Ampliación de Programación

Práctica 3

Algoritmos voraces

Sergio García Mondaray

Escuela Superior de Informática de Ciudad Real
Universidad de Castilla-La Mancha

1 Enunciado

En una ciudad el alcalde ha decidido asfaltar sus calles por motivos electorales, pero tiene el problema de que no puede asfaltar todas sus calles por motivos de presupuesto. Ha decidido asfaltar aquellas calles que unan una plaza con otra, quedando todas las calles asfaltadas unidas. Para cada calle y cada plaza sabe cuánto cuesta asfaltarla, y también su tiempo en hacerlo. Además, sabe por estadísticas de elecciones anteriores y encuestas realizadas recientemente, el número de potenciales votantes de su partido tanto para cada calle, como para cada plaza.

El alcalde se ha propuesto maximizar el número de sus partidarios con plaza o calle asfaltadas, puesto que cree que contribuirá a conseguir afianzar el voto entre los seguidores acérrimos, y en el resto, a inclinar definitivamente la balanza a su favor. Diseñe una Heurística Voraz que indique cuáles son las calles y plazas a asfaltar, supuesto que disponemos de un presupuesto P.

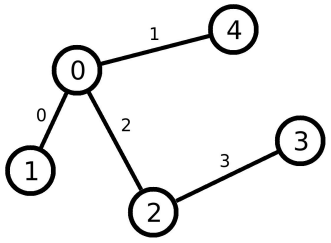
2 Planteamiento y resolución del problema

2.1 Representación del pueblo

Para representar las calles y plazas del pueblo de nuestro enunciado emplearemos una estructura de grafo. En nuestro caso, hemos optado por un grafo estático, representado por una matriz bidimensional con las siguientes características:

1. Las filas de la matriz son las calles del pueblo.
2. Las columnas representan las plazas.
3. Un 1 en la posición [i,j] significa que la calle i conecta con la plaza j.

Por ejemplo, el diagrama y la tabla siguientes representan un grafo equivalente:



	0	1	2	3	4
0	1	1	0	0	0
1	1	0	0	0	1
2	1	0	1	0	0
3	0	0	1	1	0

2.2 Nuestra heurística

Para optimizar la elección de plazas y calles a asfaltar, en función de las características descritas por el enunciado, definiremos una heurística voraz según los siguientes elementos.

1. **Lista de candidatos:** En cada estado, los candidatos serán las plazas y calles sin asfaltar que están en contacto directo con las ya asfaltadas.
2. **Conjunto de decisiones ya tomadas:** Se irá componiendo por las plazas y calles asfaltadas.
3. **Función selección:** Para seleccionar el mejor candidato, nos fijaremos en el número de votantes y en el coste de asfaltado. Más concretamente definiremos un ratio de comparación votantes/coste.

4. **Función completable:** No se añadirán al conjunto de seleccionados aquellos candidatos (plazas y calles) que, o bien tengan un ratio bajo, o bien no podamos permitírnoslos por su elevado coste.
5. **Función solución:** Cuando todos los candidatos sean asfaltados, o bien cuando nuestro presupuesto se termine, habremos finalizado.
6. **Función objetivo:** Será la suma de los ratios individuales lo que caracterice lo buena o mala que es nuestra solución.

En resumidas cuentas, lo que nuestro algoritmo hará será, al principio, encontrar la mejor opción a asfaltar (según el ratio votantes/coste) de entre todas las plazas y calles. La añadirá a la solución y la marcará como asfaltada. Después añadirá todos sus elementos adyacentes al conjunto de candidatos (si era una calle, este conjunto de adyacentes serán 2 plazas; si era una plaza, el conjunto de adyacentes estará compuesto por todas las calles que salen de dicha plaza). Elegirá el mejor candidato según el ratio ya mencionado, si nos lo podemos permitir lo asfaltamos y añadimos sus adyacentes al conjunto de candidatos (siempre que no estén ya), en caso de no poder permitirnoslo lo marcaremos para no volver a considerarlo y seguiremos buscando el mejor candidato de entre los restantes. Esto se repetirá hasta que se termine nuestro presupuesto o hayamos asfaltado todo el pueblo.

2.3 Algoritmo voraz

Según la heurística definida en el anterior apartado, nuestro algoritmo voraz es:

```
INICIO
    HAZ
        SI elegidos = 0 //Situacion inicial
            aux <- mejorCalleOPlaza
            SI aux.precio < presupuesto
                asfaltar(aux)
                presupuesto = presupuesto - aux.precio
                anadir aux a solucion
                anadir adyacentes(aux) a candidatos
                elegidos = elegidos + 1
            SI NO
                marcar como no asequible para no volver a
                   seleccionarlo
        SI NO //Cualquier otro momento
            SI aun quedan candidatos
                aux <- mejorCandidato
                SI aux.precio < presupuesto
                    asfaltar(aux)
                    presupuesto = presupuesto - aux.precio
                    anadir aux a solucion
                    anadir adyacentes(aux) a candidatos
                    elegidos = elegidos + 1
                SI NO
                    marcar como no asequible para no volver a
                       seleccionarlo
            SI NO //Ya no quedan candidatos
                fin = SI
        FIN SI
    MIENTRAS (fin = NO)
FIN
```

2.4 Nota sobre la implementación

Al estudiar el código fuente se puede apreciar que el vector solución mencionado en el pseudocódigo (vector que contiene las calles y plazas ya asfaltadas) no es creado en ningún momento, sino que se emplea como parte del vector de candidatos. Lo explicaré con más incapié: el vector de candidatos contiene todas las plazas y calles que han sido añadidas como adyacentes en alguna iteración. Cada elemento del vector (calle o plaza) contiene un campo que distingue entre 3 posibles estados: *asfaltado*, *demasiado caro*, y *sin comprobar*. De esta manera, al finalizar el algoritmo todos esos elementos van a estar en uno de los dos primeros estados. Así, hallar la solución final será tan sencillo como obtener los elementos de dicho vector que tengan estado *asfaltado*.

3 Ejemplo práctico

Con el fin de ilustrar el comportamiento de nuestro algoritmo voraz, expondré un ejemplo generado aleatoriamente y resuelto por el propio programa que he implementado. La tabla generada por la aplicación es la siguiente:

	0	1	2	3	4	5	6
0	0	0	1	0	1	0	0
1	0	0	1	0	0	1	0
2	1	0	0	0	1	0	0
3	0	0	1	1	0	0	0
4	0	1	0	0	0	0	1
5	0	1	0	0	0	1	0

Y los datos sobre las calles y plazas (densidad hace referencia al ratio votantes/coste):

```
    PLAZAS:

Coste: 1518, votantes:  565, estado: 0, densidad: 0.372200, indice: 0
Coste: 1210, votantes: 2970, estado: 0, densidad: 2.454545, indice: 1
Coste: 5466, votantes: 2355, estado: 0, densidad: 0.430845, indice: 2
Coste: 5498, votantes: 1393, estado: 0, densidad: 0.253365, indice: 3
Coste: 2314, votantes: 2211, estado: 0, densidad: 0.955488, indice: 4
Coste: 1700, votantes:  280, estado: 0, densidad: 0.164706, indice: 5
Coste: 5908, votantes: 1401, estado: 0, densidad: 0.237136, indice: 6

    CALLES:

Coste: 5939, votantes: 2135, estado: 0, densidad: 0.359488, indice: 0
Coste: 1580, votantes: 2390, estado: 0, densidad: 1.512658, indice: 1
Coste: 2075, votantes: 1767, estado: 0, densidad: 0.851566, indice: 2
Coste: 3688, votantes: 1912, estado: 0, densidad: 0.518438, indice: 3
Coste: 2608, votantes: 2178, estado: 0, densidad: 0.835123, indice: 4
Coste: 1926, votantes:  249, estado: 0, densidad: 0.129283, indice: 5
```

La solución dada por el programa es la siguiente (a continuación procederemos a realizar a mano el proceso que define nuestra heurística, con el fin de comprobar su corrección):

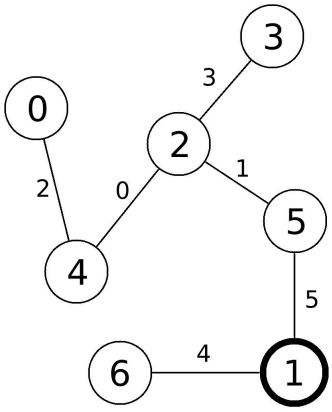
```
PRESUPUESTO INICIAL: 20000

CALLES y PLAZAS ASFALTADAS:
    Plaza  1, coste:  1210
    Calle  4, coste:  2608
    Calle  5, coste:  1926
    Plaza  6, coste:  5908
    Plaza  5, coste:  1700
    Calle  1, coste:  1580

PRESUPUESTO RESTANTE: 5068
```

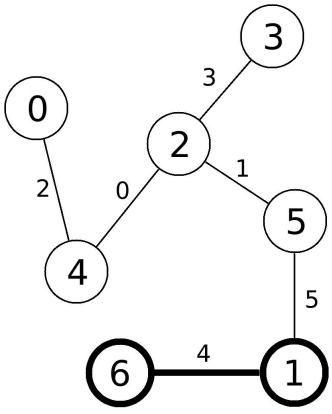
3.1 Paso a paso

En primer lugar ilustremos el grafo en forma de diagrama, con el fin de facilitar su comprensión:



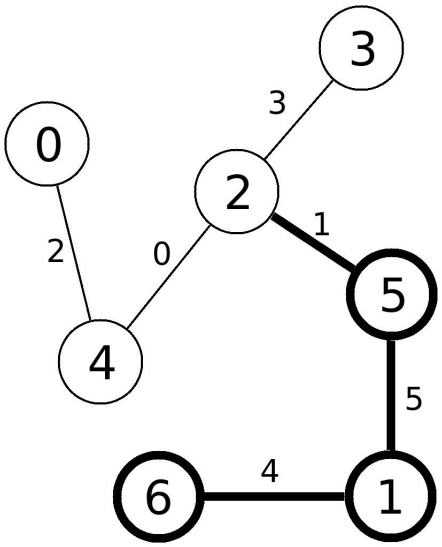
Los números representan los índices de las plazas y calles. Como ya se mencionó en los apartados 2.2 y 2.3, lo primero que hace nuestro algoritmo es encontrar la mejor calle o plaza atendiendo al ratio votantes/coste. En nuestro caso, y fijándonos en la información generada sobre las plazas y calles, ese mejor elemento del que partir es la **plaza 1**. Como su coste es menor que nuestro presupuesto y, por lo tanto, podemos permitirnos asfaltarla, lo hacemos –dejando nuestro presupuesto en 18790–. Nuestro algoritmo añade ahora a los candidatos las calles 4 y 5, que salen de la plaza 1.

De entre los candidatos (calle 4 y calle 5) la mejor opción es la **calle 4**. La asfaltaremos, ya que podemos costearlo, por tanto añadimos la plaza 6 como candidato –nuestro presupuesto ahora es de 16182–. En este instante, los candidatos son la plaza 6 y la calle 5; el mejor candidato es la **plaza 6** y, como nuestro presupuesto es bastante, la asfaltamos. Al ser la calle 4 (ya asfaltada), la única adyacente de la plaza que acabamos de asfaltar, no añadimos nuevos candidatos. En este instante, nuestro presupuesto es de 10274, y la situación es la siguiente:



Podemos observar que ahora el único candidato posible es la **calle 5**, por tanto, como nuestro presupuesto es suficiente la asfaltamos, añadiendo como nuevo candidato, y en este instante el único, a la plaza 5. Nuestro presupuesto tras asfaltar la calle 5 es de 8348, como es suficiente asfaltamos la **plaza 5**, dejando nuestro presupuesto en 6648. Tras asfaltar la plaza 5, añadimos la calle 1 como candidato. La búsqueda del mejor candidato en este momento nos devolverá la **calle 1**, por tanto, la asfaltaremos; así tendremos que añadir como candidato a la plaza 2. Nuestro presupuesto ahora será de 5068.

Como nuestro presupuesto no es suficiente para asfaltar ya ninguno de los candidatos, el algoritmo finaliza, dejando la siguiente situación (remarcadas en negro las plazas y calles asfaltadas):



4 Tiempo de trabajo y valoración de dificultad

Aproximadamente, he invertido unas 10 horas resolviendo esta práctica y unas 2 horas redactando la memoria. La valoración de dificultad total que le asigno a esta tarea es de 8 sobre 10, ya no únicamente por el diseño e implementación de la heurística voraz (núcleo de la práctica), sino también por los problemas surgidos al tener que trabajar por primera vez con grafos en lenguaje C.