Algoritmo Ford - Fulkerson: Flujo Máximo

Equipo 2: Ana, Dira, Iván y León

Agenda

Algoritmo Ford – Fulkerson

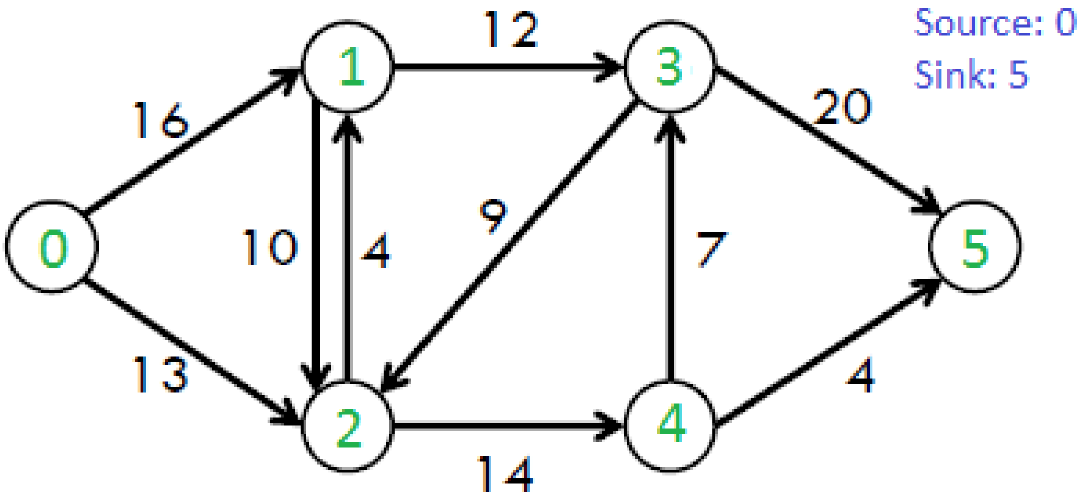
Reimplementación de ffmaxflow

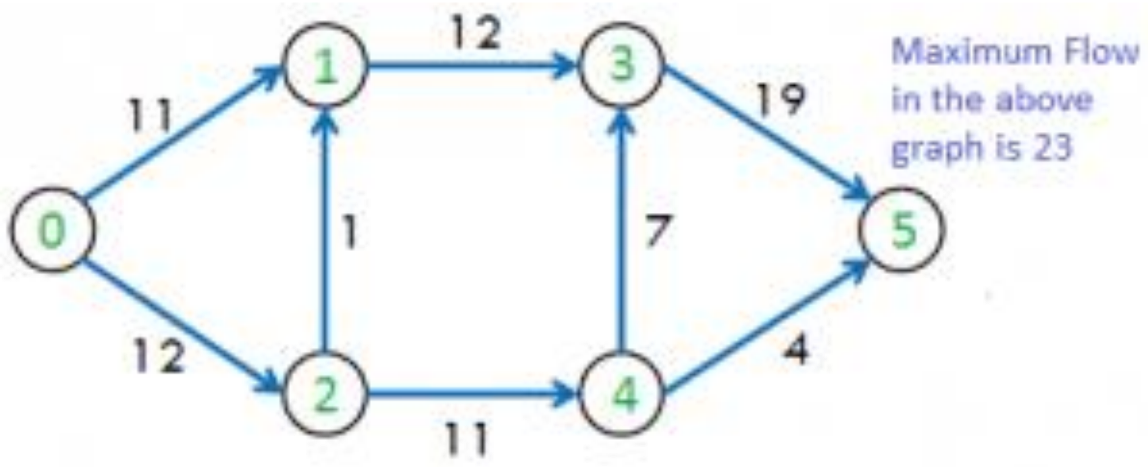
Aplicaciónes en la vida real

Flujo máximo

- Los problemas de flujo máximo implican encontrar un flujo factible a través de una red con una sola fuente y un solo sumidero que sea máximo.

- Cada arco está etiquetado por una capacidad la cual representa el flujo máximo que puede pasar del nodo i al nodo j. El objetivo es encontrar la capacidad máxima de flujo que pasa del nodo origen (fuente) al nodo destino (sumidero). El flujo máximo en esta red es 23.





Algoritmo Ford-Fulkerson

¿Qué propone el Algoritmo Ford-Fulkerson?

- El algoritmo de Ford-Fulkerson propone buscar caminos en los que se pueda aumentar el flujo, hasta que se alcance el flujo máximo. Su nombre viene dado por sus creadores, L. R. Ford, Jr. y D. R. Fulkerson.

- Sea $G(V, E)$ un grafo, con V vértices, E aristas y donde por cada arista (u, v), tenemos una capacidad $c(u, v)$ y un flujo $\mathrm{f}(u, v)$. Se busca maximizar el valor del flujo desde una fuente s hasta un sumidero t.

- El método inicia con $\mathrm{f}(u, v) = 0$ para toda $(u, v) \in V$. En cada iteración, se incrementa el flujo en G mediante el resultado de una búsqueda de un camino de aumento en una red.

- El flujo a aumentar se debe considerar factible, es decir:
  El flujo de para toda arista (u, v) no debe ser mayor que la capacidad de dicha arista. El flujo que sale de la fuente s debe ser igual al que llega al sumidero t.

Red residual

- la red residual G_f consiste en el grafo que representa el como cambia la capacidad de cada una de las aristas con respecto al flujo del camino de aumento c_f en el grafo G.

Caminos de aumento

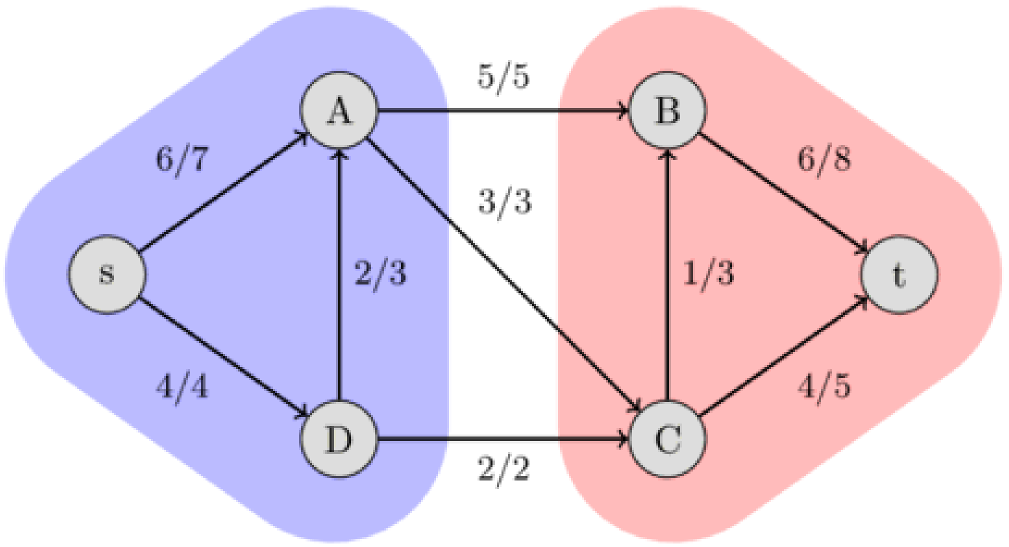
- Un camino de aumento es un camino dirigido de la fuente s al sumidero t en G_f, donde la capacidad del camino de aumento es el mínimo de las capacidades de sus aristas.

Complejidad

- Cuando las capacidades son números enteros, el tiempo de ejecución de Ford-Fulkerson está limitado por $O(E_f)$ donde E es el número de aristas en el grafo y f es el flujo máximo en el grafo.

Teorema: cortadura mínima

- Consiste en generar una partición de los vértices de una red de flujo en dos conjuntos, de modo que un conjunto incluye la fuente s y el otro incluye el sumidero t. Por lo tanto, el flujo máximo está limitado por la capacidad mínima de corte.
- El corte se define como la suma de las capacidades de los arcos desde el lado de la fuente hasta el lado del sumidero. El flujo máximo tiene que ser igual a la capacidad del corte mínimo.
- Se puede encontrar un corte mínimo después de realizar un cálculo de flujo máximo utilizando el método Ford Fulkerson. Un posible corte mínimo es el siguiente: el conjunto de todos los vértices que se pueden alcanzar desde s en el grafo residual (utilizando aristas con capacidad residual positiva), y el conjunto de todos los demás vértices.
- La cortadura mínima de este grafo contiene el siguiente conjunto de aristas: $A \rightarrow B, A \rightarrow C$ y $D \rightarrow C$.



Reimplementación de ffmaxflow

Robustecimiento

Reimplementación

- La implementación del paquete se realizó a través de Python haciendo uso de clases.

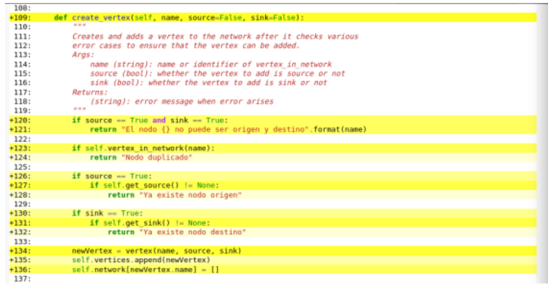
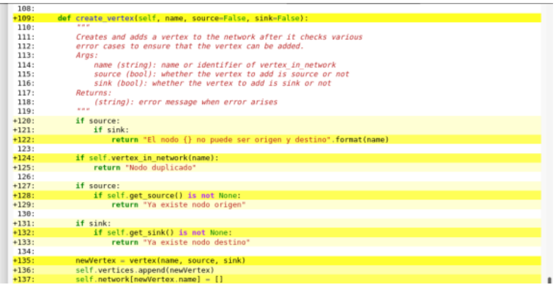
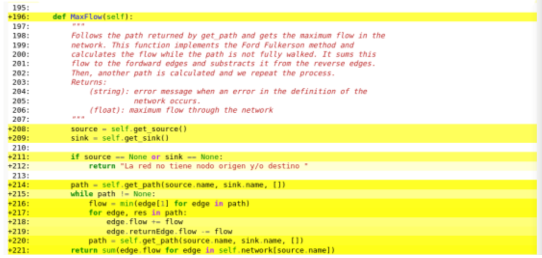
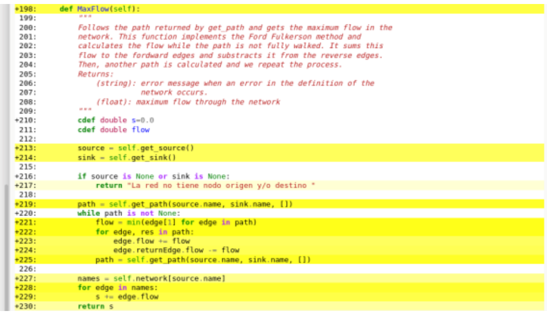
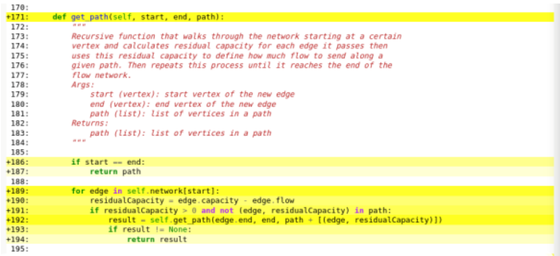
- Una vez desarrollado el paquete de manera básica utilizamos las herramientas de kale y minikube para correr varios experimentos y así detectar valores con los cuales falla nuestro paquete o posibles mejoras a realizar.

- Ante de iniciar con la reimplementación realizamos un perfilamiento de ffmaxflow para así tener una idea del desempeño de nuestro paquete. Durante el perfilamiento descubrimos que el método de get_path toma mucho tiempo en relación con los tiempos de las demás líneas, además de la creación de vertices y el cálculo mismo del flujo máximo.

- En cuanto al perfilamiento de memoria, no encontramos que nuestro paquete estuviera usando memoria excesiva por lo que decidimos optimizar únicamente el tiempo de ejecución y la optimización se realizara con Cython.

Reimplementación en Cython

- Con ayuda de las anotaciones amarillas dadas por Cython obtenidas a partir del archivo con extensión .pyx se identificaron las líneas que "más Python utilizaban". Recordemos que mientras más código que se pueda traducir a C se tenga, menor será el tiempo de ejecución. En particular, estas anotaciones se realizan sobre un archivo html, donde las líneas más amarillas representan que se está usando más Python y las blancas, más C.

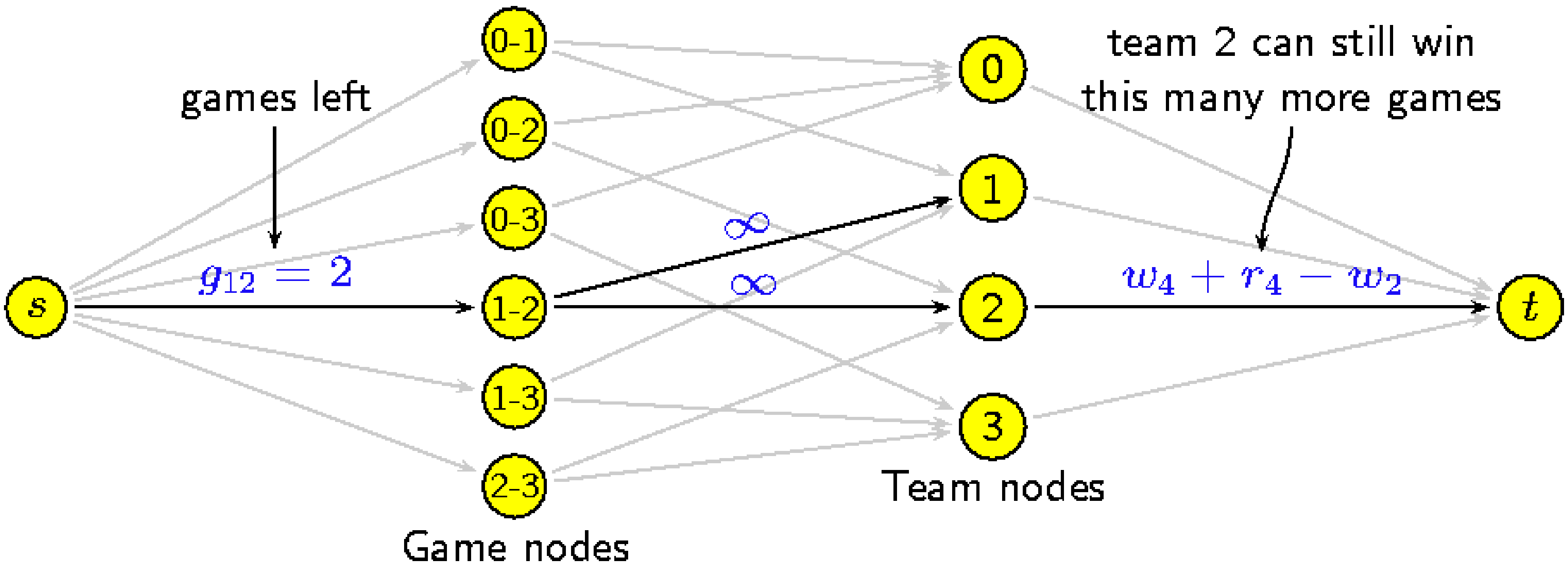

Aplicaciones en la vida real

Eliminación de béisbol

- El béisbol en los EE.UU. Consta de dos ligas diferentes, cada una de las cuales se divide en tres divisiones de aproximadamente 5 equipos. Durante la temporada regular, cada equipo juega 162 partidos en total, la mayoría de los cuales son contra equipos de su división (76 contra la misma división, 66 contra la misma liga, 20 contra una liga diferente). El objetivo de esto es intentar clasificarse para los playoffs, lo que se puede hacer terminando primero en la división.

- Todo esto se suma al deseo de poder determinar si es posible ver si un equipo en particular aún puede potencialmente ganar la división y llegar a los playoffs dada la clasificación actual de la liga. Resolver este problema significa que tenemos que mirar tablas que podrían parecerse a la siguiente:

Schwartz, B. L propuso un método que reduce este problema a como un problema de flujo máximo en una red.
En este método se crea una red para determinar si se elimina el equipo k.



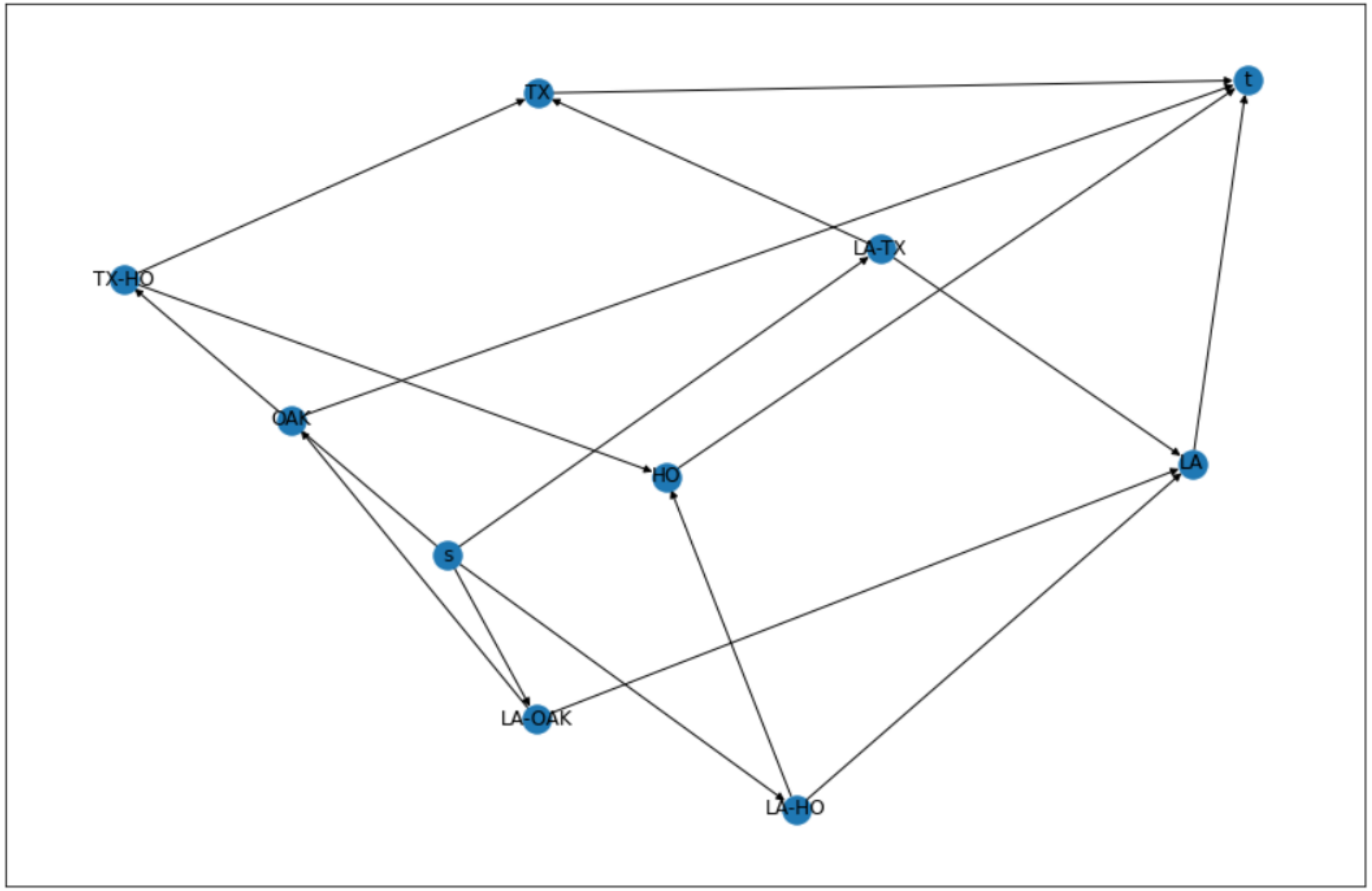
Descripción del dataset

- En este ejemplo se ve claramente que Houston ya está eliminado, la cantidad máxima de victorias que pueden obtener es 77 + 4 = 82 que es menos de las 82 victorias que ya tiene LA. Sin embargo, los casos de los tres equipos restantes están todos entrelazados. Tomemos a Texas, por ejemplo, si ganan tres juegos pueden llegar a 83 victorias y superar a Los Ángeles. Sin embargo, para ganar que Texas gane la liga, Los Ángeles debería perder 8 de sus 9 juegos restantes. Entonces se tiene el siguiente escenario para LA:

- 1 derrota contra Texas
- 2 derrotas contra Houston
- Y todavía tendría 5 derrotas necesarias de 6 juegos contra Oakland.
- Sin embargo, si esto sucede, Oakland alcanzará 83 victorias y sería el equipo ganador. Claramente, no es suficiente mirar las victorias de un equipo y los juegos que quedan por jugar. Necesitamos considerar contra quién se juegan esos juegos.

	Teams	Wins	Losses	Games_left	LA	TX	OAK	HO
0	LA	82	71	9	0	1	6	2
1	TX	80	79	3	1	0	0	2
2	OAK	78	78	6	6	0	0	0
3	HO	77	81	4	2	2	0	0

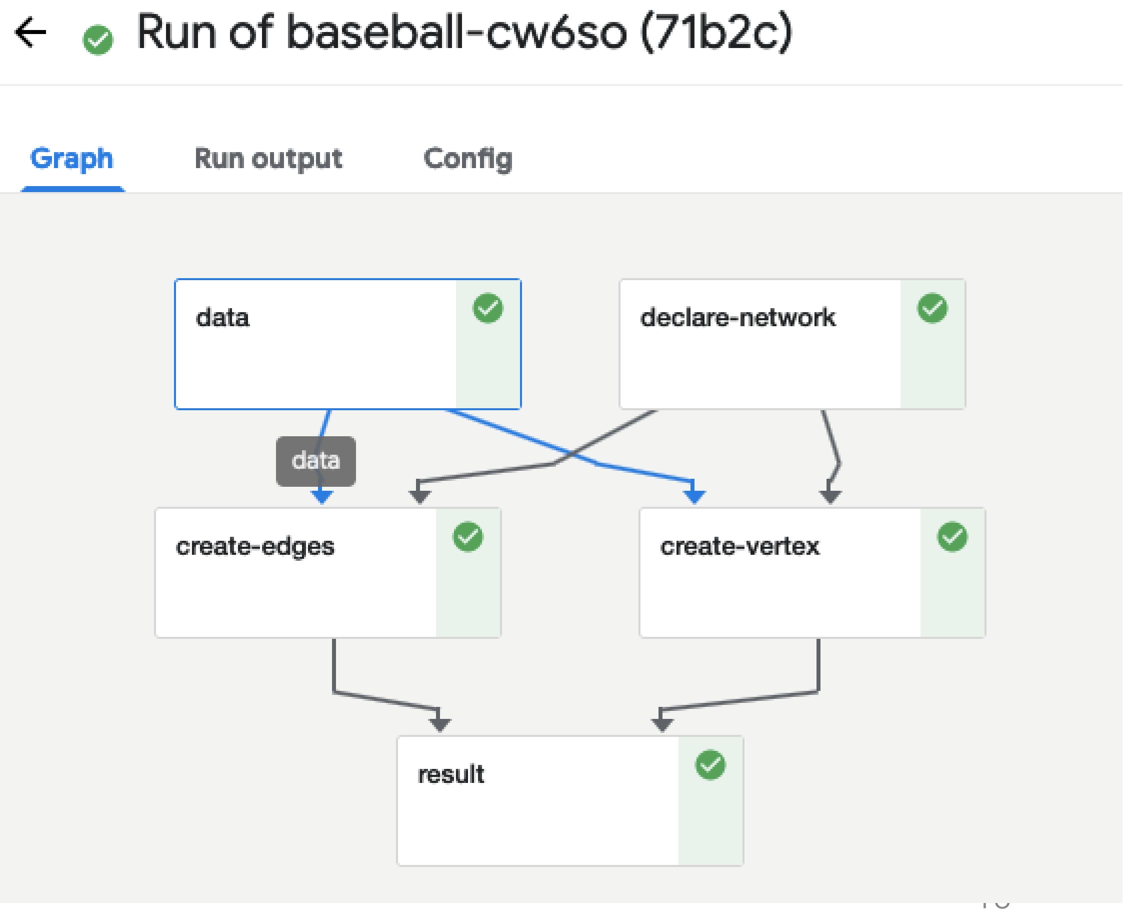
Gráfica de las rutas



Resultados

- Se realizó el ejercicio con 3 distintos paquetes (*ffmaxflow, ffmaxc y networkx*) y se comparó el tiempo de ejecución de cado uno de ellos.

Método	Resultado	Tiempo de ejecución
ffmaxflow	11	0.002524 s
ffmaxc	11	0.001595 s
networkx	11	0.006794 s



Planificación de aerolíneas

- En la industria de las aerolíneas, un problema importante es la programación de las tripulaciones de vuelo. El problema de programación de la aerolínea puede considerarse como una aplicación del flujo de red máximo extendido. La entrada de este problema es un conjunto de vuelos F que contiene la información sobre dónde y cuándo sale y llega cada vuelo. En una versión de la programación de las aerolíneas, el objetivo es producir un programa factible con un máximo de k escalas.
- Para resolver este problema, se utiliza una variación del problema de circulación llamado circulación limitada, que es la generalización de los problemas de flujo de la red, con la restricción adicional de un límite inferior en los flujos de los arcos.

Descripción del dataset

- El dataset a utilizar viene de un repositorio en github.
- El dataset contiene información de los vuelos del 6 enero de 2019 de una cierta aerolínea en Estados Unidos y muestra los registros de sus 565 vuelos de ese día para únicamente 10 aeropuertos.

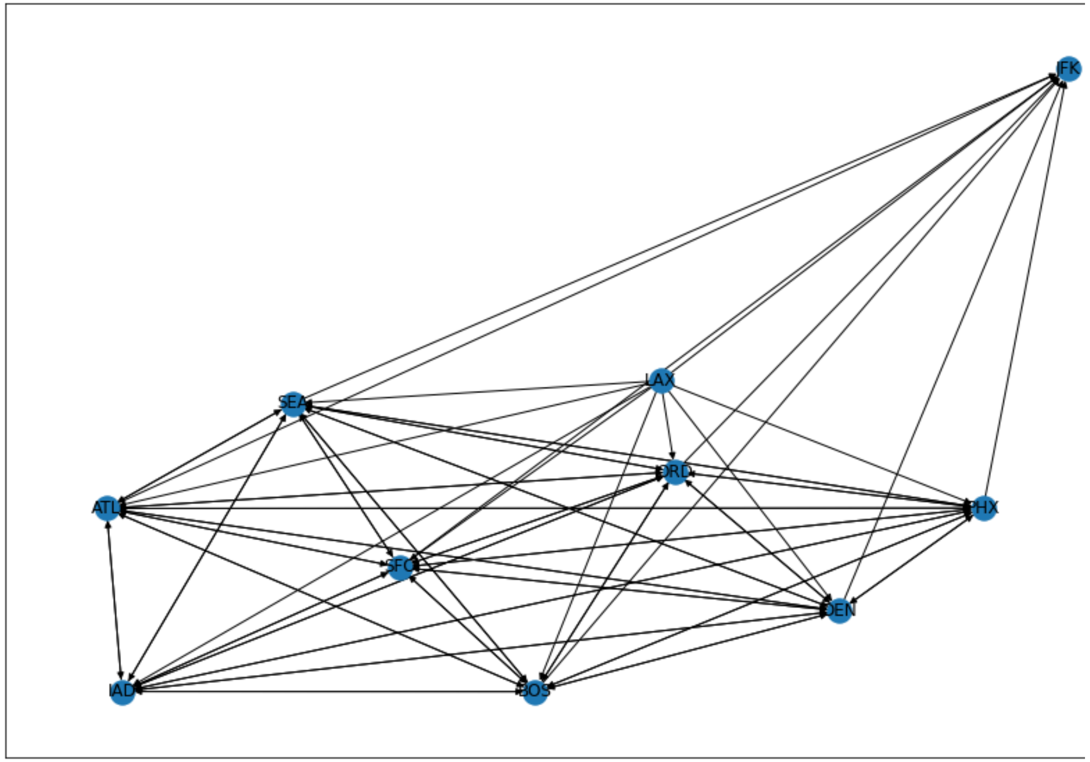
	Source	Destination	Depature	Arrival	capacity
0	ORD	ATL	530	829	185
1	BOS	ATL	545	854	185
2	LAX	SEA	600	900	165
3	PHX	ATL	600	1127	185
4	SFO	ATL	600	1332	230

El dataset cuenta con cinco columnas:

- Source: identificador del aeropuerto origen del que sale el vuelo
- Destination: identificador del aeropuerto destino al que llega el vuelo
- Departure: hora de despegue del vuelo en el aeropuerto origen en horas militares
- Arrival: hora de aterrizaje del vuelo en el aeropuerto destino en horas militares
- capacity: cantidad máxima de pasajeros que puede transportar el vuelo (sin contar al crew)

Gráfica de las rutas

Identificador	Nombre	Ciudad
ATL	Hartsfield-Jackson Atlanta International	Atlanta
BOS	Boston-Logan International	Boston
DEN	Denver International	Denver
IAD	Washington Dulles International	Washington
JFK	New York John F. Kennedy International	New York
LAX	Los Angeles International	Los Angeles
ORD	Chicago O'Hare International	Chicago
PHX	Phoenix Sky Harbor International	Phoenix
SEA	Seattle-Tacoma International	Seattle
SFO	San Francisco International	San Francisco



Solución

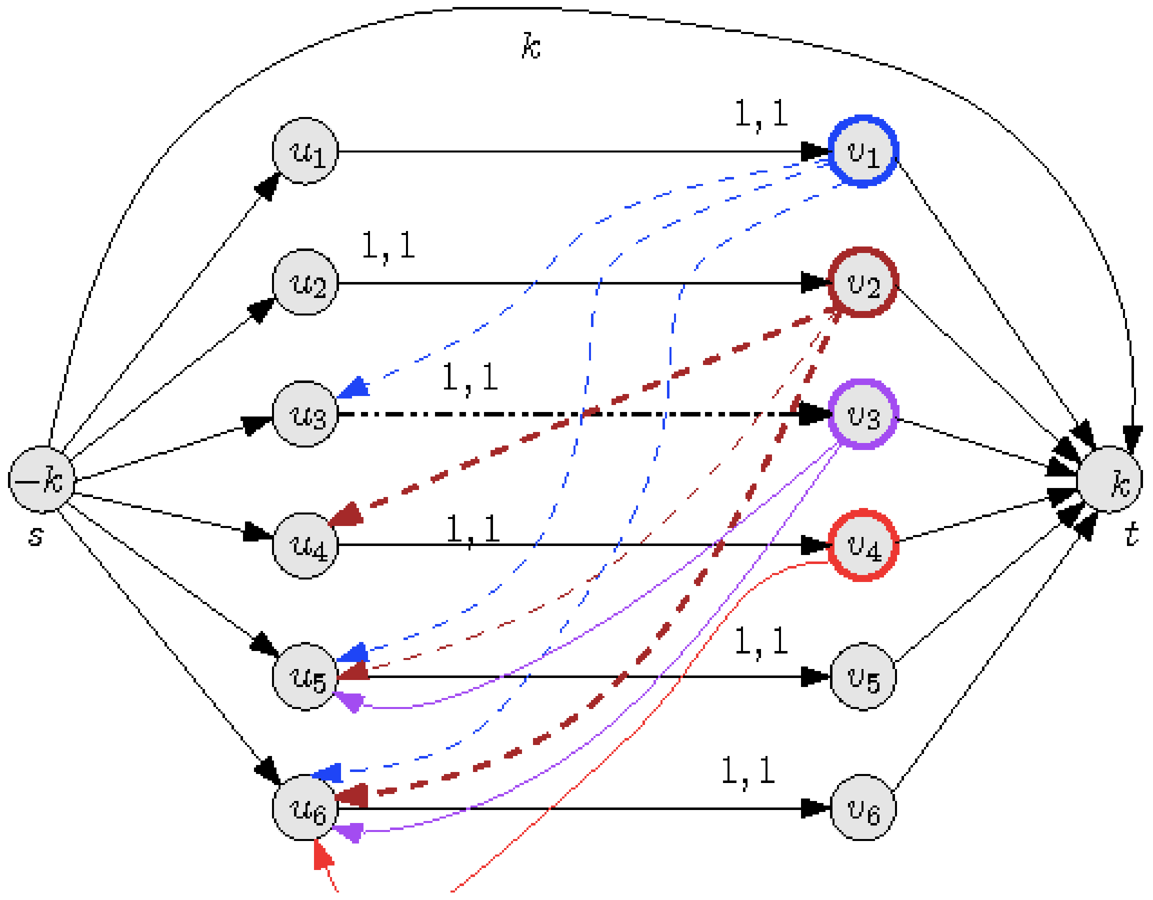
- Para cada vuelo creamos los nodos $u_i, v_i \in V$ que hacen referencia a los aeropuertos de salida y destino respectivamente. También se deben agregar los nodos origen y destino s, t respectivamente y se añade un arco de $s \to t$ con capacidad ∞ (demanda de t) unidades de flujo.

- El vuelo que va del aeropuerto u_i al aeropuerto v_i se representa con el arco $u_i \to v_i$ con capacidad 1.

- Si el mismo avión que realizó el vuelo $u_i \to v_i$ puede realizar el vuelo $v_i \to w_i$ entonces se agrega este nuevo arco con capacidad 1.

- Ya que cada avión puede comenzar el día con el vuelo i se agrega un arco con capacidad 1 del nodo origen a los nodos pertenecientes a las salidas de los vuelos.

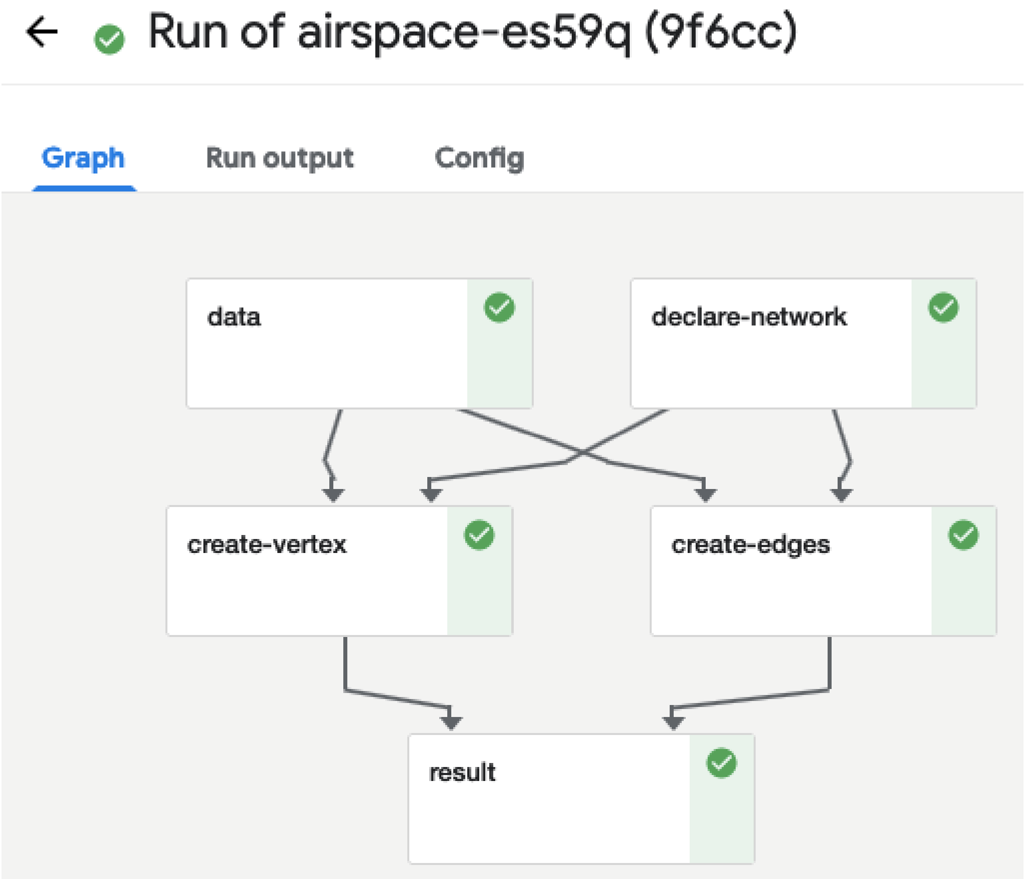
- Ya que cada avión puede terminar el día con el vuelo i se agrega un arco con capacidad 1 de los nodos pertenecientes a los arribos de los vuelos.



Resultados

- Se realizó el ejercicio con 3 distintos paquetes (*ffmaxflow, ffmaxc y networkx*) y se comparó el tiempo de ejecución de cado uno de ellos.

Método	Resultado	Tiempo de ejecución
ffmaxflow	10,179	64 s
ffmaxc	10,179	56 s
networkx	10,179	0.002 s



Problema de flujo con oferta y demanda

- Otra de las aplicaciones de este paquete es resolver problemas de flujo con oferta y demanda, este tipo de problemas es muy similar al problema original de flujo máximo pero ahora los nodos cuentan con un parámetro extra, una oferta o una demanda, es decir, un nodo origen debería tener una capacidad de oferta y un nodo destino debería tener una capacidad de demanda.

- Ahora el problema a resolver es encontrar un flujo que satisfaga la demanda, podemos hallar una solución para este problema con el algoritmo de Ford-Fulkerson, transformando la red de tal manera que la solución del problema de flujo máximo nos de la solución del problema de flujo con oferta y demanda.

- Para poder hacer esta transformación necesitamos:

Agregar un nodo de origen s y arcos desde él a cada nodo origen original p_i con capacidad p_i donde p_i es la oferta de cada nodo origen original.

Agregar un nodo destino t y arcos de todas los nodos destino originales v_i a t con capacidad d_i donde d_i es la demanda de cada nodo destino original.

Ahora con esta nueva red, existe un flujo que satisface la demanda si y solo si:

$$Flujo\ Máximo = \sum_{i}^{V} d_i$$

Si ocurre lo anterior, la solución del problema de flujo máximo nos daría la respuesta de cuantas unidades enviar por cada arco para satisfacer la demanda, es decir, obtendríamos la solución del problema de flujo con oferta y demanda.

Solución

En (a) observamos los nodos originales, estos tienen un número en su interior que representa la oferta y la demanda, los nodos blancos representan los nodos origen, por lo que el número representa una oferta y los nodos grises son nodos destino y estos tienen una demanda que cumplir. Observamos 5 arcos con un número que representa la capacidad de cada arco.

En (b) observamos la creación de dos nodos extra, s^* y t^*, que serán nuestros nuevos nodos origen y destino respectivamente, también se crearon nuevos arcos con capacidad igual a la oferta o demanda del nodo al que se conectan.

En (c) observamos la solución de este problema, la solución del problema de flujo máximo es 6 y es igual a la demanda de nuestra red, por lo que también se resolvió el problema de flujo con oferta y demanda.

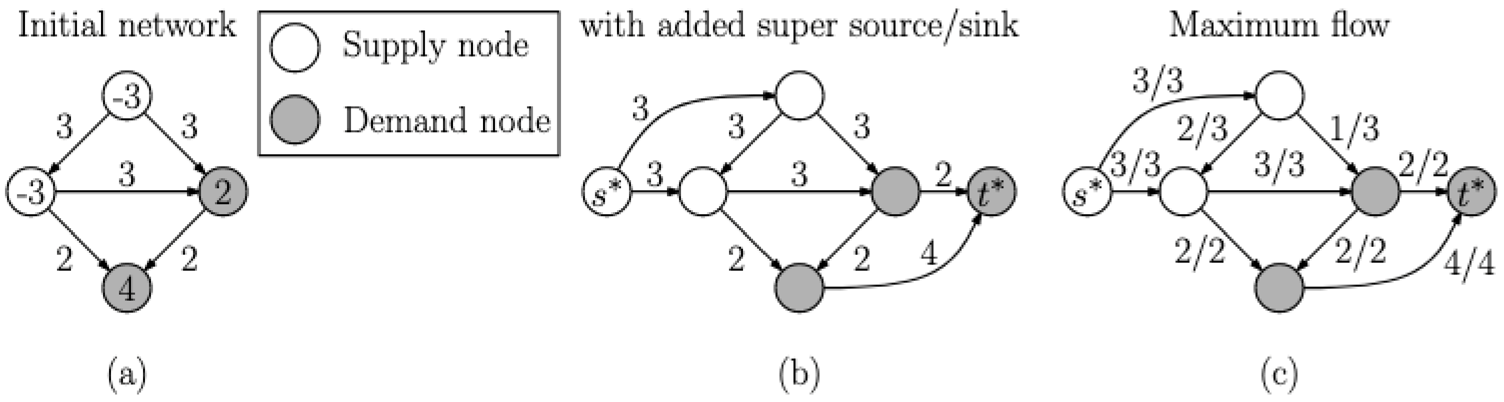
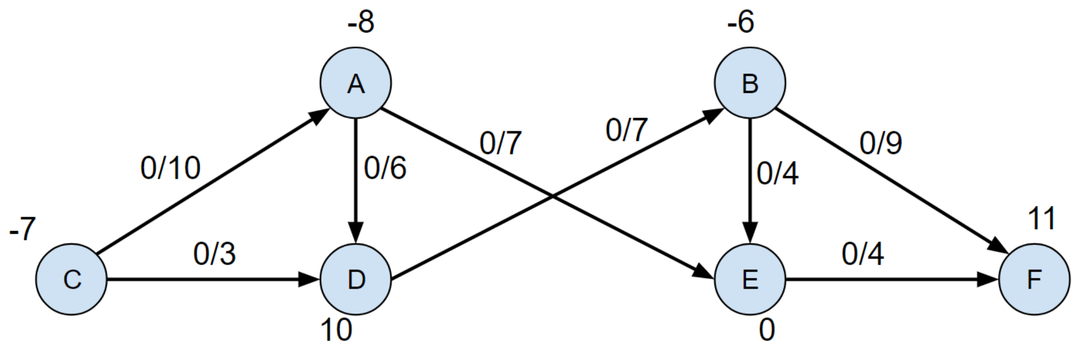


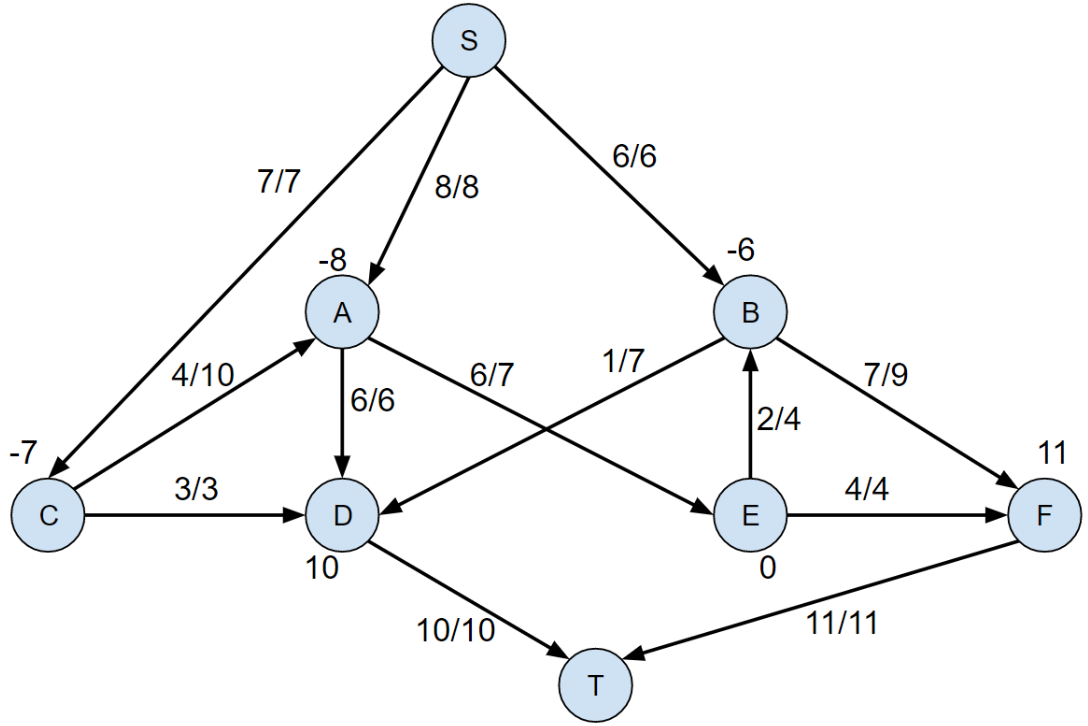
Fig. 2: Reducing the circulation problem to network flow.

Gráfica de las rutas

Ejemplo 1



Ejemplo 2



Resultados

- Se realizó el ejercicio con 3 distintos paquetes (*ffmaxflow, ffmaxc y networkx*) y se comparó el tiempo de ejecución de cado uno de ellos.

Método	Resultado	Tiempo de ejecución
ffmaxflow	6	8.86284 s
ffmaxc	6	6.13964 s
networkx	6	23.0763 s

Conclusiones

- Kale es una herramienta sencilla de utilizar que nos permite familiarizarnos con el flujo de trabajo de kubeflow con kubernetes, además nos permite llevar a cabo experimentos de manera local sin tener que preocuparnos por el momento de levantar clústeres o interactuar con la línea de comandos, nos deja enforcarnos totalmente en nuestro código.
- La compilación a C con ayuda de Cython es una buena opción para optimizar y agilizar código en Python que no utiliza objetos vectorizados o que no pueden ser vectorizados, cuando se utiliza Python "pelón" y cuando las variables no cambian de tipo durante la ejecución del código.
- No todas las líneas de código tienen su equivalente en C, pues en C no existe el concepto de clase ni de objeto.
- Cython nos permite tener la comodidad de programar en Python y la velocidad de un módulo compilado anticipadamente. El costo es que se requiere de conocimiento del lenguaje de C.

Gracias

Equipo 2

https://github.com/diramtz/ProyectoFinal_MaxFlow