Gestión Compartida de Recursos

Javier Larrosa*

Departament de Ciències de la Computació
larrosa@cs.upc.edu

Resumen

Con este proyecto se pretende que experimentéis el uso de técnicas algorítmicas vistas en clase en problemas reales. En esta ocasión nos centraremos en algoritmos de max-flow *para resolver problemas de gestión de recursos. En particular, la* asignación justa de responsables en la gestión compartida de un recurso.

La práctica se hará en grupos de 2 o 3 personas. La composición de los grupos se deberá comunicar a larrosa@cs.upc.edu antes del 15 de mayo.

La entrega de la práctica se hará on-line (hasta el 8 de Junio) a través del racó. Tras la corrección de la entrega on-line, algunos grupos pueden ser convocados para una entrevista personal (en junio, fecha por decidir) con prueba interactiva. Algunos grupos pueden ser convocados a la entrevista para aclarar dudas relativas a su práctica. Otros grupos pueden ser elegidos aleatoriamente. **Es obligatorio que en la entrevista estén todos los miembros del grupo**.

I. Descripción del Problema

I. Introducción

En los últimos años, y gracias al desarrollo tecnológico, se han desarrollado muchos modelos para que un grupo de usuarios compartan recursos de manera ágil, flexible y eficiente. Algunos ejemplos de este tipo de modelos aparecen en el campo del transporte (*carsharing*, *bikesharing*, *dial-a-ride*, etc), consumo (*cooperativas*, *huertos compartidos*...) , o incluso en el uso de recursos informáticos (*servidores virtuales*, *foros autogestionados*,...).

El éxito de estos nuevos paradigmas radica en la flexibilidad en el uso que tienen los usuarios. Diferentes usuarios pueden tener diferentes perfiles de uso. Incluso un mismo usuario puede hacer usos muy diferentes a lo largo del tiempo. Esta flexibilidad tiene que quedar reflejada en el coste y en la responsabilidad que cada usuario asume sobre el bien común. En este proyecto nos centramos en sistemas autogestionados donde no hay una autoridad que regula el buen uso, sino que esta responsabilidad se reparte entre los usuarios. En particular, nos interesa el caso en el que hay un recurso que se usa simultaneamente por muchos usuarios, pero donde siempre tiene que haber uno de los usuarios que asuma un rol especial (*responsable*, *moderador*, etc).

II. Asignación Justa

Para centrar la discusión, consideraremos un ejemplo concreto. Un grupo de amigos alquilan entre todos una embarcación de recreo para salir juntos a navegar. Cada vez que el barco se va a usar hace falta que uno de los usuarios se encargue de prepararlo antes del viaje (puesta en marcha), asuma la conducción durante el viaje, y lo atraque dejandolo listo hasta el siguiente uso. También tendría que hacerse responsable de la limpieza y reparación de desperfectos aparecidos durante el viaje. A esta persona le llamaremos el *responsable* del viaje. A nadie le apetece ser responsable y

*La versión más actualizada de este documento, así como cualquier material adicional relacionado con la práctica se encuentra en `http://algorithmics.cs.upc.edu/`.

nuestra tarea es hacer una *asignación justa* respecto al uso que hagan cada una de las personas. En cada viaje tiene que haber un responsable, que será el mismo durante todo el viaje.

Hay dos elementos que parecen razonables a la hora de definir una asignación justa. Si una persona usa el barco muchas veces, tambien debería ser responsable muchas veces. Por otra parte, si una persona usa el barco en días en los que pocas personas quieren usarlo, esto también debería incrementar el número de veces que es responsable. Si no ves claro este segundo criterio, piensa en el caso de que 2 personas viajan 20 veces juntas y con nadie más. Es lógico que sean responsables 10 veces cada una. Si por otro lado hay 20 personas que viajan 20 veces juntas, y con nadie más, es lógico que les toque ser responsables una vez a cada una.

Supongamos que una persona viaja r veces. En su primer viaje hay k_1 pasajeros, en el segundo k_2, y así hasta k_r. Parece razonable que a esta persona le toque ser responsable $S = \sum_{j=1}^{r} \frac{1}{k_j}$ veces. Como este número puede no ser entero, nos conformaremos con que sea responsable $\lfloor S \rfloor$ o $\lceil S \rceil$ veces. Si esto ocurre para todas las personas, diremos que la asignación es *justa*.

Considera el siguiente ejemplo con 4 personas y 5 viajes, donde una equis significa que va a participar en el viaje. Para la primera persona $O = \frac{1}{3} + \frac{1}{3} + \frac{1}{4}$

Las dos columnas de la derecha indican el numero de veces que le puede tocar ser responsable a cada persona de acuerdo con la definición anterior. Es fácil ver que existen varias asignaciones justas. Una de ellas es la que indican las equis mayúsculas.

Personas	Día 0	Día 1	Día 2	Día 3	Día 4	$\lfloor S \rfloor$	$\lceil S \rceil$
0	X	x	x			0	1
1	x		X			0	1
2	x	X	x	x	x	1	2
3		x	x	X	X	1	2

II. Problema a Resolver

El problema que teneis que resolver es el siguiente. Sean n personas y m viajes. Tenemos una matriz $A(n \times m)$ tal que $A(i, j)$ se refiere a la participación de la persona i en el viaje j. Si $A(i, j) = 0$ significa que esa persona no va a participar en ese viaje. Si $A(i, j) = 1$ significa que sí va a participar pero que en esa ocasión, a ser posible, no quiere ser responsable (preferencia negativa). Si $A(i, j) = 2$, entonces sí va a participar y no tiene inconveniente en ser responsable (preferencia neutra). Finalmente, si $A(i, j) = 3$ significa que sí va a participar y que en esa ocasión, a ser posible, le gustaría ser responsable (preferencia positiva).

Teneis que hacer un programa en C++ que, dada una matriz A, encuentre, si la hay, una asignacion de responsables justa y que además satisfaga todas las preferencias de todas las personas. Si no la hay, buscará una en la que se satisfagan las preferencias negativas. De no haber tal solución, buscará una que no tenga en cuenta ninguna preferencia. Si no hay ninguna solución justa, simplemente dará una respuesta negativa.

Por su naturaleza, las dimensiones del problema que hemos usado de ejemplo serán pequeñas, pero en el proyecto tenéis que pensar que el mismo modelo se puede aplicar a casos con instancias muy grandes como por ejemplo decenas de miles de usuarios que usan un servicio web y siempre tiene que haber uno que se encargue de resolver las posibles incidencias.

Internamente, el programa calculará la solución usando un grafo de maximización de flujos. El programa tiene que incluir como mínimo el algoritmo de *Shortest Augmentation Path* [Edmons and Karp, 1972]. Se valorara positivamente que también se incluya el algoritmo de *Preflow-Push* ([Kleinberg and Tardos, 2005], Sección 7.4, o [Cormen and Leiserton and Rivest, 1990] Sección 27.4)

El programa tiene que funcionar con los formatos de entrada y salida que se describen a continuación. Para que podáis hacer experimentos, se os facilitara un conjunto de instancias en el formato apropiado.

No se puede utilizar ninguna librería que no sea estándar.

I. Formato de Entrada y de Salida

El programa tiene que leer instancias del problema dadas en ficheros de texto. La primera línea tendrá dos enteros: el numero de personas n i el número de viajes m. Seguirán n líneas, con m enteros (entre 0 y 3) cada una que representarán la matriz A indicada. Es decir, el ejemplo anterior se podría corresponder con el siguiente fichero:

```
4 5
2 2 1 0 0
1 0 3 0 0
2 2 1 3 2
0 2 1 1 3
```

La salida del programa tiene que ser un fichero de texto con dos lineas. La primera linea del fichero nos indicará el tipo de solución encontrada. Una A indicará que se han podido satisfacer todas las preferencias, una B indicará que sólo se han podido satisfacer las preferencias negativas, una C indicará que se ha encontrado una asignación justa pero que incumple preferencias positivas y negativas. Finalmente, una D indicará que no se ha encontrado ninguna asignación justa. La segunda línea contendrá m números, uno por viaje. El número en la j-esima posicion indicará la persona responsable de ese viaje.

Una salida posible para el ejemplo de entrada anterior seria:

```
A
0 2 1 2 3
```

III. Qué hay que entregar

Tenéis que entregar una carpeta con lo siguiente:

- Un informe (en pdf) que incluya:
  - Descripción que cómo habeis modelado el problema y cómo funciona vuestro algoritmo (sin entrar en detalles sobre el algoritmo de flujos). Puede ser útil algun dibujo de cómo son las redes de flujos sobre las que calculais flujos máximos.
  - Para cada algoritmo que hayáis implementado (como mínimo Edmons-Karp), el coste temporal del algoritmo en función de los parámetros del problema (numero de personas, número de viajes, etc)
  - Tablas de resultados experimentales sobre las instancias que se os han facilitado para cada versión del problema que hayais considerado. Como veréis, los nombres de las instancias tienen 5 parámetros: el número de personas n, el numero de viajes m, la densidad d que indica cómo de probable es que una persona quiera ir a un viaje, el nivel de preferencias p que indica cómo de probable es que una persona tenga preferencia sobre un viaje, y la semilla aleatoria s usada para generar la instancia. Para cada tipo de problemas (n, m, d, p)se han generado 10 instancias. La tabla de resultados tiene que incluir, para cada tipo de problemas, y para cada algoritmo de max-flow que hayáis implementado, el tiempo medio (sobre las 10 instancias) de CPU necesario para resolverlos.
  - Lista de referencias bibliográficas y recursos on-line que hayáis consultado durante la elaboración de la práctica.
- Una carpeta con todas las fuentes necesarias para compilar y ejecutar vuestro(s) programa(s) en un entorno Linux. Se deben incluir instrucciones para la compilación y ejecución en entorno linux.

- Un fichero de texto "Resultado.txt" que contenga en orden alfabético el nombre de las instancias seguidas de la letra de la solución encontrada (A, B, C o D). Tenéis que ser muy estrictos con el formato de este fichero, pues lo usaremos para comprobar la corrección de los resultados que obtienen vuestros programas haciendo un "diff" con nuestra solución.

Referencias

[Kleinberg and Tardos, 2005] Kleinberg, J. and Tardos, E. (2009). Algorithm Design.

[Edmons and Karp, 1972] Edmonds, J. and Karp, R.M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM 19, 2 (1972), 248–264.

[Cormen and Leiserton and Rivest, 1990] Cormen, T. and Leiserton, C. and Rivest, R. (1990). Introduction to Algorithms