Índice general

Proyecto de
Diseño y Análisis de Algoritmos

Belsai Arango Hernández
Daniela Rodríguez Cepero

Universidad de La Habana,
San Lázaro y L. Plaza de la Revolución, La Habana, Cuba
https://github.com/Daroce1012/DAA-problems/tree/Tito_el_corrupto
http://www.uh.cu

1. Problema

Tito el corrupto

Tito se dió cuenta de que la carrera de computación estaba acabando con él y un día decidió darle un cambio radical a su vida. Comenzó a estudiar Ingeniería Industrial. Luego de unos años de fiesta, logró finalmente conseguir su título de ingeniero. Luego de otros tantos años ejerciendo sus estudios (?), consiguió ponerse a la cabeza de un gran proyecto de construcción de carreteras.

La zona en la que debe trabajar tiene n ciudades con m posibles carreteras a construir entre ellas. Cada ciudad que sea incluida en el proyecto aportará a_i dólares al proyecto, mientras que cada carretera tiene un costo de w_i dólares. Si una carretera se incluye en el proyecto, las ciudades unidas por esta también deben incluirse.

El problema estaría en que Tito quiere utilizar una de las habilidades que aprendió en sus años de estudio, la de la malversación de fondos. Todo el dinero necesario para el proyecto que no sea un aporte de alguna ciudad, lo proveería el país y pasaría por manos de Tito. El dinero aportado por las ciudades no pasaría por sus manos. Tito quiere maximizar la cantidad de dinero que pasa por él, para poder hacer su magia. Ayude a Tito a seleccionar el conjunto de carreteras a incluir en el proyecto para lograr su objetivo

Resumen Se tienen n ciudades y m posibles carreteras a construir, cada ciudad que sea incluida en el proyecto aportará a_i dólares y la construcción de la carretera cuesta w_i dólares. Todo el dinero necesario para el proyecto que no sea un aporte de alguna ciudad, lo proveería el país y entonces pasaría a manos de Tito. Se quiere seleccionar un conjunto de carreteras a incluir en el proyecto que maximice la cantidad de dinero que pasa por Tito.

2. Solución

Tenemos que elegir un subconjunto de carreteras y un subconjunto de ciudades, de modo tal que si se elige una carretera, también se elijan todas las ciudades de los que depende este proyecto, y tenemos que maximizar la diferencia entre la suma de los costos de las carreteras elegidas(cantidad de dinero que pasa por Tito) y la suma de los aportes de las ciudades unidas a estas(cantidad de dinero que no pasa por Tito), para así maximizar la cantidad de dinero que pasa por Tito.

Es bastante inconveniente que por los carreteras obtengamos dinero y por las ciudades que están unidas a esta se pierdan ganancias, así que supongamos que obtenemos dinero del país por adelantado por las carreteras y que tenemos que devolver el dinero, si construirla no nos genera ganancias. Estas 2 formulaciones son claramente equivalentes, pero la última es más fácil de modelar.

Una forma sencilla sería crear un grafo con n+m+2 vértices, donde cada vértice representa la fuente(S), el sumidero(T), una carretera o una ciudad. Notaremos a la i-ésima carretera como W_i y la i-ésima ciudad como A_i. Luego, se agregan aristas de peso w_i entre la fuente y la i-ésima carretera, aristas de peso a_i entre la i-ésima ciudad y el receptor, como el dinero que pasa por Tito de las carreteras dependen de las ciudades a las que está unida, para representar esta restricción se agrega una arista de peso ∞ entre la carretera y las ciudades que están unidas a esta. Por último aplicar un corte mínimo sobre este grafo.

Si se corta la arista entre la fuente y la i-ésima carretera, entonces tenemos que devolver el dinero para la i-ésima carretera. Si se corta la arista entre la i-ésima ciudad y el receptor, entonces podemos valorar la construcción de las carreteras unidas a la i-ésima ciudad. Es fácil observar que para todas las dependencias entre una carretera y una ciudad, uniremos las ciudades necesarias o abandonaremos el proyecto ya que sería imposible cortar la arista de peso ∞ entre ellos.

Luego de aplicar el flujo máximo con el corte antes mencionado nos quedaríamos con una red en donde las aristas que están presentes representan las aristas no saturadas, y las que están conectadas al receptor serían las ciudades cuyo aporte es mayor que el costo de la carretera a la que está unida y por tanto la contrucción de esta no le permitiría a Tito lograr su objetivo. Entonces

el conjunto de solución sería el de las carreteras que no estén unidas a estas ciudades.

3. Correctitud

Como problema requiere que se seleccione un conjunto de carreteras y ciudades unidas a estas de tal manera que nos quede el mximo nmero de beneficios. Digamos que inicialmente tenemos todos los ingresos de los proyectos. As que nuestro ingreso total (inicialmente) es igual a $\sum_{i=1} R(w_i)$.

Ahora, como cuando elegimos una carretera, se unen las ciudades correspondientes. Supongamos que las carreteras que no hemos seleccionado estn en un conjunto P y las ciudades que nos hacen perder dinero estn en un conjunto M. Entonces, definitivamente podemos escribir algo como esto:
$\max\{\text{ingresos}\} = \sum_i R(w_i) - \sum_i R(Pi) - \sum_i C(Mi)$.

Podemos ver que la primera de las tres sumas en el lado derecho de la ecuación es constante. Ahora, si podemos minimizar las dos últimas sumas, podemos obtener los ingresos máximos.

Entonces queremos minimizar $\sum_i R(Pi) + \sum_i C(Mi)$.

As que construimos un grfico donde cada proyecto est conectado a la fuente S con capacidad R(pi) y cada mquina est conectada al sumidero T con capacidad C(mi). Y en caso de que un proyecto en particular necesite una cierta cantidad de mquinas, agregamos bordes de cada proyecto a las mquinas que necesita con capacidad para garantizar que si se toma un proyecto, tambin se toman las mquinas requeridas correspondientes.

Finalmente, calculamos el corte mnimo del grfico construido

4. Complejidad Temporal

El algoritmo lee la entrada del problema e itera hasta n (cantidad de ciudades ubicadas en $X0$) para colocar las coordenadas en una arreglo a. El orden de realizar dicha operación es $O(n)$. Luego ordena las coordenadas utilizando el algoritmo de ordenación de python que tiene una complejidad temporal de $O(n \log n)$. Para colocar en la primera posición de la lista un elemento se empleó dos veces el reverse() de las listas de python lo cual es $O(n)$. Para ofrecer la solución se realizan dos recorridos, el primero desde k hasta n, y el segundo desde 1 hasta k, en cada uno se realiza una cantidad constante de operaciones matemáticas elementales $O(1)$, como $0 \leq k \leq n$, la complejidad temporal será en el orden de $O(n)$. Por regla de la suma la complejidad temporal del algoritmo es de $O(n \log n)$.