Cálculo del Flujo Máximo en una red

TTPS - Opción C

Facundo Miglierini

Índice

1. Introducción

En este informe explicaré en qué consiste el problema del cálculo del flujo máximo en una red. Para comprender este asunto con mayor claridad, mencionaré algunos conceptos que permitirán asentar las bases sobre el tema, tales como qué son las redes de flujo y algunas de sus características (por ejemplo, la capacidad y el flujo de un eje). Luego, expondré los algoritmos existentes para solventar el problema sobre el que se basa este trabajo y haré una comparativa sobre los mismos respecto de su eficiencia.

Para finalizar, describiré los problemas elegidos y la solución implementada para cada uno de ellos, junto con una breve conclusión.

2. Redes de flujo

En esta sección definiré en qué consiste una red de flujo y explicaré a qué se refiere el concepto de flujo máximo.

2.1. Definición

Una red de flujo es un grafo dirigido donde los ejes transportan algún tipo de flujo (por ejemplo agua, gas o autos, donde la unidad de medida a utilizar varía en función de cada tipo de flujo particular) y los vértices actúan como conmutador de tráfico entre los diferentes ejes (tales como las conexiones en las cañerías o las intersecciones en las rutas). En este tipo de grafos existen dos vértices especiales, uno de ellos es el vértice fuente, el cual genera el tráfico saliente. Por otro lado, se encuentra el vértice sumidero, quien absorbe el tráfico entrante.

A continuación expondré dos atributos importantes en este tipo de redes: la capacidad y el flujo de un eje.

La capacidad de un eje indica la cantidad máxima que un eje puede transportar, en las unidades relacionadas al problema que se esté tratando (litros, m^3, número de autos, etc).

Por otra parte, el flujo es la cantidad transportada por un eje en un momento dado. Para asegurar la correctitud de la red, debe asegurarse que el flujo de un eje siempre sea un valor no negativo menor o igual a su capacidad.

Para cerrar con este apartado, daré una definición formal de una red de flujo:

Sea G=(V,E) un grafo dirigido con V vértices y E ejes,

$\forall$ e $\in$ E, existe un único s $\in$ V llamado fuente (source) y un único t $\in$ V llamado sumidero (sink).

El resto de los vértices son internos.

Cada arco $(u, v) \in$ E tiene una capacidad no negativa $c(u, v) \geq 0$

Un flujo en G es una función $f : V \times V \mapsto R$ tal que $\forall\ u, v \in V$, $f(u, v) \leq c(u, v)$

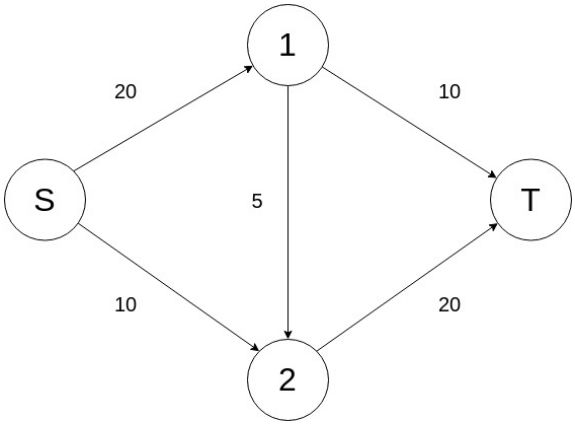

Figura 1.1: Ejemplo de una red de flujo con fuente S y sumidero T

2.2. Flujo máximo

El flujo máximo en una red de flujo consiste en la tasa mayor a la cual el material que se transporta en la red puede ser transportado desde la fuente al sumidero sin violar ninguna restricción de capacidad.

Cabe destacar que el cálculo del flujo máximo de una red se encuentra íntimamente relacionado con el corte mínimo de la red (Min Cut), el cual indica el peso total (mínimo) que necesitamos desconectar para que un grafo deje de estar conectado. El flujo máximo y el corte mínimo tienen el mismo valor.

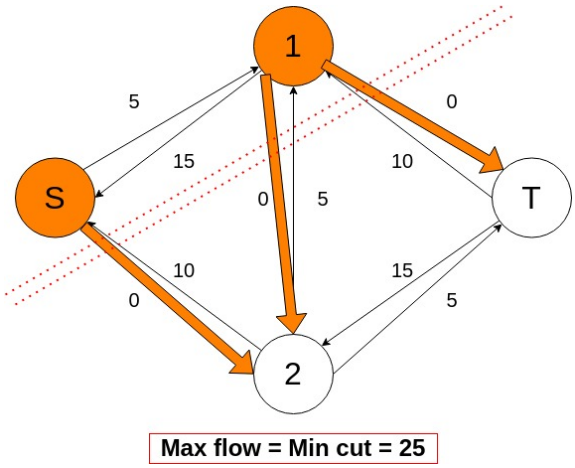

Figura 1.2: Flujo máximo de la red de flujo expuesta en la figura 1.1, donde el corte mínimo es igual a la suma del peso inicial de las aristas resaltadas

Una vez aclarados estos conceptos, podemos continuar con el tema central del informe.

3. Algoritmos existentes para el cálculo del flujo máximo en una red

Existen distintos algoritmos para calcular el flujo máximo de una red, los cuales utilizan un mismo mecanismo de cálculo, pero difieren en la forma en la que recorren el grafo. Al finalizar la explicación de las distintas variantes haré una comparativa sobre la eficiencia de las mismas.

3.1. Método Ford-Fulkerson

El método Ford-Fulkerson es un algoritmo iterativo que busca repetidamente caminos de aumento, los cuales son caminos desde la fuente s hasta el sumidero t que pasan a través de ejes con peso positivo en el grafo residual. Un grafo residual inicialmente es igual al grafo original. Si un eje del mismo es usado en un camino de aumento y un flujo pasa por éste con un peso f menor a su capacidad, entonces la capacidad restante (o residual) del eje va a ser decrementada en f, mientras que la capacidad residual del eje reverso será incrementada en f. Una vez que se tiene un camino de aumento que posee a g como el peso mínimo de un eje en el camino encontrado (el cuello de botella en este camino), el algoritmo de Ford-Fulkerson realizará una serie de pasos: decrementar/incrementar en g la capacidad de los ejes forward/backward de todo el camino de aumento respectivamente, y sumar g al valor de flujo máximo general, el cual va acumulando el peso mínimo que se obtiene en el camino de aumento de cada iteración.

La capacidad de los ejes forward es decrementada para que se reduzca la capacidad restante de los arcos usados en el camino de aumento encontrado. Por otro lado, se incrementa la capacidad de los arcos backward para permitir que las iteraciones futuras cancelen parte de la capacidad usada por los ejes forward que fue utilizada incorrectamente por algunos flujos previos.

Cabe destacar que el método Ford-Fulkerson usa DFS (Depth First Search) para computar el valor de flujo máximo.

3.2. Algoritmo Edmonds-Karp

Como mencioné previamente, las distintas variantes en el cálculo del flujo máximo de una red implementan el mismo mecanismo (decrementar el peso mínimo del camino de aumento a los ejes forward de todo el camino e incrementarlo en los ejes backward). La única diferencia que presenta este algoritmo respecto del método Ford-Fulkerson consiste en que en lugar de usar DFS para recorrer el grafo, realiza el recorrido mediante BFS (Breadth First Search), lo cual lo vuelve más eficiente que su antecesor debido a que se realizan menos iteraciones. Esto es así porque en cada iteración el camino encontrado es el camino más corto que tiene capacidad disponible, y está demostrado que si uno elige el camino creciente más corto con cada paso, entonces la longitud de los caminos crecientes está aumentando en el sentido amplio[1].

3.3. Algoritmo de Dinic

Este algoritmo funciona de manera similar a la implementación de Edmonds-Karp, pero existe una diferencia en la forma en la que se usa BFS en ambas técnicas.

Mientras que en la variante expuesta previamente se usa BFS para buscar un camino de aumento y enviar el flujo en este camino, en el método de Dinic se usa para verificar si es posible enviar más flujo y para construir un grafo de niveles, donde se asigna a cada vértice un nivel que representa la distancia más corta (medida en cantidad de ejes) desde la fuente hasta sí mismo. Una vez construido este grafo, se envían múltiples flujos. Debido a esto, funciona mejor que Edmonds-Karp, donde se envía un único flujo por cada camino de aumento encontrado.

Además, este método usa el concepto de flujo de bloqueo. Un flujo es de bloqueo si no puede enviarse más flujo usando el grafo de niveles (es decir, utilizando un camino cuyos vértices tengan los niveles 0, 1, 2... de forma ordenada). Una vez que es alcanzado un flujo de bloqueo, se puede decir que se obtuvo el flujo máximo de la red, con una fuente y un sumidero dados.

Dado que se realiza una menor cantidad de iteraciones, esta variante presenta una complejidad de tiempo en el peor caso que es teóricamente mejor que el algoritmo de Edmonds-Karp.

[1] Para conocer la demostración sobre la complejidad del algoritmo Edmonds-Karp, véase `https://brilliant.org/wiki/edmonds-karp-algorithm/#complexity-proof`

4. Comparativa de los algoritmos expuestos

Una vez enunciados los distintos métodos para obtener el cálculo del flujo máximo de una red de flujo, vamos a comparar los tiempos de ejecución que poseen las distintas alternativas.

En primer lugar, el método Ford-Fulkerson podría ejecutarse con un orden O(mf x E), donde mf es el valor de flujo máximo, y E el número de ejes del grafo.

Por otro lado, el algoritmo de Edmonds-Karp tiene una cota superior de $O(VE^2)$, lo cual es más eficiente que el método previo.

Por último, el algoritmo de Dinic tiene un tiempo de ejecución de $O(V^2E)$. Dado que un grafo de flujo típico tiene $V < E$ y $E << V^2$, la complejidad temporal del peor caso es teóricamente mejor que el algoritmo de Edmonds-Karp.

Para concluir este apartado, podemos decir que, si bien se distinguen diferencias notables en cuanto al tiempo de ejecución del primer método respecto de los últimos dos, no existen diferencias sustanciales entre las dos últimas alternativas. En la mayoría de los casos, la utilización del algoritmo de Edmonds-Karp será más que suficiente para resolver el problema del flujo máximo en un tiempo razonable. Sin embargo, trasladando el problema a su aplicación en ejercicios de programación competitiva, existen algunas ocasiones particulares donde el algoritmo de Edmonds-Karp recibe como resultado un aviso de TLE (Time Limit Exceeded), mientras que el algoritmo de Dinic recibe un AC (Accepted) con el mismo grafo de flujo. Por lo tanto, la elección de un algoritmo u otro dependerá de la red con la que se debe trabajar, y la complejidad que presente la implementación de la solución a utilizar.

5. Problemas elegidos

A continuación enunciaré de forma breve en qué consisten los ejercicios que seleccioné para este trabajo, junto con las soluciones implementadas por mí y aceptadas por el juez en línea UVa Online Judge.

5.1. Internet Bandwidth

Este ejercicio trata sobre un conjunto de máquinas interconectadas, las cuales pueden disponer de varias conexiones entre dos mismas computadoras, donde cada conexión tiene un ancho de banda que se refiere a la máxima cantidad de datos por unidad de tiempo que puede ser transmitida desde un nodo a otro. Además, gracias a la conmutación de paquetes es posible transmitir los datos por muchos caminos a la vez. Cabe destacar que el ancho de banda de una conexión es el mismo en ambas direcciones.

Se solicita que, dada una topología de red ingresada por teclado, se calcule el ancho de banda total desde un origen hacia un destino dados.

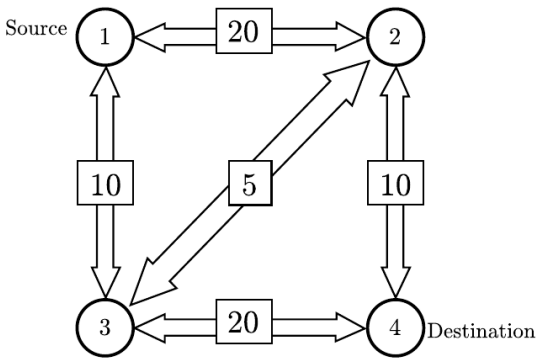

Figura 2: Ejemplo de una red de flujo válida que aparece en el enunciado.

Para resolver este problema, implementé una matriz de adyacencias donde cada componente $[i, j]$ de la misma almacena el peso acumulado de todas las conexiones que van desde la computadora i hacia la computadora j. Luego, realicé el cálculo del flujo máximo a partir de un origen y destino indicados por la entrada mediante el algoritmo de Edmonds-Karp, ya que considero que es fácil de implementar y resuelve el problema en un tiempo óptimo. Una vez obtenido el valor, muestro el resultado como salida del programa.

5.2. My T-shirt suits me

El enunciado de este problema explica que se deben repartir N remeras a M personas, y siempre se dará que $N \geq M$. Además, existe una clasificación con seis talles diferentes, por lo que N debe ser múltiplo de seis, de manera que exista la misma cantidad de remeras para cada talle. Cada persona puede usar únicamente dos talles particulares.

Se pide escribir un programa que reciba N y M por teclado, junto con los talles que puede usar cada persona, y que se retorne como salida si existe una forma de distribuir las remeras de manera tal que cada persona pueda obtener una remera que le quede bien.

Considero que la principal dificultad que se me presentó en este ejercicio fue el planteo de un grafo que me permitiese realizar el cálculo del flujo máximo desde un origen hasta un destino para poder dar con el resultado. Finalmente, opté por implementar una red de flujo como la que se visualiza en la siguiente imagen.

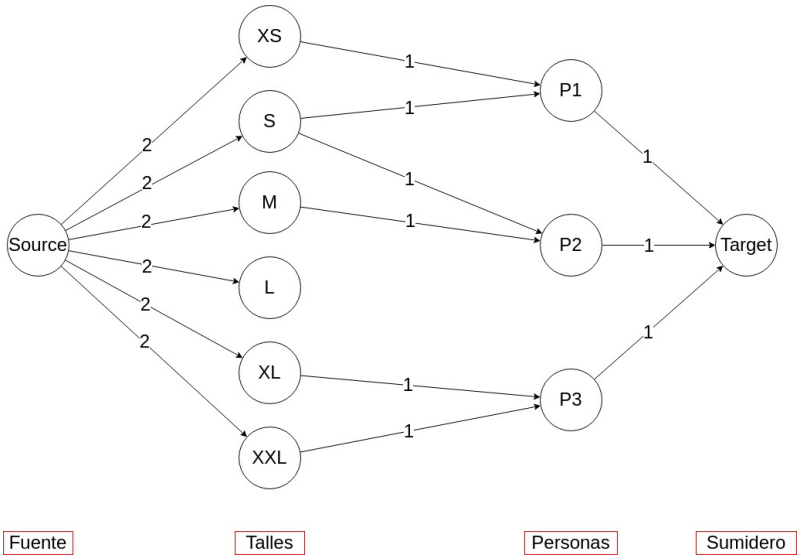

Figura 3: Red de flujo que comienza en el nodo "Source", el cual se encuentra conectado por ejes con capacidad 2 a cada uno de los talles posibles (en este caso, N = 12), y los mismos se encuentran conectados con capacidad 1 a cada persona que pueda usar dicho talle. Finalmente, todas las personas se conectan con capacidad 1 a un vértice "Target".

Una vez que dí con el grafo adecuado, solo restaba implementar un algoritmo que procese el flujo máximo de la red. Dado que N podía tener un valor no mayor a 36, y M podía alcanzar un valor máximo de 30, desarrollé una matriz de tamaño $(6 + M + 2) * (6 + M + 2)$ (se reservan seis filas y columnas para los nodos que representan talles, M para las personas, y las dos componentes adicionales permiten almacenar los nodos origen y destino) sin que existan costos significantes en términos de memoria. En la fila del nodo fuente almacené la cantidad de remeras disponibles para cada talle (con valor $N/6$ para cada nodo de talle). Por otro lado, en la fila de cada talle, almacené el valor 1 para aquellas personas que pudiesen usar remeras de dicho tamaño. Por último, la fila de cada persona almacena un arco hacia el vértice sumidero con capacidad 1.

Finalmente, apliqué el algoritmo de Edmonds Karp para calcular el flujo máximo de la red. Si el resultado obtenido es mayor o igual a N, entonces el programa devuelve la salida "YES". En caso de que el valor sea menor, retorna "NO". La lógica de este procesamiento se basa en que si todas las personas pueden enviar flujo desde sí mismas hacia el nodo "Target"de forma simultánea, es porque cada una recibió una remera que puede utilizar.

5.3. Collectors Problem

Este problema nos cuenta que existe un grupo de amigos que colecciona stickers que se obtienen de paquetes de chocolates. Dado que suelen tener stickers repetidos, los intercambian por stickers que no tienen. Bob, el personaje principal del ejercicio, se percata de que en algunas ocasiones le conviene intercambiar un sticker duplicado por otro que ya tiene. Asumiendo que los amigos de Bob solamente intercambian stickers con él, y que solo lo harán con stickers que tienen repetidos por otros que ellos no tienen, se pide ayudar a Bob y decirle la cantidad máxima de stickers diferentes que puede obtener intercambiando stickers con sus amigos.

Al igual que en el ejercicio previo, la dificultad principal que se presenta se basa en el planteo de una red de flujo apropiada para lograr resolver el problema. En este caso, se me ocurrió que el vértice fuente podría ser Bob, y que cada amigo y cada sticker sea un vértice interno del grafo. Además, agregué un eje desde cada persona hacia todos los stickers que esta persona posee, donde la capacidad del mismo es igual a la cantidad de veces que tiene ese sticker, restándole el valor 1 (solo se intercambian stickers duplicados). Por otro lado, agregué un eje con capacidad 1 desde cada sticker hacia cada persona que no tenga dicho sticker (a excepción de Bob, el cual es el origen de la red). Por último, añadí un eje con capacidad 1 desde cada sticker hacia un vértice target.

A continuación se muestra un grafo de ejemplo que representa lo mencionado previamente.

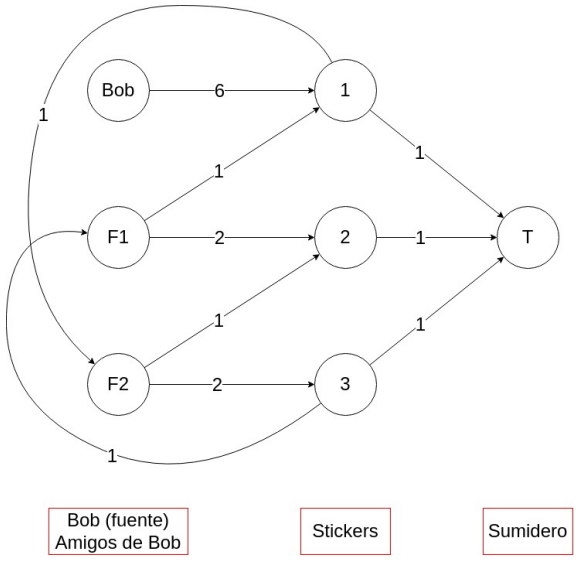

Figura 4: Red de flujo que comienza en el nodo "Bob". Tanto él como sus amigos están conectados a los vértices de stickers, con una capacidad equivalente a la cantidad de veces repetidas que se tiene un sticker decrementada en 1. Por otra parte, hay un arco con capacidad 1 desde cada sticker hacia cada amigo de Bob que no disponga del mismo. Finalmente, todos los stickers se conectan con capacidad 1 a un vértice sumidero.

Luego, dado que la cantidad de amigos y la cantidad de stickers se encuentra acotada, definí una matriz de adyacencias de tamaño $n + m + 1$ (entre las n personas se encuentra Bob que es el origen, el valor m comprende la cantidad de stickers, y se agrega un vértice que funciona como sumidero), mediante la cual implementé el algoritmo de Edmonds-Karp para obtener el flujo máximo en la red, cuyo valor indica la cantidad máxima de stickers diferentes que puede obtener Bob.

6. Conclusión

En este informe se explicó en qué consisten las redes de flujo, y algunos conceptos asociados tales como el flujo de los ejes y su capacidad.

Luego, se enunció el problema del flujo máximo en una red, y se expusieron los distintos algoritmos existentes para resolverlo. Posteriormente, se hizo una comparativa sobre la eficiencia de los mismos respecto del tiempo que conlleva su ejecución y una breve reflexión sobre su uso.

Para finalizar, describí los problemas elegidos sobre el tema en cuestión y dí una breve explicación sobre su resolución.

Referencias

[1] Steven Halim, Felix Halim, Suhendry Effendy. (2020, 19 de julio). *Competitive Programming 4: The Lower Bound of Programming Constest in the 2020s*.

[2] Podberezski, V. [vpode]. (2020, 8 de junio). *FIUBA. Teoría de algoritmos 1. Módulo 6: Redes de flujo*. [Video]. YouTube. `https://youtu.be/P-zDGuhHf60`

[3] Buchwald, M. E. [Algoritmos Fiuba Curso Buchwald]. (2020, 27 de julio). *07 - 20 [Opcional] Corte Mínimo en una Red de Flujo*. [Video]. YouTube. `https://www.youtube.com/watch?v=FxQLeDkoq1M&ab_channel=AlgoritmosFiubaCursoBuchwald`

[4] GeeksforGeeks. (2023, 3 de marzo). *Dinic's algorithm for Maximum Flow*. GeeksforGeeks. `https://www.geeksforgeeks.org/dinics-algorithm-maximum-flow/`