ANÁLISIS

El problema propuesto tiene un conjunto de plazas y calles, que pueden modelarse como un grafo G=(V,A), donde V son los vértices, que se asociarían a las plazas, y A las aristas, que se asociarían a las calles.

El enunciado no indica que las calles tengan direcciones, por lo que podríamos suponer que es no dirigido. Además, por la topología dada en el enunciado, podemos decir que es conexo. A cada calle se le podría asignar una ponderación (el precio de asfaltarla), que sería no negativo.

El objetivo es encontrar el subconjunto de calles a asfaltar de modo que se pueda viajar desde cualquier plaza a cualquier otra plaza mediante caminos entre calles, de modo que el coste total de asfaltar las calles sea mínimo.

Por tanto, se podría decir que el problema es un caso particular del problema del árbol generador minimal, para el que existen dos algoritmos Greedy conocidos óptimos: Prim y Kruskal.

DISEÑO DEL ALGORITMO

Se optará por adaptar el algoritmo de Prim para resolver el problema. La **idea general** sería partir de un nodo (plaza) ya conocido, e ir uniendo este nodo con algún otro de mínimo coste. En iteraciones posteriores, escogeríamos un nodo previamente no utilizado que tuviese una arista a alguno de los nodos ya utilizados de mínimo coste. Se terminaría cuando ya no quedasen nodos por seleccionar. El resultado sería un conjunto de aristas (calles) con peso (coste) mínimo que se deberían asfaltar.

El diseño del algoritmo y sus componentes sería el siguiente:

- **Función objetivo**: Minimizar la suma de los costes de las calles que forman un AGM en la solución.
- **Lista de candidatos:** Los nodos del grafo (plazas).
- **Lista de candidatos utilizados:** Los nodos del grafo (plazas) que ya han sido insertados en la solución.
- **Criterio de selección**: Se escogerá un nodo no utilizado tal que exista una arista (calle) de coste mínimo que una ese nodo con alguno ya utilizado previamente.
- **Criterio de factibilidad**: No se deben formar ciclos. Por cómo se seleccionan los candidatos, este criterio siempre se cumple.
- **Criterio de solución**: El algoritmo terminará cuando no queden nodos (plazas) por seleccionar.

La solución T estará formada por las aristas que han provocado la selección de cada nodo en el criterio de selección.

Con este diseño, adaptamos la plantilla Greedy para resolver el problema de la siguiente forma:

ALGORITMO T= Greedy(G=<V,A>)

```
p = Seleccionar un nodo cualquiera (el primero) de V
B= { p } # Lista de candidatos ya usados
T= ∅ # solución a generar por el algoritmo
```

Mientras |B| ≠ |V| :
    v ← Nodo tal que exista una arista de peso mínimo a=(v,b) a algún nodo de B
    Insertar a en T
    Insertar v en B
Fin-Mientras
Devolver T

DETALLES DE IMPLEMENTACIÓN

En la solución planteada en el código fuente hay varios ficheros:

- **Problema.h / problema.cpp:** Contienen una clase Problema que modela una instancia del problema a resolver. En particular:
  - Se mantiene en memoria el conjunto de nombres de las plazas (campo nombresPlazas). A cada plaza se referirá el algoritmo por su índice dentro de este conjunto (implementado como array).
  - Se mantiene en memoria una matriz con los nombres de calles (campo nombresCalles). Si una entrada nombresCalles[i][j] de esta matriz tiene valor cadena vacía (“”), se entiende que no hay una una calle que una las plazas nombresPlazas[i] y nombresPlazas[j] .
  - Se mantiene en memoria una matriz con los costes de asfaltar cada calle (campo precioCalles). Un valor precioCalles[i][j]= -1 indica que no existe la calle que une nombresPlazas[i] con nombresPlazas[j] y que, por tanto, no se debe asfaltar.
  - Adicionalmente, se implementan varios getters para conocer el número de plazas existentes (getNumPlazas()), el nombre de una plaza(getNombrePlaza(i)), y el nombre/coste de asfaltado de calles (getNombreCalle(i,j) y getPrecioCalle(i,j) respectivamente).
  - Se incorpora un método cargar desde flujo para poder leer una instancia del problema desde fichero.
- **Solucion.h / solucion.cpp:** Contienen una clase Solucion para representar la solución a una instancia del problema (conjunto de aristas/calles a asfaltar junto con su coste). En particular:
  - El campo Aristas implementa un array de aristas como pares de nodos (v,w). Se implementa como una matriz de número de aristas filas por 2. La longitud de este array se guarda en el campo Num.
  - El campo coste guarda el coste completo de la solución.
  - Se tiene un método addArista(origen, destino) para insertar una arista en la solución.
  - Se tiene un método getNumAristas() para obtener el número de aristas que tienen la solución. A cada arista en particular se accede mediante el método getArista(a, i, j), que devuelve por referencia el par de nodos (i,j) de la a-ésima arista insertada en la solución.
  - El coste de la solución se calcula sobre una instancia de un problema p mediante la llamada al método Evaluar(p), y se obtiene posteriormente con el método getCoste().
- **Algoritmo.h / Algoritmo.cpp:** Contiene la definición del algoritmo de Prim, que actúa sobre un problema y devuelve una solución. La lista de candidatos se implementa mediante un array de bool **LC**, indicando con valor LC[i]= true si el candidato ha sido ya seleccionado, y false en otro caso.

Una instancia del problema se proporciona al programa a través de fichero, con el siguiente formato:

Linea 1: Valor N, número de nodos del grafo.

Linea 2→ N+1 : Nombre de las plazas
Lineas N+2→ fin de fichero: Cada línea proporciona información de una calle (plazas que unen, precio y nombre de la calle).

Adicionalmente, se ha construido un fichero makefile para construir el programa de forma simple. Se compila y ejecuta de la siguiente forma:

1. Nos situamos en la carpeta donde está el código fuente y, desde la terminal, escribimos: ***make***
2. Ejecutamos el programa desde la terminal escribiendo: ***./Solucion***
3. Se nos mostrará por consola la solución al problema, junto con su coste.