Diseño y Análisis de Algoritmos

Proyecto # 2: Tito el corrupto

Laura Victoria Riera Pérez
Marié del Valle Reyes

Cuarto año. Ciencias de la Computación.
Facultad de Matemática y Computación, Universidad de La Habana, Cuba

8 de mayo de 2023

I. Repositorio del proyecto

https://github.com/computer-science-crows/algorithms-design-and-analysis

II. Definición inicial del problema

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 está 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 proveerá el país y pasará por manos de Tito. El dinero aportado por las ciudades no pasará 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.

III. Definición en términos matemático - computacionales

1. Preliminares

Definición 1. *Una **red de flujo** $G = (V, E)$ es un grafo dirigido en el que a cada par ordenado (u,v), u, v∈V, se le asocia una función de capacidad no negativa c(u,v)≥ 0 y en el que se distinguen dos vértices: la fuente s y el receptor t.*

Definición 2. *Sea G=⟨V,E⟩ una red de flujo con función de **capacidad** c y vértices fuente y receptor s y t respectivamente. Un **flujo** en G es una función real $f : VxV \to \mathbb{R}^+$ que satisface las siguientes propiedades:*

1. *Restricción de capacidad: Para todo u,v∈V se cumple que*

$$0 \leq f(u, v) \leq c(u, v) \tag{1}$$

2. Conservación de flujo: Para todo u∈V- {s,t}, se cumple que

$$\sum_{v \in V} f(v,u) = \sum_{v \in V} f(u,v) \tag{2}$$

Definición 3. *A la cantidad no negativa f(u,v) se le denomina **flujo neto** de u a v.*

Definición 4. *El **valor de un flujo** se define como:*

$$|f| = \sum_{v \in V} f(s,v) \tag{3}$$

Definición 5. *Sea G=⟨V,E⟩ una red de flujo con origen s y receptor t. Sea f un flujo en G, y sean u,v∈V. Se define la **capacidad residual** mediante la siguiente función:*

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{,si } (u,v) \in E \\ f(v,u) & \text{,si } (v,u) \in E \\ 0 & \text{,en otro caso} \end{cases} \tag{4}$$

*La **red residual** de G inducida por f es una red $G_f = (V, E_f)$, donde $E_f = \{(u,v) \in VxV \mid c_f(u,v) > 0\}$.*

Definición 6. *Si f es un flujo en la red original G y f' es un flujo en la red residual correspondiente G_f, entonces, se define el aumento del flujo f (en la red original) por f', y denotado por $f \uparrow f'$, a la función $f \uparrow f'$: $VxV \rightarrow \mathbb{R}$ dada por la expresión:*

$$f \uparrow f'(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & \text{si } (u,v) \in E \\ 0 & \text{en otro caso} \end{cases} \tag{5}$$

Lema 1. $|f \uparrow f'| = |f| + |f'|$

Definición 7. *Dada una red de flujo G=⟨V,E⟩ y un flujo f, un **camino aumentativo** p es un camino simple de s a t en la red residual G_f.*

Definición 8. *La capacidad residual de un camino aumentativo p, denotada por $c_f(p)$, es el valor máximo en el cual es posible aumentar el flujo en cada arista del camino sin violar la restricción de capacidad*

$$c_f(p) = min\{c_f(u,v) | (u,v) \in p\} \tag{6}$$

Lema 2. *Sea G=⟨V,E⟩ una red de flujo, sea f un flujo en G y sea p un camino aumentativo en G_f. Si se define el flujo f_p , sobre la red residual, como*

$$f_p(u,v) = \begin{cases} c_f(p) & \text{si } (u,v) \in p \\ 0 & \text{en otro caso} \end{cases} \tag{7}$$

entonces el valor de f_p es $|f_p| = c_f(p) > 0$

IV. Línea de pensamiento

V. Algoritmo Ford-Fulkerson

El algoritmo Ford-Fulkerson es un algoritmo utilizado para encontrar el flujo máximo en una red de flujo. Esta red se representa como un grafo dirigido donde cada arista tiene una capacidad que indica la cantidad máxima de flujo que puede pasar por ella.

El algoritmo Ford-Fulkerson encuentra el flujo máximo en la red de flujo utilizando la técnica de aumentar caminos. En cada iteración del algoritmo, se busca un camino aumentante, es decir, un camino desde el nodo fuente hasta el nodo sumidero donde todas las aristas tienen capacidad positiva y suficiente para aumentar el flujo actual. Una vez encontrado el camino aumentante, se aumenta el flujo en esa ruta tanto como sea posible. Este proceso se repite hasta que ya no haya caminos aumentantes en el grafo residual.

El grafo residual se obtiene a partir del grafo original restando el flujo actual del flujo máximo para obtener la capacidad residual de cada arista. De esta manera, se pueden buscar caminos aumentantes en el grafo residual sin utilizar aristas que ya están completamente saturadas.

El algoritmo Ford-Fulkerson garantiza que, una vez que se alcanza el flujo máximo, no hay caminos aumentantes en el grafo residual y, por lo tanto, el flujo es óptimo.

Existen varias implementaciones del algoritmo Ford-Fulkerson, como el método de la ruta más corta o el método de Edmonds-Karp, que utilizan diferentes estrategias para buscar caminos aumentantes de manera eficiente.

i. Explicación del algoritmo

ii. Complejidad Temporal

iii. Complejidad espacial

VI. Generador de casos de prueba

En *src/app/generator.py* fue implementado un generador, el cual recibe una cantidad *s* de muestras a producir, genera valores random con el formato de entrada de los algoritmos implementados, halla la solución óptima con *backtrack* y las guarda en *json/test_cases.json*. Se generaron 3000 casos de prueba, con *n* máximo igual a 11, dado que, como se mencionó anteriormente, es lo que puede ejecutar el *backtrack*.

VII. Tester

En *src/app/tester.py* fue implementado un tester, que recibe una función y prueba el desempeño de la misma en cuanto a si obtuvo la solución óptima o no, y el tiempo que demoró en hacerlo, comparando con los casos de prueba obtenidos con el generador. Dichos resultados se muestran en consola de la siguiente forma:

Además, estos resultados se guardan en un *.json* con el nombre de la función en la carpeta tests. Las soluciones implementadas fueron testeadas para todos los casos de prueba generados y pueden encontrarse en *json/tests/simplex_solution.json* y *json/tests/hungarian_solution.json*

VIII. Comparación de soluciones implementadas

Referencias

[1] Cormen, Thomas H. y otros. *Introduction to Algorithms*. The MIT Press. 4ta Edición. Cambridge, Massachusetts. 2022.