Ciclo de identificación de arbitraje en el comercio de criptomonedas

(Arbitrage Identification Cycle in Crypto Trading)

Equipo 1:
Nyrma Paulina Hernández Trejo
Aide Jazmín González Cruz
Joel Jaramillo Pacheco
Jesús Enrique Miranda Blanco

Introducción

¿Qué es el *trading*?

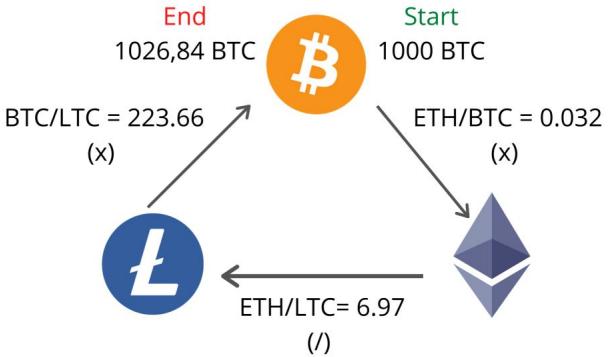
El *trading* consiste en la compra-venta de activos cotizados con mucha liquidez en un mercado que puede ser de acciones, divisas e inversiones. Este mercado financiero es electrónico y está regulado. Su objetivo es obtener un beneficio económico cuando la operación genera una plusvalía.

Las criptomonedas no tienen una autoridad centralizada, como un banco central que las emite o que regule su circulación en la economía. Por lo que el *trading* en el ámbito de criptomonedas son instrucciones o programas diseñados para detectar patrones y usarlos como disparadores para realizar operaciones. Esta técnica aprovecha la velocidad y la ausencia de emociones que tienen las computadoras sobre los comerciantes humanos.

¿Qué es el *arbitraje*?

El arbitraje es la compra y venta simultánea del mismo activo en diferentes mercados para beneficiarse de pequeñas diferencias en el precio de cotización del activo. Explota variaciones de corta duración en el precio de instrumentos financieros idénticos o similares en diferentes mercados o en diferentes formas.

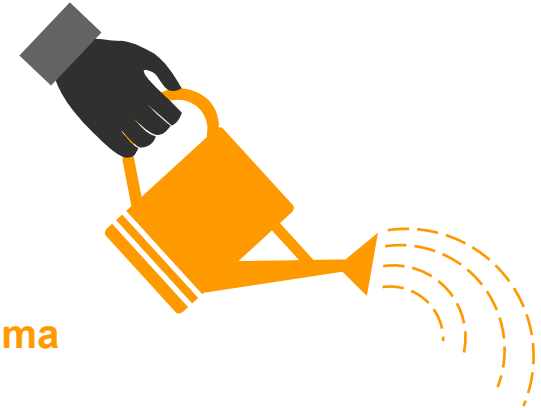
El arbitraje existe como resultado de las ineficiencias del mercado y explota esas ineficiencias y las resuelve.



Planteamiento del Problema

❖ Problema de Negocio

Identificar las ventajas en el arbitraje de criptomonedas, donde se necesita computar rápidamente el camino o costo más pequeño entre todas las criptomonedas en tiempo real.



❖ Problema de flujo en redes. Problema de Distancia Mínima

La idea principal es la de encontrar el camino con un costo mínimo entre una fuente (V) y un destino (E), donde V y E son nodos. Y el flujo se puede traducir como el costo de V a E que se puede denotar como c.

Dado $G = (V, E)$ grafo (dirigido), $c \in \mathbb{R}^E$ y $s, t \in V$

Métodos que resuelven el problema

- **Algoritmo de Dijkstra,** resuelve el problema de los caminos más cortos desde un único vértice origen hasta todos los otros vértices del grafo.
- **Algoritmo de Bellman - Ford**, resuelve el problema de los caminos más cortos desde un origen si la ponderación de las aristas es negativa.
- **Algoritmo de Búsqueda A***, resuelve el problema de los caminos más cortos entre un par de vértices usando la heurística para intentar agilizar la búsqueda.
- **Algoritmo de Floyd - Warshall,** resuelve el problema de los caminos más cortos entre todos los vértices.
- **Algoritmo de Johnson,** resuelve el problema de los caminos más cortos entre todos los vértices y puede ser más rápido que el de Floyd-Warshall en grafos de baja densidad.
- **Algoritmo de Viterbi**, resuelve el problema del camino estocástico más corto con un peso probabilístico adicional en cada vértice.

Objetivo

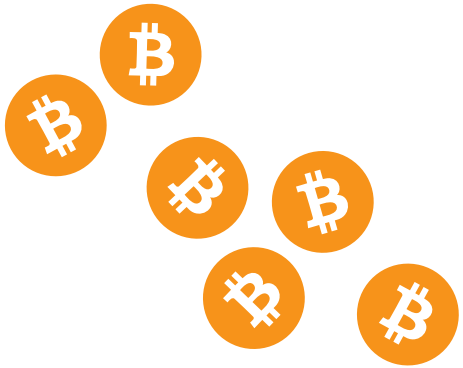
Resolver el tema de **Ciclos de Arbitraje en Criptomonedas**, a través de encontrar el **camino más corto** en un grafo dirigido ponderado, usando el **método de Bellman Ford**

Bellman Ford

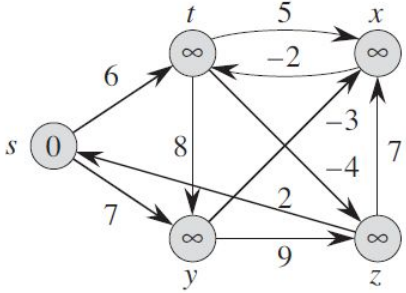
Algoritmo de Bellman Ford

Características

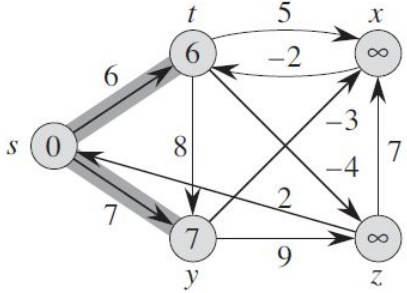
- Calcula la **ruta más corta** desde un nodo origen hacia los demás nodos en un **grafo dirigido** con pesos en las aristas.
- La ventaja de este algoritmo es que los pesos pueden tomar **valores negativos** y, en dicho caso, se detectaría la existencia de un ciclo negativo (un ciclo negativo se forma cuando la suma de todos sus pesos toma un valor menor a cero).
- La idea de este algoritmo es recorrer **todos los bordes** del grafo uno por uno en un orden aleatorio.
- Se basa en el **Principio de Relajación** al sobreestimar la longitud del nodo inicial a cada vértice para después actualizarla a un valor más cercano.



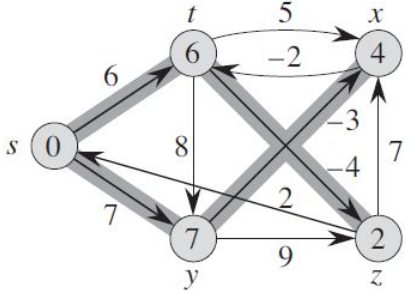
Algoritmo



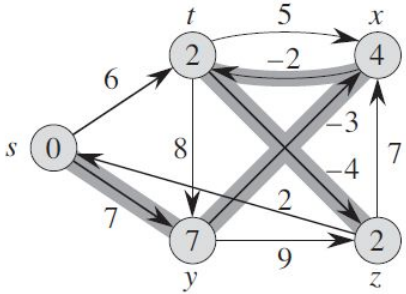
(a)



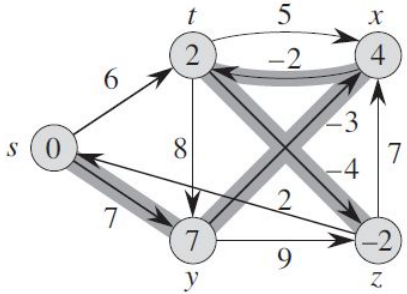
(b)



(c)



(d)



(e)

Fuente: https://www.researchgate.net/figure/The-schematic-procedure-of-the-Bellman-Ford-Moore-algorithm-other-related-explanations_fig7_282135528

Desarrollo

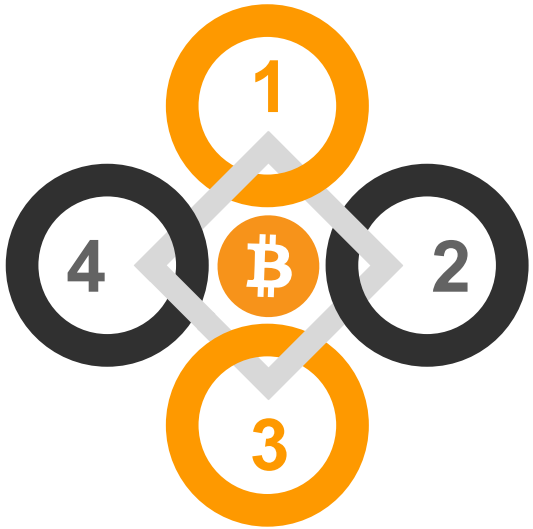
Desarrollo

Extracción de Datos

Extracción de datos usando el módulo “pandas datareader” del API de yahoo finanzas. Se descargaron 10,105 monedas del 2022-05-01 al 2022-05-13

Ciclo Resultante

Una vez aplicado el algoritmo de Bellman Ford al grafo de intercambios se obtiene la ruta más corta entre dos criptomonedas



Preprocesamiento de Datos

Creación de un pipeline para el preprocesamiento de los datos con los siguientes pasos:

- Creación de matriz de intercambios
- Transformación de matriz a grafo

Algoritmo de Bellman Ford

Implementación y optimización del algoritmo en python.
En la optimización se removieron NaN y se compilo en C.

Datos

Data Set

Web scraping

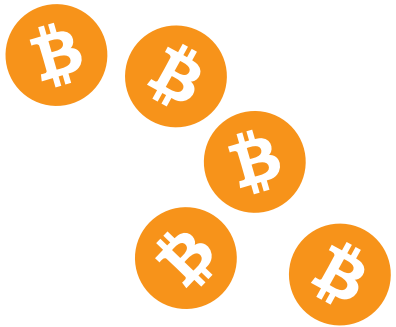
Se realiza web scraping al api de **coinmarketcap** para obtener los siguientes datos:

- Simbolos
- Fecha en la que se agregó
- Última actualización
- Precio
- Capitalización de mercado

Yahoo Finance

Una vez obtenidos los símbolos se hace la descarga de los históricos usando el módulo de pandas **datareader** con la siguiente estructura:

- Fecha
- High
- Low
- Open
- Close
- Volume
- Adj Close



Transformación de datos

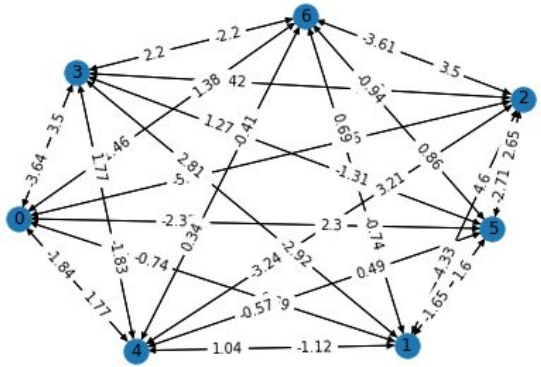
Matriz de intercambio

	0	1	2	3	4	5	6
0	1.00	2.10	156.56	38.22	6.32	10.73	4.33
1	0.50	1.00	76.17	18.60	3.08	5.22	2.11
2	0.01	0.01	1.00	0.25	0.04	0.07	0.03
3	0.03	0.06	4.14	1.00	0.17	0.28	0.11
4	0.17	0.35	25.77	6.29	1.00	1.77	0.71
5	0.10	0.20	15.18	3.71	0.61	1.00	0.42
6	0.25	0.50	37.30	9.11	1.51	2.56	1.00

Matriz de intercambio transformada

	0	1	2	3	4	5	6
0	-0.000000	-0.741937	-5.053439	-3.643359	-1.843719	-2.373044	-1.465568
1	0.693147	-0.000000	-4.332968	-2.923162	-1.124930	-1.652497	-0.746688
2	4.605170	4.605170	-0.000000	1.386294	3.218876	2.659260	3.506558
3	3.506558	2.813411	-1.420696	-0.000000	1.771957	1.272966	2.207275
4	1.771957	1.049822	-3.249211	-1.838961	-0.000000	-0.570980	0.342490
5	2.302585	1.609438	-2.719979	-1.311032	0.494296	-0.000000	0.867501
6	1.386294	0.693147	-3.618993	-2.209373	-0.412110	-0.940007	-0.000000

Grafo



Implementación

Implementación

El proyecto se fue desarrollando en entregas parciales a lo largo del semestre:

- **Exploración del problema**

  Al inició se evaluaron paquetes de Python y distintos problemas de optimización, eligiendo el problema de Crypto Trading con CVXPY y NetworkX.

- **Primera aproximación**

  Se decidió resolver el problema con el método de Bellman Ford, el cual se implementó usando Python. Para la primera prueba del método se eligió el dataset conformado por el precio en dólares del top 100 criptomonedas registrado cada 15 minutos en Coin Market Cap API. Obtenido de Kaggle

- **Segunda aproximación**

  Se re-implementó el método de Bellman Ford definiendo los parámetros que existen, y se construyó el pipelines con Kale y monitorearlos con el dashboard de Kubeflow

- **Implementación final**

  Se realizó el perfilamiento del código y se aplicaron técnicas que ayudaron a hacer marginalmente más eficiente el código, e incluso se transformó parte del Código a lenguaje C mediante la librería de Cython

Implementación

```
%%file bf_cython2.pyx
import numpy as np
def bf_negative_cycle_cc(graph, node_ini=None, unsigned int distance_ini=np.inf):

    cdef unsigned int i,n,n_nodes

    assert distance_ini>=1, f"La distancia inicial debe de ser mayor o igual a 1. El parámetro fue igual a {distance_ini}"

    n_nodes = len(graph.nodes)

    if node_ini is not None:
        assert node_ini <= n_nodes, f"El nodo definido es mayor a los del grafo. Deberia de ser menor a {n_nodes}."
        n_nodes = node_ini

    n = n_nodes + 1
    # Remove nan borders inside graph
    edges = [edge for edge in graph.edges().data() if ~np.isnan(edge[2]['weight'])]

    # Add a start node and add zero weighted edges to all other nodes
    for i in range(n-1):
        edges.append((n-1, i, {'weight': 0}))

    # Initialize distances of nodes and predecessors
    # https://codingdeekshi.com/initialize-an-array-in-python/
    distance= [distance_ini ]*n
    distance[n_nodes] = 0
    predecessors = [-1]*n
```

Implementación

```
    for i in range(n):
        x = -1
        for edge in edges:
            if distance[int(edge[0])] + edge[2]['weight'] < distance[int(edge[1])]:
                distance[int(edge[1])] = distance[int(edge[0])] + edge[2]['weight']
                predecessors[int(edge[1])] = int(edge[0])
                x = int(edge[1])
        if x == -1:  # If relaxation is not possible, there is no negative cycle
            return None

    # Identify negative cycle
    for i in range(n):
        x = predecessors[int(x)]
    cycle = []
    v = x
    while True:
        cycle.append(int(v))
        if v == x and len(cycle) > 1:
            break
        v = predecessors[int(v)]

    return cycle.reverse()
```

Perfilamiento

Perfilamiento

Herramienta usada en la comparación	Código antes de Eficientar	Código Eficientado
Time	Tomo 2.65 segundos en correr	Tomo 2.09 segundos en correr
Timeit	2.67 s ± 13.6 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)	2.04 s ± 182 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
cProfile	29979 function calls (29977 primitive calls) in 1.371 seconds	20063 function calls (20062 primitive calls) in 1.089 seconds
line_profiler	Timer unit: 1e-06 s Total time: 4.06207 s	Timer unit: 1e-06 s Total time: 4.12395 s
memory_profiler	memory_usage 120.125	memory_usage 120.1640625

Perfilamiento

Herramienta usada en la comparación	Código antes de Eficientar	Código Eficientado
perf	2.52719 +- 0.00924 seconds time elapsed (+- 0.37%) Estadísticas por core: 2.52596 +- 0.00867 seconds time elapsed (+- 0.34%) Estadísticas generales: - 2575.60 msec task-clock # 1.019 CPUs utilized (+- 0.33%) - 27 context-switches # 0.010 K/sec (+- 2.43%) - 0 cpu-migrations # 0.000 K/sec (+- 58.49%) - 26093 page-faults # 0.010 M/sec (+- 0.04%)	2.22617 +- 0.00621 seconds time elapsed (+- 0.28%) Estadísticas por core: 2.21754 +- 0.00313 seconds time elapsed (+- 0.14%) Estadísticas generales: - 2268.51 msec task-clock # 1.022 CPUs utilized (+-0.25%) - 27 context-switches # 0.012 K/sec (+- 2.33%) - 0 cpu-migrations # 0.000 K/sec (+- 35.04%) - 26132 page-faults # 0.012 M/sec (+- 0.15%)

Compilación en C

Herramienta usada en la comparación	Código antes de Eficientar	Código Eficientado
Cython	Bellman Ford tomó 0.8361616134643555 segundos	Bellman Ford tomó 0.527334451675415 segundos

Retos

Capacidad Computación

el modelo Bellman Ford demanda una gran cantidad de recursos computacionales, debido a que tiene que evaluar en cada uno de los nodos de la red, por lo que redes superiores a 100 nodos, no eran posibles de ejecutar en computadoras personales

Eficientar el código

A pesar de hacer un perfilamiento exhaustivo de nuestro código, solo conseguimos disminuir marginalmente el tiempo de ejecución. Por lo que en realidad no logramos hacer una optimización de código como tal.

Retos

Uso de AWS

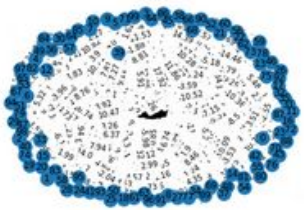
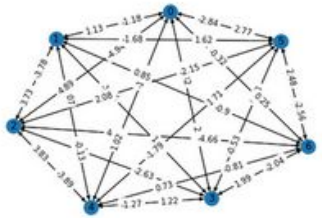
Para el uso de pipelines con Kale y monitoreo con el dashboard de Kubeflow, Se tuvo que configurar una máquina virtual y experimentamos problemas temporales con la ejecución de los servicios en la máquina virtual

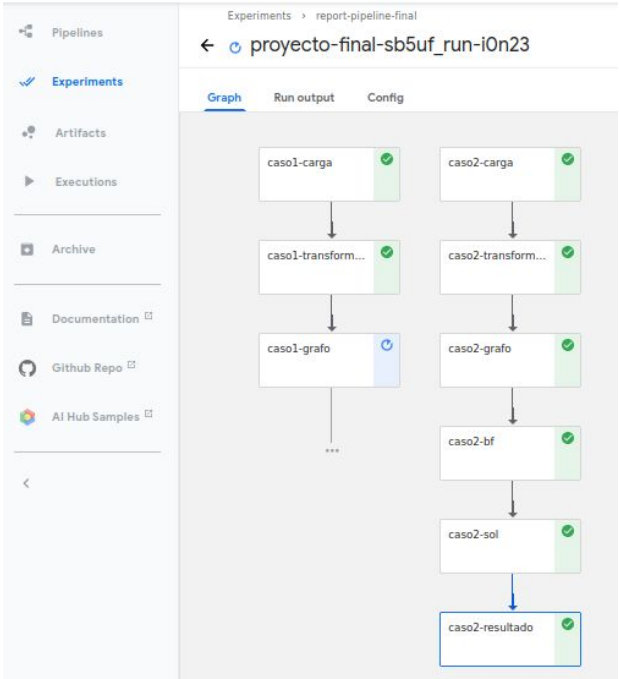
Perfilamiento de CPU

El perfilamiento con PERF fue muy complicado ya que requiere privilegios especiales de administración en los sistemas operativos. al usar Docker no se tenían los permisos y se tuvo que hacer instalación de paquetería a nivel S.O y cambios de configuración.

Resultados

Implementación de casos de uso

	Caso 1	Caso 2
Datos	(100, 2) Símbolo Precio 0 REAPER 0.000903 1 SQUA 5.031241 2 DHR 0.003381 3 GSAIL 0.119018 4 vBUSD 0.021519	Símbolo Precio 0 BCH 263.117533 1 DASH 82.388514 2 EOS 1.930541 3 ETC 26.157788 4 LTC 92.195750 5 NEO 16.032580 6 XMR 198.262798
Grafo		
Lista solución	[99, 98, 99]	[5, 6, 5]
Resultado	Símbolo Precio 99 DTH 0.005611 98 DEL 0.065131 99 DTH 0.005611	Símbolo Precio 5 NEO 16.032580 6 XMR 198.262798 5 NEO 16.032580



Conclusiones

- Se aplicaron diversos conceptos y temas vistos en clase, los cuales se reforzaron de forma práctica en cada etapa del proyecto.
- Se puso en práctica el uso nuevas herramientas como *Kale* y *Kubeflow*.
- Todo este bagaje de herramientas permitió encontrar una solución en el campo de trading, donde una vez identificado el problema de optimización que implicaba, el encontrar los diferentes métodos que están disponibles para la solución del costo mínimo fue más sencillo, y donde se logró seleccionar un método adecuado, como fue el algoritmo *Bellman Ford.*
- El resultado obtenido después de aplicar el perfilamiento y la compilación con C en el código se pasó de 0.836 a 0.527 segundos, fue una pequeña disminución en el tiempo de procesamiento, sin embargo analizando los resultados no fue tan grande la diferencia debido a que el método lo considerábamos rápido desde la primera implementación, ya que cuando se evaluó el método ingresando 500 criptomonedas tardó alrededor de 4 minutos, lo cuál se nos hizo aceptable, por el procesamiento que implicaba.
- Con la disminución de tiempo que se logró hacer en el procesamiento, nos dimos cuenta que se tuvo un aumentó en el uso de memoria, que era de esperarse ya que se sacrifica un recurso por otro.

Referencias

- https://www.bbva.com/es/que-es-trading-que-hace-falta-para-operar/
- https://www.solunion.cl/blog/que-es-y-para-que-sirve-la-tecnologia-blockchain/
- https://www.plus500.com/es-ES/Trading/CryptoCurrencies/What-Is-Cryptocurrency-Trading~3
- https://coinmarketcap.com/es/all/views/all/
- https://www.programiz.com/dsa/bellman-ford-algorithm
- https://www.simplilearn.com/tutorials/data-structure-tutorial/bellman-ford-algorithm
- https://itam-ds.github.io/analisis-numerico-computo-cientifico/5.optimizacion_de_codigo/5.3/Compilacion_a_C.htmll
- https://ocw.unican.es/pluginfile.php/301/course/section/239/tema_02.pdf
- https://es.wikipedia.org/wiki/Problema_del_camino_m%C3%A1s_corto
- https://www.u-cursos.cl/ingenieria/2011/1/IN3701/2/material_docente/bajar?id_material=342208

Gracias

Maestría en Ciencia de Datos - ITAM 2022