Informe:

Integrante : Nelson Gabriel González

email: nelgonzalez88@gmail.com

legajo:39915

Introducción:

Para resolver el problema fue necesario el uso de las siguientes clases:

\*Buffer: que actúa como una cola FIFO concurrente de capacidad de 2, dispone de un msj pop y un msj add con el que podemos administrar el ingreso y egreso de las unidades de trabajo.

El método pop tiene dentro de su lógica un wait, que si la cola esta vacía, se queda esperando el ingreso de una unidad de trabajo para poder retornarla, una vez que dispone de una unidad de trabajo para retornar hace un notifyAll con que el que avisa que ya retorno su unidad de trabajo, quedado la cola a la espera de una nueva unidad de trabajo. Así mismo el método add tiene un wait que cuando la cola está llena realiza el mismo procedimiento pero en sentido inverso. Vale aclarar que la cola es de tamaño 2 ya que la unidad de trabajo tiene un valor de inicio y un valor de final.

\* ThreadPool: Esta clase es la encargada de instanciar a los PowWorker, esta clase recibe como parámetro la cantidad de PowWorker, la cadena que luego se concatenara con cada valor del nonce, el buffer y la dificultad que es la cantidad de bytes en cero que deberá tener el hash. Dispone de un array de PowWorker que son inicializados en su constructor, estos comienzan su ejecución mediante el método run donde mediante un for que recorre dicho array llamando al método start de cada Pow. Tiene un método getArrayByteToHexa con el que puede mostrar en pantalla un array de bytes en hexa, este msj lo usamos para mostrar por pantalla el hash que cumple con la condición. Tiene un método pasarUnSegundo() que lo llamamos en el main cada vez que pasa un segundo, esto lo logramos con un cronometro implementado con la clase TimerTask. Dispone de un método encontreNoce() el cual es llamado cuando algunos de los pow encuentra el nonce, este método avisa a todos los pow que el nonce ya fue encontrado para que el pow pueda parar su búsqueda.

\* PowWorker: Esta clase extiende de Thread es la encargada de realizar la búsqueda del nonce. Recibe en su parámetro una cadena que luego será concatenada con el nonce, un id que se lo asigna su Threadpool, un buffer, su ThreadPool y la dificultad de búsqueda. Su método principal es run()que lo Hereda de su de súper clase, en este método se realiza el proceso de búsqueda, primero toma el inicio y el final del barrido de nonce que le corresponde del buffer, llama a su método buscar pasándole como parámetro el inicio y el final del barrido. Dentro de este método se realiza un for de todos los nonce posibles, en cada iteración concatena la cadena y el valor de nonce, sacando el hash y verifica que cumpla la dificultad solicitada. Si encuentra un nonce que cumpla la dificultad le avisa a su ThreadPool. En el caso de no encontrarlo manda un mjs a pantalla que no fue posible la búsqueda y se detiene.

\*Main: Esta clase además de correr el programa, se encarga de hacer de productor del buffer, generando las unidades de trabajo. También cuenta con un Timer que es el encargado de contar los segundos que tardan en encontrar el nonce

Evaluación:

Las pruebas fueron ejecutadas en un procesador Intel Core I5-3210 2.50GHZ con 8Gb de ram, Windows seven 64 bits.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| dificultad | cantThread | 1 | 2 | 4 | 6 | 8 | 10 |
| 2 | | 1 seg | 1 seg | 1 seg | 1 seg | 1 seg | 1 seg |
| 3 | | 27 seg | 24 seg | 5 seg | 25 seg | 9 seg | 10 seg |

Dificultad 2

Dificultad 3

Analisis:

Dado la maquina donde se corrió en la dificultad 2 no se nota la diferencia en el tiempo de la búsqueda. En la dificultad 3 se hace más notorio el tiempo en el que los pow encuentran el nonce. con 1 thread se obtiene la menor eficiencia. Se podría pensar que la mayor eficiencia se encontrara con mayor cantidad de thread ya que hay mas procesos corriendo al mismo tiempo barriendo mas valores en menor cantidad de tiempo, pero el grafico nos muestra que con 4 thread se encontró la mayor eficiencia, aun mas que con 6, 8 y 10 thread. ¿A qué se debe esto?, en el momento que se le asigno el barrido a cada thread, el comienzo de alguno de ellos quedo muy próximo al nonce y gracias a eso se encontró mas rápido el resultado.

El golden noce lo encontró después de 577 seg y como es de esperar con mayor dificultad el tiempo tardado es mayor