminando al 5 ya que ya ha sido sentado) y la lista de candidatos seleccionados (añadiendo al 5 por el mismo motivo)

S = [1, 5]

C = [2,3,4]

Suma Total = 40

8.- Como todavia quedan invitados por sentar, escogemos otro candidato para sentar.

9.- Buscamos el candidato cuyo nivel de conveniencia es mayor con respecto al candidato 5 -> Inv 2, Suma de conveniencia -> 40+35 = 75

Inv 3, Suma de conveniencia -> 40+20 = 60

Inv 4, Suma de conveniencia -> 40+15 = 55

Seleccionamos el invitado 2 ya que su nivel de conveniencia (Suma) con respecto al invitado 1 y 5 es el mayor de todos (fila 2 columna 5)

10.- Comprobamos que el asignar al invitado 2 junto al invitado 5 no provoca conflictos (es factible) por lo tanto sentamos al invitado 2 junto al 5.

11.- Actualizamos la lista de candidatos pendientes de sentar (eliminando al 2 ya que ya ha sido sentado) y la lista de candidatos seleccionados (añadiendo al 2 por el mismo motivo)

S = [1, 5, 2]

C = [3,4]

Suma Total = 75

12.- Como todavia quedan invitados por sentar, escogemos otro candidato para sentar.

13.- Buscamos el candidato cuyo nivel de conveniencia es mayor con respecto al candidato 2 -> Inv 3, Suma de conveniencia -> 75+15 = 70

Inv 4, Suma de conveniencia -> 75+25 = 100

Seleccionamos el invitado 4 ya que su nivel de conveniencia (Suma) con respecto al invitado 1, 5 y 2 es el mayor de todos (fila 4 columna 3)

14.- Comprobamos que el asignar al invitado 4 junto al invitado 2 no provoca conflictos (es factible) por lo tanto sentamos al invitado 4 junto al 2.

15.- Actualizamos la lista de candidatos pendientes de sentar (eliminando al 4 ya que ya ha sido sentado) y la lista de candidatos seleccionados (añadiendo al 4 por el mismo motivo)

S = [1, 5, 2, 4]

C = [3]

Suma Total = 100

16.- Como todavia quedan invitados por sentar, escogemos otro candidato para sentar, que en este caso el unico que queda es el 3 por lo tanto lo añadimos a candidatos seleccionamos y calculamos la suma con respecto al 4 y al 1 (mesa circular):

S = [1, 5, 2, 4, 3]

C = [ ]

Suma Total = 100 + 10 = 110 + 20 = 130

17.- Como ya no quedan mas invitados que sentar la suma total de conveniencia es 130.

- Estudio de optimalidad

A partir del ejemplo anterior en el cual se ha propuesto una solución óptima para la elección de los candidatos en dicho orden S = [1, 5, 2, 4, 3] con un coste total de 130 y donde se obtiene dicho resultado con una eficiencia en tiempo de O(n) elegimos otro ejemplo para corroborar que esta y solo esta es la posible combinación de candidatos que obtiene el coste máximo en la asignación, por ejemplo:

Dados los mismos candidatos C = [1,2,3,4,5], en el ejemplo anterior hemos comenzado eligiendo como primer candidato seleccionado al invitado 1, en este caso probamos con la elección del primer candidato con el invitado 5. El resultado de esta decisión dará una lista de candidatos seleccionados de S =5,1,4,2,3 con coste total de 130. Esto nos lleva a la conclusión y sin necesidad de explorar las posibles combinaciones de que este algoritmo va a encontrar la solución máxima siempre ya que con cada paso que se de al explorar los posibles candidatos para un invitado en concreto el tamaño de exploración se va a ir reduciendo ya que partimos de una mesa redonda donde no se pueden asignar dos sitios adyacentes a un mismo invitado y un invitado no puede ser sentado dos veces además de que cada invitado se le va a asignar el máximo nivel de conveniencia local.

Por lo tanto si comparamos este algoritmo con uno de fuerza bruta, el de fuerza bruta siempre va a hallar el óptimo pero con mucho coste ya que en cada paso habría que recorrer la matriz entera y en el caso del greedy da

también el óptimo pero en una eficiencia mucho menor ya que el tamaño n a recorrer se va a ir reduciendo en cada paso que se de.

Problema 3

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:

- Lista de candidatos: Lista de gasolineras ordenadas por su posición kilométrica
- Lista de candidatos usados: Lista de gasolineras seleccionadas para repostar.

- Criterio de selección: Seleccionar la gasolinera más lejana que pueda alcanzarse sin quedarse sin gasolina.
- Criterio de factibilidad: Verificar si el autobús puede llegar a la gasolinera seleccionada sin quedarse sin gasolina.
- Función solución: La solución es óptima cuando el autobús llega al destino sin quedarse sin gasolina.
- Función objetivo: Minimizar el número de paradas en gasolineras.

- Diseño del algoritmo (en base a la plantilla greedy):

- Idea General

1. Inicialización:
    a. Ordenar las gasolineras por su posición kilométrica
    b. Inicializar una lista vacía para las gasolineras seleccionadas
    c. Inicializar una variable para el kilometraje de autobús
2. Selección Greedy
    a. Seleccionar la gasolinera más lejana a la que podamos llegar sin agotar la gasolina.
    b. Añadimos dicha gasolinera a la lista de gasolineras.
    c. Actualizamos el kilometraje del autobús con la posición de la gasolinera que se haya seleccionado.
3. Solución
    a. Devolver la lista de gasolineras

- Pseudocódigo

```
Función gasolineras_min(gasolineras, distancia_total, capacidad_gasolina)
    Ordenar gasolineras por distancia
    gasolineras_repostar =[ ]
    kilometraje = 0
    mientras que kilometraje<distancia_total:
        gasolinera_seleccionada= null
        distancia_max= kilometraje+capacidad_gasolina
    for(gasolinera : gasolineras):
        si gasolinera está en el rango del autobus:
            gasolinera_seleccionada= gasolinera
        sino:
            romper el bucle
```

si gasolinera distinto de null:

agregar gasolinera seleccionada a gasolineras_repostar

actualizar kilometraje= posicion de gasolinera seleccionada

sino:

no se puede completar la ruta

devolver las gasolineras_repostar

- Estudio de optimalidad

Podemos demostrar que este algoritmo produce una solución óptima en este caso mediante contradicción. Para ello vamos a suponer que existe una solución óptima que no sigue el enfoque greedy, del cual podemos ver el ejemplo en el siguiente apartado. Entonces, existiría un gasolinera en el recorrido del autobús que el algoritmo no ha seleccionado, la cual se agrega a la solución de gasolineras_repostar y que reduciría el número de paradas. No obstante, el algoritmo greedy selecciona la gasolinera más lejana a la que podemos llegar sin repostar.

En conclusión, el algoritmo greedy en este problema es la solución más óptima.

- Ejemplo paso a paso del funcionamiento para un caso pequeño

Para poder mostrar un ejemplo del funcionamiento de este problema primero vamos a establecer los parámetros iniciales.

El Bus puede recorrer 200 kilómetros sin repostar.

La ruta total son 1000 kilómetros.

La gasolineras que tenemos en el camino se encuentran en los kilómetros 200 500 300 700 400 900

1. Ordenamos las gasolineras por orden de kilometraje 200 300 400 500 700 900.
2. Comenzamos la ruta en el kilómetro 0 y el depósito lleno.
3. Comenzaremos la ruta y llegamos al kilómetro 200 donde paramos a repostar .

4. Fijamos el siguiente objetivo, en este caso tenemos a nuestro alcance la gasolinera 300 y la 400, seleccionamos la del kilómetro 400 ya que es a la que más lejos podemos llegar sin repostar, seguidamente en este ejemplo pararemos en el kilómetro 500 700 y 900.
5. Una vez en el kilómetro 900, con el depósito lleno, partimos al destino sabiendo que nos van a sobrar 100 kilómetros de combustible, es decir puede terminar la ruta sin problemas.
6. Finalmente, como hemos parado en todas las gasolineras excepto la 300 devolveremos todas las gasolineras donde hemos parado, si no hubiésemos parado en una gasolinera, esa no la incluimos como es el caso:

   return 200 400 500 700 900

Problema 4

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 n i, nj (entre los que se incluye el servidor central), conocemos el tiempo de envío entre ambos nodos como t(n i, 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 (greedy) 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.

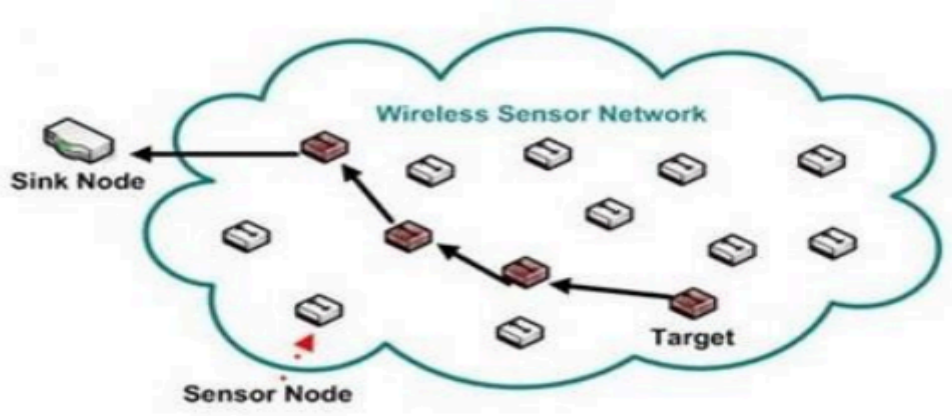


Figura 1: Configuración de envío desde un sensor inalámbrico (target) al receptor (sink). Fuente: https://elb105.com/pfc-design-and-implementation-of-a-dynamic-wireless-sensor-network

- Diseño de componentes:

- Lista de candidatos: Los nodos sensores en la red, excepto el servidor central. Cada nodo representa un sensor que necesita enviar datos al servidor central.
- Lista de candidatos usados: Al principio esta lista estará vacía y a medida que el algoritmo se ejecuta, se van añadiendo los nodos a esta lista una vez que se ha determinado la ruta más rápida al servidor central.
- Criterio de selección: Seleccionar el nodo que puede transmitir datos al servidor central en el menor tiempo posible. Este tiempo se calcula sumando el tiempo de transmisión que hay desde el nodo actual hasta el siguiente nodo intermedio y el tiempo que ya se ha tardado en llegar al nodo actual.
- Criterio de factibilidad: Un nodo es factible si no ha sido seleccionado antes(es decir, no está en la lista de candidatos usados) y si es posible transmitir datos desde este nodo al servidor central(lo que nos indica que el tiempo de transmisión no es infinito).
- Función solución: El algoritmo encuentra la solución cuando todos los nodos sensores han sido añadidos a la lista de candidatos usados, es decir, que se ha determinado la ruta más rápida para cada sensor para enviar sus datos al servidor central.
- Función objetivo: Minimizar el tiempo total de transmisión de datos desde todos los nodos servidores al central.

- Diseño del algoritmo (en base a la plantilla greedy):

- Idea General

El algoritmo que hemos diseñado se basa en el algoritmo de Dijkstra, que es un algoritmo greedy utilizado para encontrar la ruta más corta entre nodos de un grafo. En este caso, los nodos son sensores y el servidor central, y las aristas del grafo representan las conexiones entre los sensores. El peso de cada arista es el tiempo que se tarda en transmitir datos entre dos sensores.

El algoritmo comienza en el servidor central y explora los nodos vecinos. Para cada vecino, calcula el tiempo total de transmisión si los datos se envían a través de ese vecino. Si este tiempo es menor que el tiempo de transmisión actualmente conocido para ese nodo, actualiza el tiempo de transmisión y guarda el nodo actual como el nodo previo para ese vecino. Este proceso se repite hasta que se han explorado todos los nodos.

- **Pseudocódigo.**

Función buscarRutaMasCorta(defino nodos como Nodo[], defino inicio como entero)

Inicializo el tiempo de nodos en la posición inicio a 0.

Declaro visitado como visitado[] con tamaño y nodos.size() y lo inicializo a false.

Para i desde 0 hasta nodos.size() hacer

Declaro u como entero y lo inicializo a -1.

Para j desde 0 hasta nodos.size() hacer

Si visitado[j] es false y (u es igual a -1 o nodos[j].tiempo < nodos[u].tiempo) entonces

u es igual a j

Fin Si

Fin Para

Si el tiempo de nodos[u] es 1e9

break

Fin Si

Se le asigna a visitado en la posición u el valor true.

Para arista desde cada arista del vector nodos en la posición u

Declaro v como entero y la inicializo como arista.nodo

Declaro nuevoTiempo como entero y lo inicializo a al tiempo de nodos[u] + el tiempo de la arista.

Si nuevoTiempo es menor que nodos[v].tiempo

nodos[v].tiempo se le asigna nuevoTiempo
nodos[v].previo se le asigna u.

Fin Si

Fin Para

Fin Función

- Estudio de optimalidad

Para demostrar la optimalidad de nuestro algoritmo vamos a utilizar un argumento de contradicción, demostrando que nuestro algoritmo siempre encuentra la ruta de transmisión más óptima desde cada nodo sensor hasta el servidor central.

Supongamos que existe una ruta más rápida desde un nodo sensor hasta el servidor central que la ruta encontrada en nuestro algoritmo.

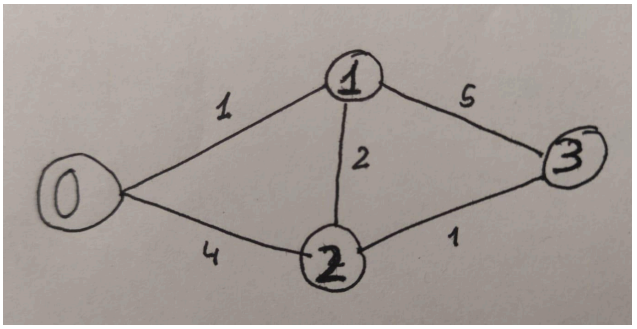
La contradicción está en que si esa ruta existiera, significa que hay al menos un nodo de esta ruta para el cual nuestro algoritmo no seleccionó la arista de menor tiempo de transmisión. Sin embargo, esto no es posible porque siguiendo la estrategia de nuestro algoritmo, siempre se selecciona la arista de menor tiempo de transmisión en cada paso.

Por lo tanto, nuestra suposición inicial debe ser falsa, lo que significa que nuestro algoritmo greedy siempre encuentra la rita de transmisión más óptima desde cada nodo sensor hast a cada nodo central.

- Ejemplo paso a paso del funcionamiento para un caso pequeño

Para el ejemplo suponemos que tenemos una red de sensores con 4 nodos (0, 1, 2, 3) donde el nodo 0 es el servidor central. Las conexiones entre los nodos y sus tiempos de transmisión son los siguientes:

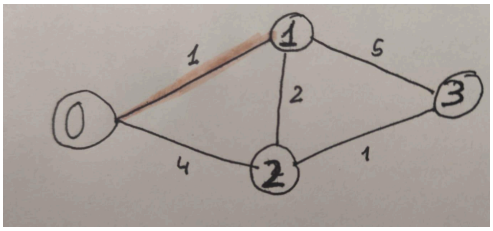
- Nodo 0 a Nodo 1: el tiempo es 1
- Nodo 1 a Nodo 2: el tiempo es 2
- Nodo 0 a Nodo 2: el tiempo es 4
- Nodo 2 a Nodo 3: el tiempo es 1
- Nodo 1 a Nodo 3: el tiempo es 5.



Las iteraciones que haremos serán las siguientes:

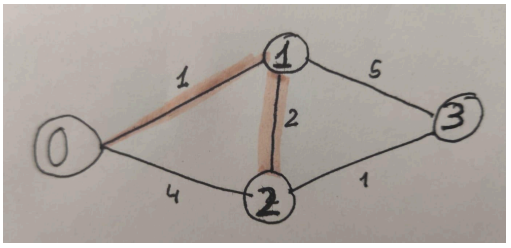
Iteración 1 (Inicialización): Primero inicializamos el tiempo de transmisión del nodo de inicio (nodo 0) a 0 y el tiempo de transmisión del resto de nodos a infinito (1e9).

Iteración 2: Comenzamos desde el nodo 0. Revisamos sus vecinos(el 1 y el 2). Para el nodo 1 el tiempo es 1, que es menor que el tiempo actual(que recordemos que es infinito), por lo que actualizamos el tiempo de transmisión del nodo 1 a 1 y marcamos el nodo 0 como su nodo previo. Hacemos lo mismo para el nodo 2, actualizando su

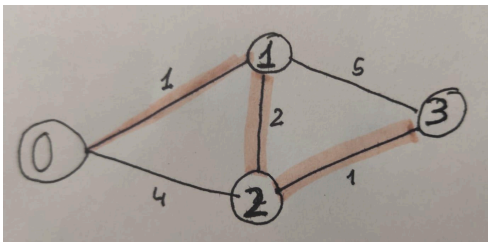


tiempo de transmisión a 4.

Iteración 3: Ahora seleccionamos el nodo que no hayamos visitado y que tenga menor tiempo de transmisión, que es el nodo 1, y revisamos sus vecinos(0 y 3). Para el nodo 3 el tiempo de transmisión es de 6 (1 del nodo 1 + 5 de la arista 1-3), que es menor que el tiempo actual que es infinito, por lo que actualizamos el tiempo de transmisión del nodo 3 a 6 y marcamos el nodo 1 como su nodo previo.



Iteración 4: De nuevo volvemos a seleccionar el nodo que no hayamos visitado y que tenga menor tiempo de transmisión, que es el nodo 2, y revisamos sus vecinos(0 y 3). Para el nodo 3 el tiempo de transmisión es de 5 (4 del nodo 2 + 1 de la arista 2-3), que es menor que la actual que es de 6 (1 del nodo 1 + 5 de la arista 1-3), por lo que actualizamos el tiempo de transmisión del nodo 3 a 5 y marcamos el nodo 2 como su



nodo previo.

Iteración 5: Finalmente, seleccionamos el nodo no visitado con el tiempo de transmisión más corto, que es el nodo 3. Como el nodo 3 no tiene vecinos no visitados, terminamos el algoritmo.

Al final obtenemos los tiempos de transmisión más cortos desde cada nodo hasta el servidor central, siendo estos:

- 0 para el nodo 0
- 1 para el nodo 1
- 4 para el nodo 2
- 5 para el nodo 3

También obtenemos la ruta más rápida desde cada nodo hasta el servidor central.

Problema 5

El alcalde de “Algovilla del Tuerto”, un conocido pueblo, desea renovar el embaldosado de las calles de su localidad. Sin embargo, las arcas del ayuntamiento no están muy saneadas y no se puede permitir embaldosar todas las calles del pueblo. El encanto de Algovilla reside en sus múltiples plazas (una en cada intersección de calles), que son monumentos locales, y en la belleza de pasear entre dichas plazas en época de verano, por lo que es un atractivo turístico que trae riqueza y trabajo a la localidad en esta época del año. El no realizar el embaldosado puede disminuir el atractivo del pueblo, la visita turística y, por tanto, reducir el desahogo económico que se produce en verano para sus habitantes, debido al turismo.

Para solucionar el problema, el concejal de urbanismo ha propuesto la siguiente solución: Gastar el menor dinero posible en asfaltar un conjunto de calles, de forma que siempre que se pueda llegar desde una plaza a cualquier otra a través de calles asfaltadas. Así, cualquier turista podrá disfrutar de los paseos entre estos monumentos. Como asesor, se te requiere que formulen el problema y lo resuelvas, proporcionando una solución que permita viajar desde cualquier plaza a cualquier otra plaza, con la restricción de que el paseo se realice siempre por una calle asfaltada y que el coste de asfaltar las calles necesarias para ello sea mínimo. Se te proporcionará información sobre qué plazas están unidas entre sí directamente por una única calle y el coste de asfaltarlas (suponer coste igual a +∞ cuando no exista una calle que une dos plazas de forma directa). Un ejemplo serían las siguientes plazas, calles y costes que se muestran en la Figura 2:

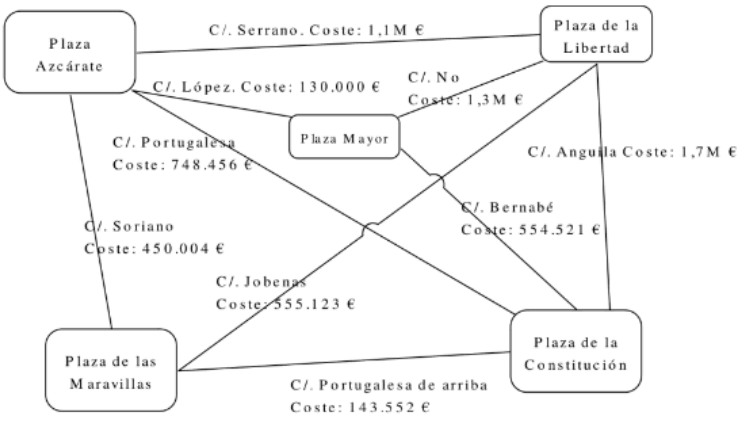


Figura 2: Esquema con las plazas del pueblo, las calles que las unen y el precio de asfaltar cada calle.

Diseño de componentes:

- Lista de candidatos: Calles no asfaltadas de Algovilla
- Lista de candidatos usados: Las calles que se han ido asfaltando
- Criterio de selección: Prioriza la calle de menor coste y que no conecte con una plaza ya conectada.(se miran todas las calles de plazas ya conectadas.)
- Criterio de factibilidad: Hay plazas sin conectar
- Función solución: Todas las plazas están conectadas por calles asfaltadas directa o indirectamente (desde cualquier plaza se puede llegar a otra)
- Función objetivo: Minimizar el coste del asfaltado de las calles conectando todas las plazas

Diseño del algoritmo (en base a la plantilla greedy):

- Idea General

- Se supone que no hay ninguna calle asfaltada y empezamos por cualquier plaza.
- En cada paso, se selecciona para asfaltar la calle más barata que una con una plaza que no esté conectada(según el criterio de selección escogido) y se miran todas las calles de plazas ya conectadas(Algoritmo de kruskal).
- Se repite el procedimiento hasta que no queden plazas por conectar.

- Pseudocódigo

```
function conectarPlazas(C[1..n][1..n])
        U ← {2, . . . , n}        // plazas a conectar
        X ← {0}
        E[1..n]          // plazas conectadas, si está conectada en su posición habrá un 1, en caso contrario un 0
        A[1…nCalles]←{ }
        i=0
        calles=0
        while calles<n-1
                if i no emparejado then// E[i]!=1
                        //Para todo nodo posible a conectar con el nodo i que no esten conectados,
                for all j que C[i][j]!=+∞ and e[j]==0
```

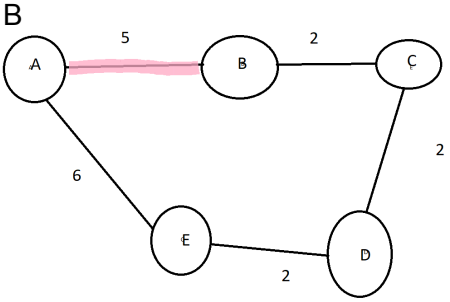
```
                a←hacer_arista(i,j)
                A←A U a
            end for
            b←∞
            for all a in A
                b←a si a<b
            end for
            A←A\{b}
            j← encontar_ciudad_destino(b,E)
            E[i] = 1
            X←X+ C[i][j]
            U ← U \ {i}
            i=j
        end while
        return X
end function
```

Estudio de optimalidad

Sí es óptimo y se puede demostrar porque siempre se escoge la calle más barata para conectar una plaza. Es irrelevante la plaza en la que empieces ya que siempre se va a elegir las mismas calles para asfaltar y siempre van a ser las más baratas sin repetir y se puede demostrar con reducción a lo absurdo:

Si empezamos por una plaza A que está conectada a una plaza B y una plaza C

y conectamos la plaza A con la B



Podríamos decir que era más óptimo llegar a B desde C pero entonces estaríamos diciendo que B-C,C-D,A-E y D-E < A-B, entonces A-E<A-B, Pero si hubiera sido así hubiera ido por E en vez de por B y nos daría una contradicción.

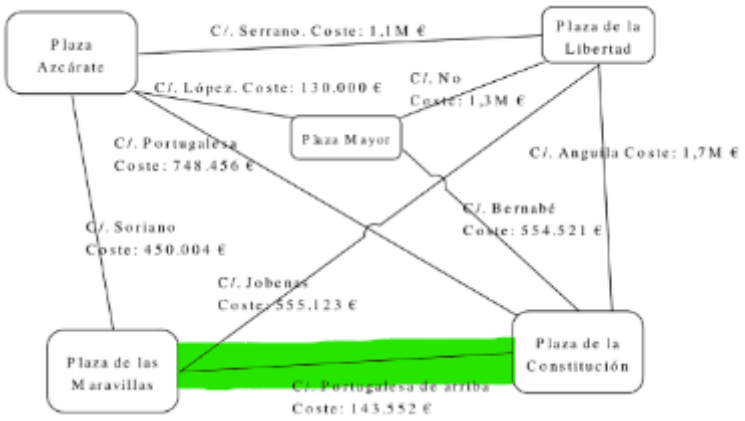
Ejemplo paso a paso del funcionamiento para un caso pequeño

Para este ejemplo vamos a usar el de la imagen del enunciado , para que se entienda mejor. Partimos de la matriz de costos

	P. de las Maravillas	P. de la constitución	P. de la Libertad	P.Azcárate	P.Mayor
P. de las Maravillas	—----	143 552	555 123	450 004	+∞
P. de la constitución	143 552	—----	1 700 000	748 456	554 521
P. de la Libertad	555 123	1 700 000	—----	1 100 000	1 300 000
P.Azcárate	450 004	748 456	1 100 000	—----	130 000
P.Mayor	+∞	554 521	1 300 000	130 000	—----

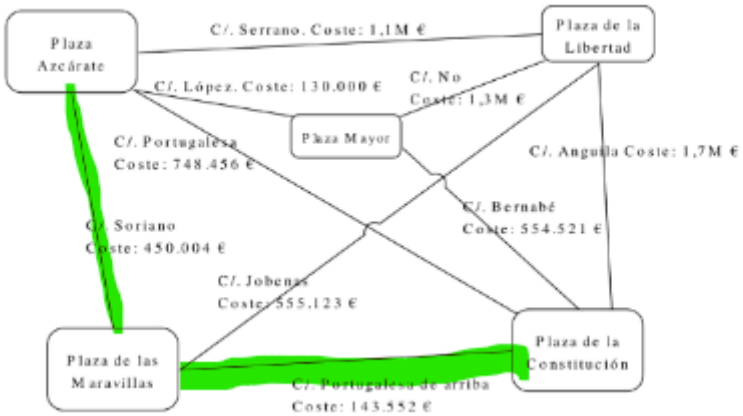
Una vez tengamos la matriz de costo empezamos el ejemplo con:

1. Se empezaria por la Plaza de la Maravillas y se comprueba que no esté conectada(como acaba de empezar el algoritmo no estaría conectada)
2. se comprueba el coste de la calles que conectan con la plaza de la maravilla
3. Al comprobar, sería la plaza de la constitución la elegida para conectar con la calle portuguesa de arriba (143 552 euros)



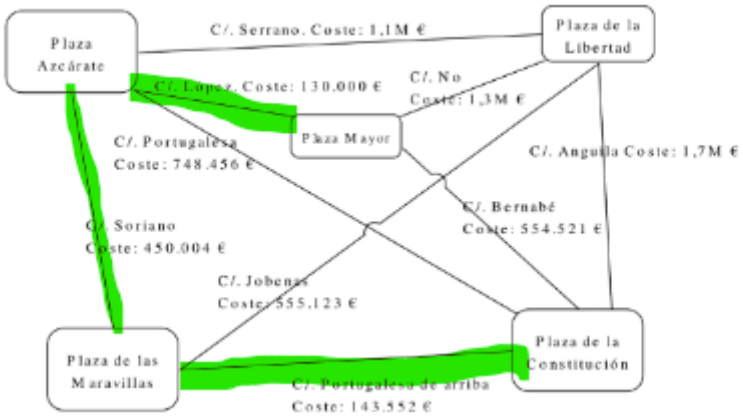
4. se marcaria que la plaza de las maravillas ya ha sido conectada y se empezaria lo mismo con la plaza de la constitución

5. en este caso el coste menor es la c Soriano que conecta a la plaza de las Maravillas con la Azcárate con un coste de 450 004 euros.



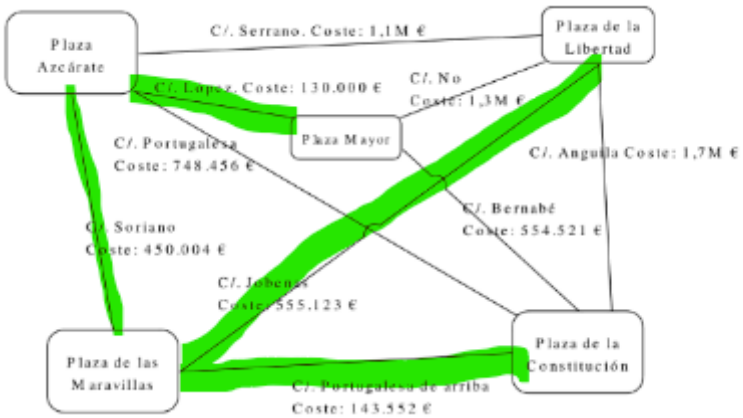
6. Ahora se marcaría la plaza de la Constitución como conectada y se empezaria a comparar a plaza Azcárate con las plazas que faltan por conectar

7. En este caso el coste menor es la c Lopez que conecta a la plaza Mayor con la plaza Azcárate con un coste de 130.000 euros.



8. Ahora se marcaría la plaza Azcárate como conectada y se empezaria a comparar a plaza mayor con las plazas que faltan por conectar

9. como solo falta una se conectaría con esa, la plaza de la Libertad mediante la Calle jovenas con un precio de 555 123 euros



10. Ahora se marcaría la plaza de la Libertad como conectada y como ya estan conectadas todas se acabaria el algoritmo devolviendo el costo que en este caso sería 1.278.679 euros.

Ejemplo de ejecución de P4.