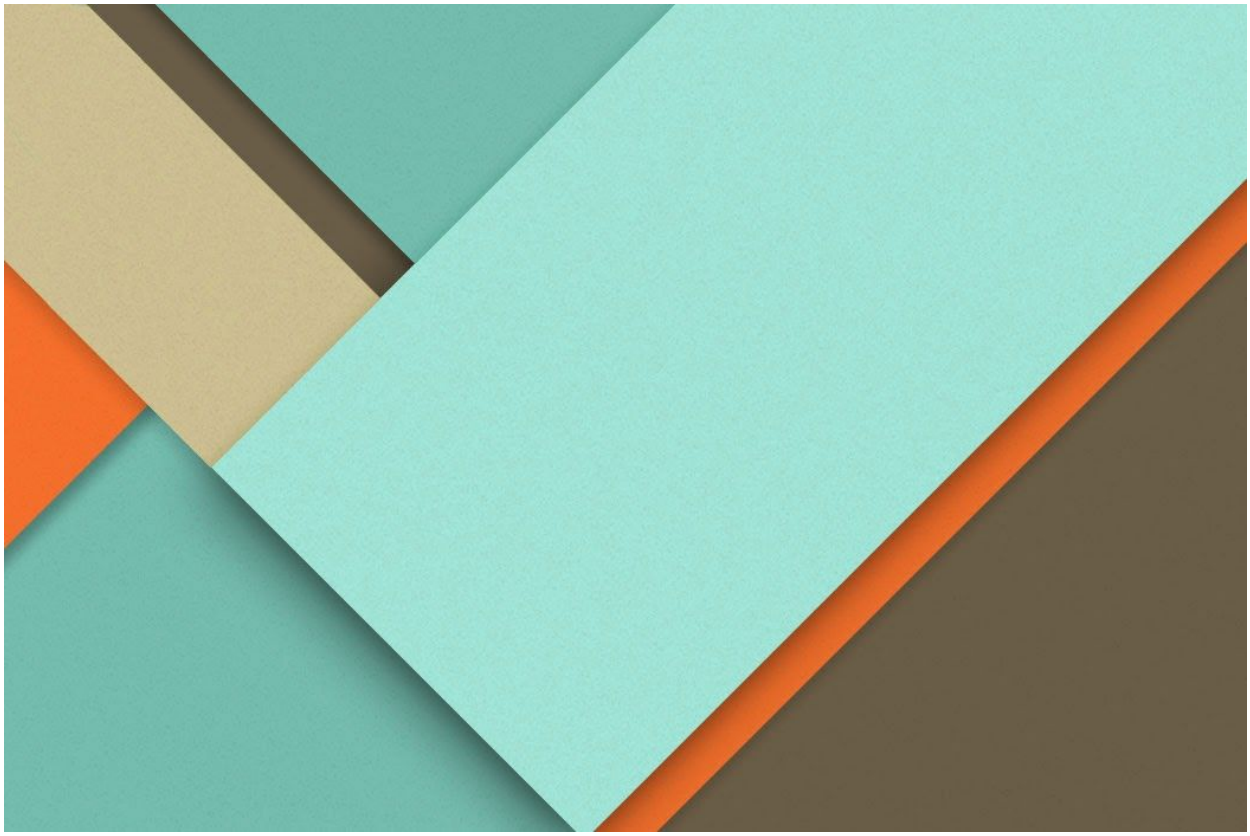


Trabajo de P. Concurrente: Algoritmo Proof Of Work

27/10/2020

Autores:

Nicolas Martinez, legajo: 45328.

Federico Sandoval, legajo: 45955.

Introducción

Dominio

El trabajo que realizamos se basa en un esquema simplificado sobre la forma en que Bitcoin usa el algoritmo Proof of Work. Este consta en calcular una cadena concatenada con alguna combinación de 4 bytes(nonce), cumplan con la dificultad mínima solicitada descrita en el algoritmo `Proof of Work`, donde al concatenar lo antes dicho y aplicar la función de hash SHA-256, de como resultado una secuencia de los primeros n bytes iguales a 0, donde n es la dificultad mínima solicitada. En este trabajo por ser un esquema simplificado se solicitaron las dificultades 2, 3 y 4 como opcional (la cual es la dificultad mínima utilizada en el minado de Bitcoin).

Diseño

Comenzamos por intentar encontrar la forma en que los PowWorkers buscarán el hash solicitado dependiendo la dificultad, pero terminamos complicándonos ya que no comprendemos bien cómo funcionaban los bytes en java.

Decidimos hacer una puesta en común de los ejercicios prácticos de Buffer y ThreadPool para sacar lo mejor de cada uno y corregir ambos para lograr la base de buffer, threadpool, y estructura del powworker. A partir de allí seguimos diseñando el main, donde creamos la interacción con el usuario para que inserte los threads, la dificultad y la cadena deseada para encontrar el nonce que cumpla la dificultad.

A partir de allí, creamos el buffer y modificamos la forma en que se construye un ThreadPool, le agregamos un array de bytes(la cadena deseada pasada a bytes) y la dificultad, ya que como el encargado de generar las unidades de trabajo es el Main, el ThreadPool ya no podría poner a correr con cosas específicas a los PowWorkers, así que se realizó esto para poder almacenar esta información en los powworkers y así buscar los nonces una vez que se pongan a correr, a través de ser creados por el ThreadPool, y consumir las unidades de trabajo, almacenadas en el Buffer creadas por el Main de manera tal que los threads barran los nonces de forma equitativa.

Una vez que logramos que los PowWorkers encontrarán al hash, no podíamos parar el programa, por lo que decidimos que éstos conozcan al ThreadPool que los creó para poder avisarle al mismo cuándo había o bien terminado de trabajar, o bien encontrado el nonce.

El siguiente paso fue ajustar la creación de unidades de trabajo, ya que habíamos tomado un primer diseño propuesto en clases pero finalmente notamos que éste no cumplía con lo solicitado por el enunciado, por lo que procedimos a utilizar algunos cálculos aprendidos

en mate2 sumados a probar y probar hasta que pudimos lograr encontrar las fórmulas para que las unidades de trabajo sean equitativas.

Notamos que la eficiencia en la búsqueda del nonce era considerablemente menor si quitamos el print de cada hash generado por los PowWorkers, por lo que, como ya sabíamos que la funcionalidad estaba cumplida, simplemente decidimos imprimir por consola el nonce y hash que cumplían con la dificultad.

Evaluación

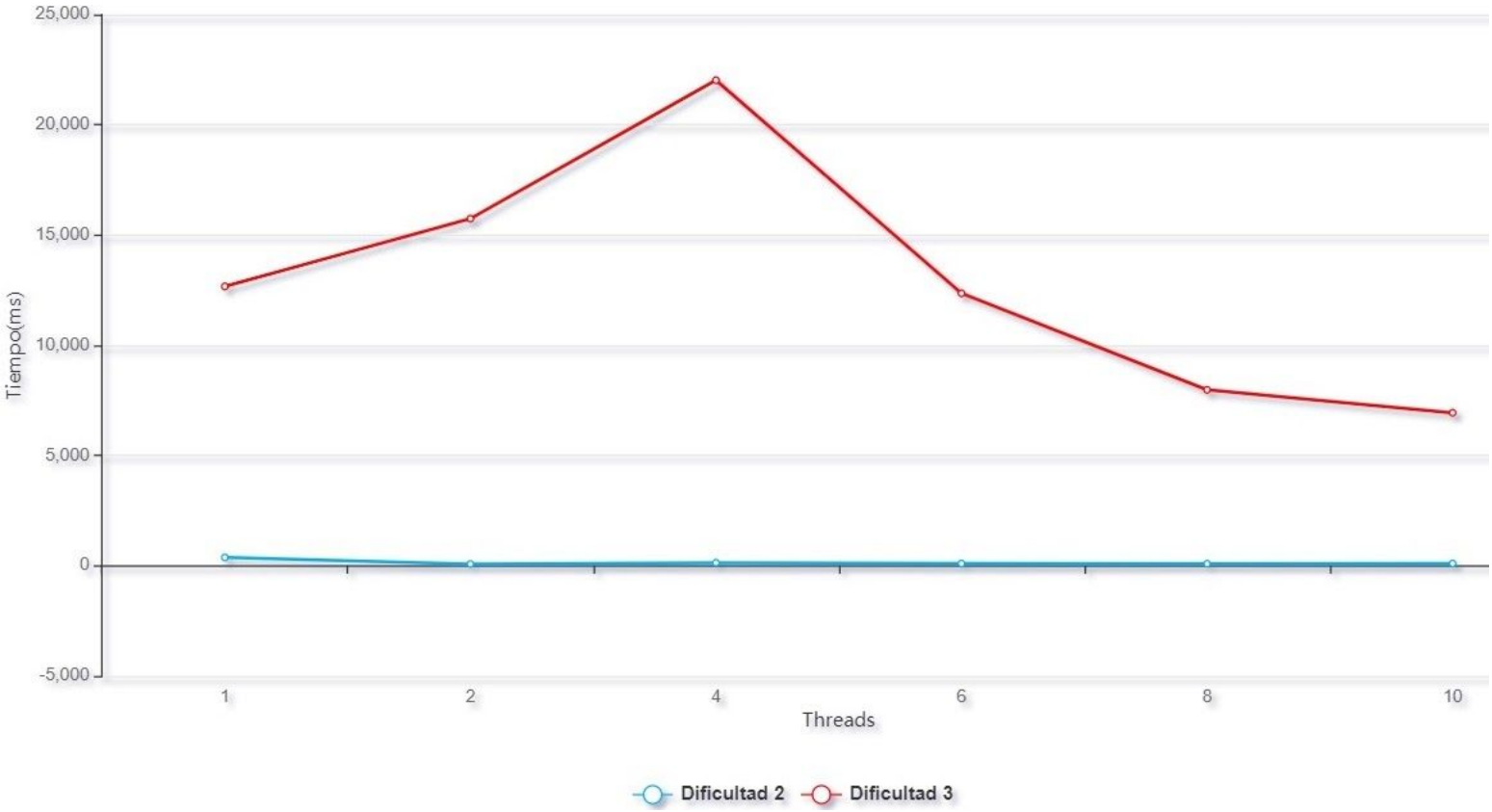
Información del Equipo(Federico Agustin Sandoval):

- Micro: Intel Core i3-6006U 2.0GHz doble núcleo y 4 subprocesos.
- Memoria RAM: 8GB(dos de 4GB).
- SO: Windows 10 Home x64

Comparacion Analitica:

Threads	Dificultad 2	Dificultad 3
1	378	12672
2	79	15745
4	136	22024
6	99	12360
8	93	7980
10	104	6941

Comparación Gráfica:



Informe del Equipo(Nicolas Martinez):

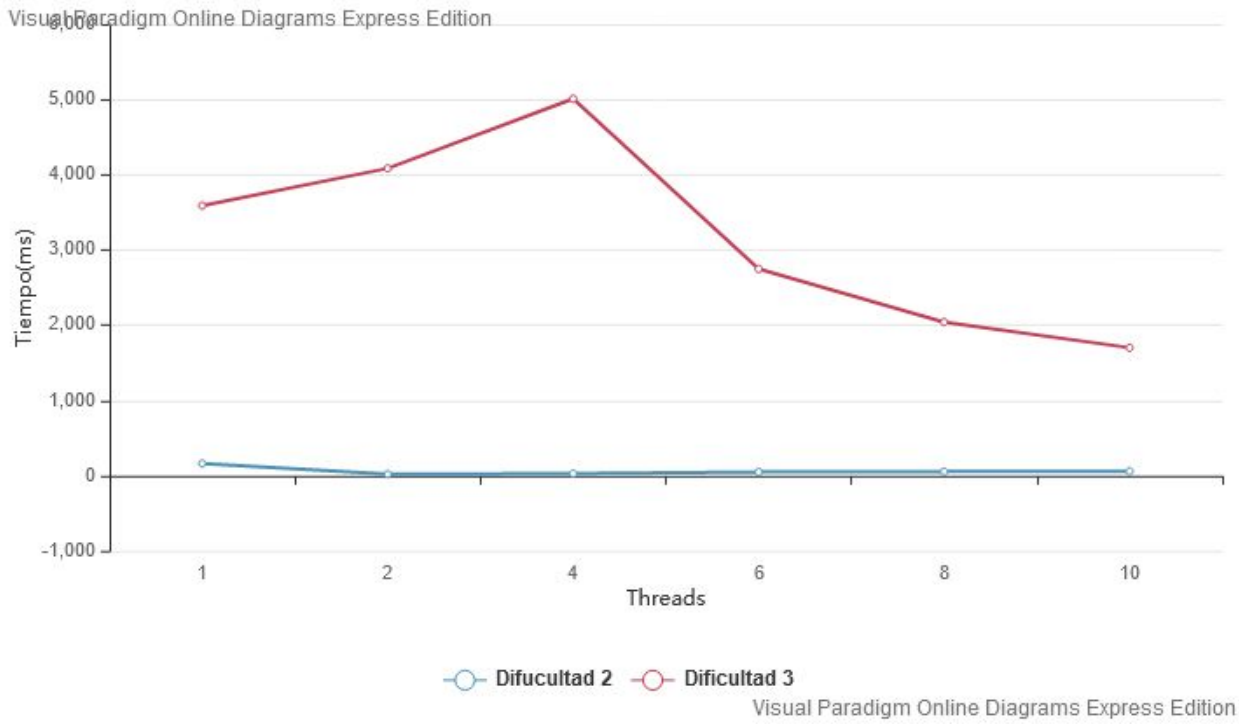
- Microprocesador: Intel i3-9100
- Memoria RAM: 8gb DDR4 2400mhz
- SO: Windows 10 Enterprise x64 bits

Comparacion Analitica:

Threads	Dificultad 2	Dificultad 3
1	166 ms	3590 ms
2	25 ms	4085ms
4	33 ms	5009 ms

6	53 ms	2749 ms
8	59 ms	2044 ms
10	63 ms	1704 ms

Comparación Gráfica:



Análisis

Como puede apreciarse en el primer gráfico, el mejor tiempo de búsqueda se da al utilizar diez threads. Al ver esta particularidad decidimos hacer las pruebas en un segundo equipo pero, como se aprecia en el segundo gráfico, los 10 threads seguían teniendo la mejor eficiencia ante la dificultad 3. Así que esto es probablemente debido a que al ser más threads hay unidades de trabajo más pequeñas para barrer por thread y esto genera que el nonce se encuentre más rápidamente.

Nos sorprendió que el peor tiempo de búsqueda de nonces con dificultad 3 fuera de 4 threads, ya que pensábamos que con los resultados de la dificultad 2, la búsqueda con un thread era el que más iba a tardar en encontrar el nonce. Esto nos dejó desconcertados porque pensábamos que luego ver ese resultado, a medida que los threads fueran

incrementando, la búsqueda comenzaría a tardar más, y que el más ineficiente fuera el de 10 threads, pero este ni siquiera es el peor en dificultad 2, por lo que fue un giro inesperado y al debatir el por qué sucedía esto pudimos concluir que la distribución de las unidades de trabajo para 4 threads dejan al nonce más cercano de la respuesta en una posición alejada, cercana al límite de alguna unidad de trabajo, por lo que tardarían más en chequear los nonces hasta llegar al buscado, y que, luego de eso, para 6, 8 y 10 threads, la partición de las unidades dejarán a la respuesta mucho más cercana al inicio de una unidad de trabajo para ser barrida rápidamente y encontrar el nonce que cumpla con la dificultad buscada. Un dato interesante es que probamos con 5,7,9 y 11 threads en dificultad 3, y estos mejoraron los tiempos considerablemente en comparación con los threads solicitados: *3 threads en 13780 ms, 5 threads en 3771 ms, 7 threads en 2544 ms, 9 threads en 4219 ms y 11 threads en 5056 ms.* Con estos resultados pudimos confirmar que la obtención del nonce no depende de que a más threads trabajando, más rápida la obtención del nonce, sino que lo que más importa es el cómo quedan distribuidas las **unidades de trabajo**, ya que una buena distribución generaría que un nonce buscado quede lo más cercano al inicio de un rango, y una mala distribución que queden al final del mismo, haciendo que se tarde más tiempo en la búsqueda del nonce que tenga la respuesta.

El resultado de buscar el *golden nonce* con 10 threads, los cuales fueron los más óptimos al buscar el nonce con dificultad 3, fue:

```
Cantidad de threads:
10
Dificultad (max 4):
4
Cadena:

No pudimos encontrar el nonce con la dificultad 4
Tiempo total de busqueda: 2106963 ms
```

Esto nos dejó un sabor amargo al no poder encontrarlo, pero el intento estuvo y eso fue lo divertido. Pudimos observar el cómo se comportaba nuestro código al no encontrar un resultado nos incomodamos porque tardaba mucho pero al dar un resultado entendimos que los threads barrieron todos los nonces posibles y no encontraron el *golden nonce*.

Sin embargo, debido a que no pudimos hallarlo con la cadena vacía, intentamos buscar el golden nonce del prefijo “Hola”, el *golden nonce* aparece, lo cual nos alegró y confirmó que nuestro trabajo estaba bien hecho y simplemente para la cadena vacía no existe una solución para la dificultad 4.

```
Cantidad de threads:
10
Dificultad (max 4):
4
Cadena:
"Hola"
[FOUND]: [0, 0, 0, 0, 43, 89, 25, -27, 35, 3, 7, 26, -32,
[WITH] : [-89, -98, -28, 9]

Tiempo total de busqueda: 196988 ms
```

Como conclusión podemos decir que en la presentación del trabajo en la clase parecía sencillo pero tuvimos que hacer varias investigaciones para nutrirnos de información como el manejo de bytes y el cómo hackear la cadena dada con el nonce, que a su vez los primeros días resultó muy tedioso de hacer al no entender muy bien la teoría sobre la cual estaba fundada el trabajo. Sin embargo, a medida que comenzamos a destrabar los nudos y solucionar los problemas, el trabajo se volvió bastante grato de realizar y ver el cómo los PowWorkers encontraban las soluciones. Al probar el *golden nonce* pensábamos que estábamos haciendo las cosas mal, aunque como ya se vió, para la cadena vacía no existe un nonce para dificultad 4. Pero por otro lado, los threads pasaron de buscar en dificultad 3 en 5 segundos al nonce, a casi 10 minutos en encontrar el golden nonce(para la cadena “Hola), fue algo muy impactante el escalado de tiempo que se hizo para poder encontrarlo, por lo que se puede confirmar que a medida que se va incrementando la dificultad, la cantidad de nonces que la satisfacen se reducen considerablemente. Terminó por ser una experiencia divertida de realizar en equipo el ejercicio de hacer las comparaciones gráficas y el poder debatir sobre el diseño y las posibles soluciones al enunciado.