“Backtracking”

Profesor/es:

Cuadrado Estrebou, Maria Fernanda

Foglino, Alejandro Luis

Integrantes:

Tomas Gonzalez Humphreys – LU: 1175809

Nicolas Blanco – LU: 1175783

Alberto Gallego – LU: 1173456

**El resto de participantes informaron que no participan más de la cursada.*

Buenos Aires, 8 de Noviembre de 2024.-

Tabla de Contenidos

Introducción

El presente trabajo práctico se centra en la implementación de un sistema de blockchain, donde se organizan y distribuyen transacciones dentro de bloques bajo ciertas restricciones. El objetivo es construir una estructura de blockchain en la que se agrupen transacciones respetando limitaciones en el tamaño, valor y cantidad máxima de transacciones por bloque, contemplar posibles dependencias entre transacciones y se cumpla una prueba de trabajo básica. Este proyecto simula aspectos clave del funcionamiento de una blockchain, aplicando principios de backtracking para resolver problemas de agrupamiento y validación de transacciones.

Descripción del Problema

Dado un conjunto de transacciones con distintos atributos (tamaño, valor y posibles dependencias), se busca construir todas las combinaciones posibles de blockchain válidas. Cada blockchain se compone de uno o más bloques que deben cumplir las siguientes restricciones:

1. **Tamaño máximo del bloque**: La suma del tamaño de todas las transacciones en un bloque no debe superar el valor especificado.
2. **Valor máximo del bloque**: La suma del valor de todas las transacciones en un bloque no debe exceder el valor máximo permitido.
3. **Número máximo de transacciones por bloque**: Cada bloque puede contener un número limitado de transacciones.
4. **Prueba de trabajo**: La suma de los valores de las transacciones en cada bloque debe ser divisible por 10.
5. **Dependencias entre transacciones**: Algunas transacciones dependen de otras, por lo que no pueden incluirse en un bloque hasta que las transacciones de las que dependen ya hayan sido añadidas en bloques anteriores de la misma blockchain.

El desafío consiste en implementar un algoritmo que explore todas las combinaciones posibles de transacciones en bloques, asegurando que todas las restricciones sean cumplidas y generando múltiples configuraciones válidas de blockchain a partir del conjunto de transacciones inicial.

Estrategia de Resolución

Para resolver este problema, se ha empleado un enfoque de **backtracking** que permite explorar de manera exhaustiva todas las combinaciones de transacciones en bloques, retrocediendo y probando distintas configuraciones cuando una combinación no cumple con las restricciones establecidas. La estrategia general se estructura de la siguiente manera:

Primeramente, vamos a considerar cada transacción en cada posición de los bloques como opción.

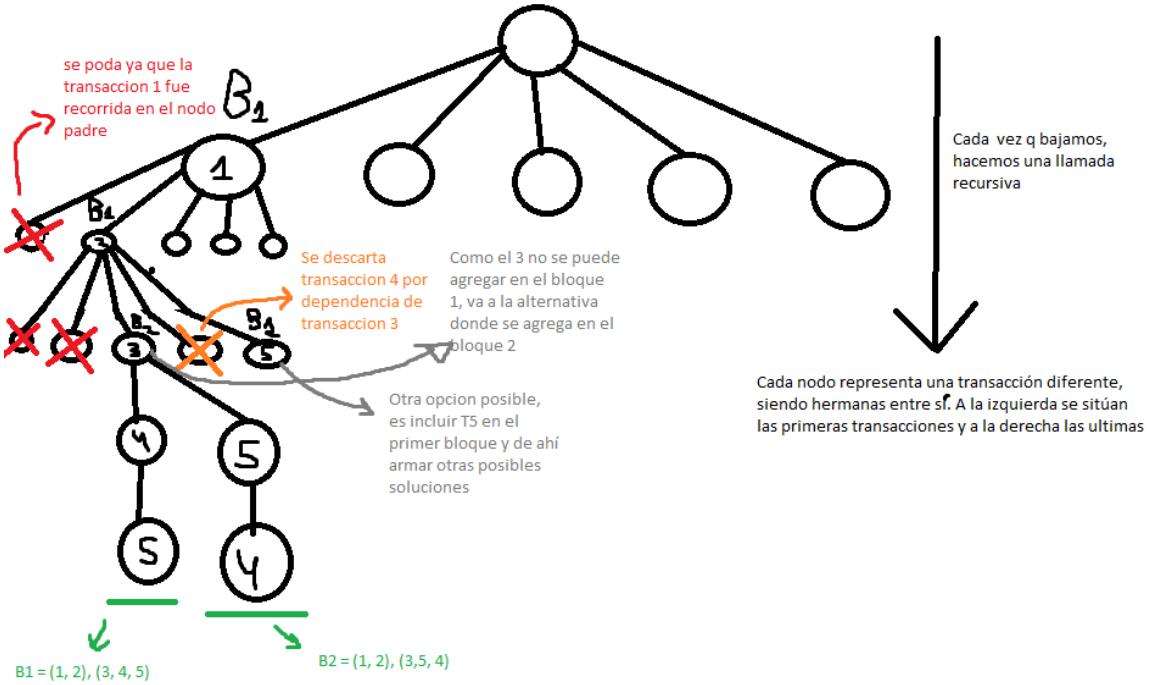


Figura 1.1: árbol "incompleto" de alternativas exploradas por el algoritmo

En la figura 1.1, se muestra el camino tomando como opción la transacción 1 en el primer bloque. De ahí, se baja recursivamente y se exploran las alternativas. Dentro de las condiciones de poda, se encuentra:

- Si la transacción ya fue tratada en un nivel anterior, se descarta
- Si existe una dependencia de la transferencia, y esta transferencia de la que depende no fue agregada, se descarta la opción

- Si se supera el valor máximo de un bloque al agregar esta transferencia, se descarta
- Si agregar la transferencia supera el tamaño permitido por bloque, se descarta

Utilizaremos el ejemplo dado en el pdf de la consigna para explicar el algoritmo

Transacción	Tamaño (Kb)	Valor (satoshis)	Dependencias	Firmas
T1	200	30	Ninguna	2
T2	150	40	T1	1
T3	300	50	Ninguna	2
T4	100	20	T3	1
T5	250	30	Ninguna	2

Figura 1.2: conjunto de transacciones

En la figura 1.1 y 1.2 se ve cómo descarta T4 en el nivel 3 ya que depende de T3, y cómo va recorriendo distintas alternativas según las condiciones.

Construcción de la blockchain mediante backtracking:

El algoritmo comienza creando bloques vacíos y, de manera recursiva, intenta agregar cada transacción en los bloques existentes o en nuevos bloques, según las restricciones de tamaño, valor y cantidad de transacciones.

Para cada transacción, se verifica que cumpla con las restricciones del bloque en el que se intenta colocar. Además, se asegura que todas las transacciones de las que depende ya hayan sido incluidas en bloques anteriores.

Verificación de condiciones y generación de soluciones:

Cada vez que se logra distribuir todas las transacciones en bloques cumpliendo todas las restricciones, se almacena esa combinación como una blockchain válida.

Si una combinación no cumple con las condiciones, el algoritmo hace backtracking, retrocediendo y probando otra opción para asegurar que todas las combinaciones posibles sean exploradas.

Se revisa que todas las transacciones de las que depende una transacción actual ya estén presentes en los bloques previos, respetando así las dependencias entre transacciones.

Pseudocódigo del Algoritmo de Resolución del Problema

ALGORITMO construirBlockchain

Entradas: T: Lista<Transaccion>, maxTamanioB: entero, maxValorB: entero, maxTransaccionesPorB: entero

Salid

a: S: Lista<Lista<Bloque>>

S <-- InicializarLista //lista de lista de bloques

BC <-- InicializarLista //lista de bloques

backtrackingBlockhain(T, 0, S, BC, maxTamanioB, maxValorB, maxTransaccionesPorB)

DEVOLVER S

FIN ALGORITMO

ALGORITMO backtrackingBlockhain

Entradas: T: Lista<Transaccion>,

indice: entero,

S: Lista<Lista<Bloque>>,

BC: Lista<Transaccion>,

maxTamanioB: entero,

maxValorB: entero,

maxTransaccionesPorB: entero

```
SI indice == longitud(T) //O(n)
        AgregarASolucion(S, BC)

    SINO
        PARA cada Transaccion en T
            SI NO blockchainVacia(BC)
                PARA cada Bloque en BC
                    SI transaccionValidaEnBloque(BC, Bloque, Transaccion, maxTamanioB, maxValorB, maxTransaccionesPorB) //O(c)
                        SacarDeBlockchain(BC, Bloque)
                        AgregarABloque(Bloque, Transaccion)
                        AgregarABlockchain(BC, Bloque)
                        backtrackingBlockhain(T, indice + 1, S, BC, maxTamanioB, maxValorB, maxTransaccionesPorB) //a = n * c^2, b = 1, k = n
                        SacarDeBloque(Bloque, transaccion)
                    FIN SI
                FIN PARA
            FIN SI
        FIN PARA

        nuevoBloque <-- InicializarBloque
```

```
            transaccionActual <-- sacarTransaccionLista(transacciones,
indice)

            SI transaccionValidaEnBloque(BC, Bloque, transaccionActual,
maxTamanioB, maxValorBloque, maxTransaccionesPorB)

                  AgregarABloque(nuevoBloque, transaccionActual)

                  AgregarABlockchain(BC, nuevoBloque)

                  backtrackingBlockhain(T, indice + 1, S, BC,
maxTamanioB, maxValorB, maxTransaccionesPorB)

                  SacarDeBloque(nuevoBloque, transaccionActual)

            FIN SI


      FIN SI

FIN ALGORITMO
```

ALGORITMO transaccionValidaEnBloque

Entradas: BC: Lista<Bloque> B: Bloque, Tr: Transaccion, maxTamanioB: entero, maxValorB: entero, maxTransaccionesPorB: entero

Salida: verdadero o falso

```
      SI tamanioBloque(B) + tamanioTransaccion(Tr) > maxTamanioB

            DEVOLVER falso
```

```
PARA cada Bloque en BC
    PARA cada Transaccion en Bloque
        SI Transaccion == Tr
            DEVOLVER falso
        FIN SI
    FIN PARA
FIN PARA

SI valorBloque(B) + valorTransaccion(Tr) > maxValorB
    DEVOLVER falso
SI cantTransaccionesBloque(B) + 1 > maxTransaccionesPorB
    DEVOLVER falso
SI (valorBloque(B) + valorTransaccion(Tr)) % 10 <> 0
    DEVOLVER falso
Si dependencia(Tr) <> null
    encontrado <-- falso
    PARA cada Bloque en BC //O(c)
        SI existeTransaccionEnBloque(Bloque, dependencia(Tr))
            encontrado <-- verdadero

    SI encontrado == falso
        DEVOLVER falso
```

DEVOLVER estaFirmada(Tr) //verifica que la transaccion tenga las suficientes firmas

FIN ALGORITMO

Análisis de Complejidad Temporal

Complejidad Temporal Teórica

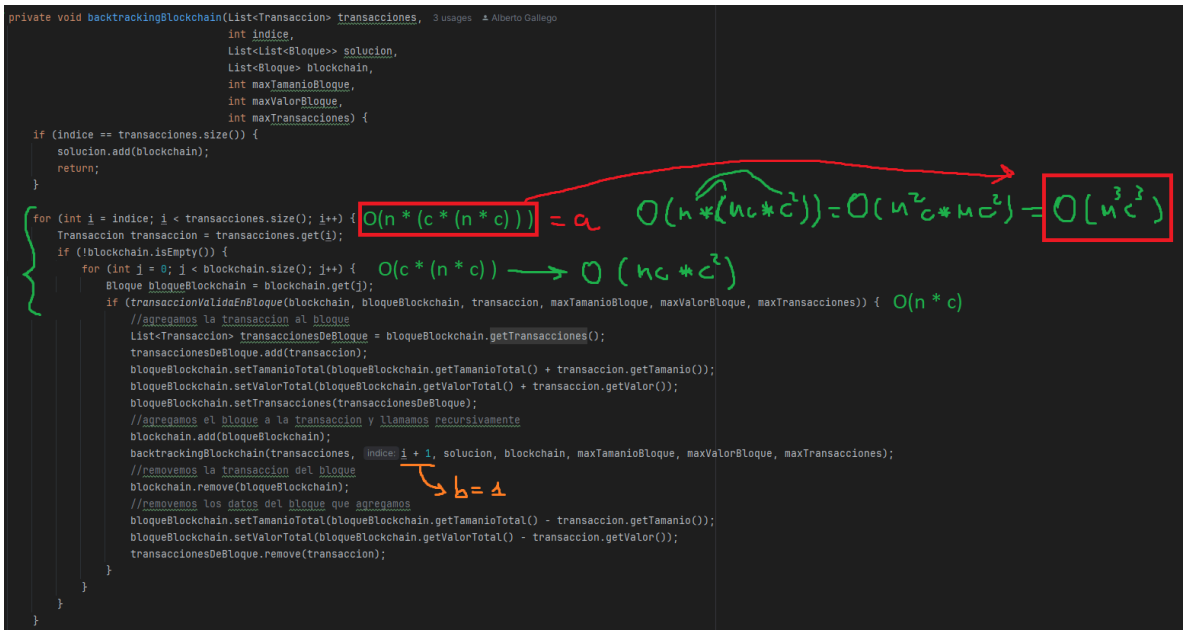
En este caso, nos presentamos ante llamadas recursivas del tipo sustracción, donde por cada llamada iremos “sacando” una transacción. Vamos a especificar a, b y k de la siguiente forma:

```java
public static boolean transaccionValidaEnBloque(List<Bloque> BC,  2 usages  Alberto Gallego
                                                Bloque B,
                                                Transaccion Tr,
                                                int maxTamanioB,
                                                int maxValorB,
                                                int maxTransaccionesPorB) {
    // Verificar el tamaño del bloque
    if (B.getTamanioTotal() + Tr.getTamanio() > maxTamanioB) {
        return false;
    }

    // Verificar el valor total del bloque
    if (B.getValorTotal() + Tr.getValor() > maxValorB) {
        return false;
    }

    // Verificar la cantidad de transacciones en el bloque
    if (B.getTransacciones().size() + 1 > maxTransaccionesPorB) {
        return false;
    }

    // Verificar si el valor total del bloque más el de la transacción es múltiplo de 10
    if ((B.getValorTotal() + Tr.getValor()) % 10 != 0) {
        return false;
    }

    // Verificar dependencia
    if (Tr.getDependencia() != null) {
        boolean encontrado = false;

        for (Bloque bloque : BC) {
            for (Transaccion transaccion : bloque.getTransacciones()) {
                if (transaccion.equals(Tr.getDependencia())) {
                    encontrado = true;
                    break;
                }
            }
        }
```

costo total

O(c * n)

O(n)



```
private void backtrackingBlockchain(List<Transaccion> transacciones,  3 usages  Alberto Gallego
                                    int indice,
                                    List<List<Bloque>> solucion,
                                    List<Bloque> blockchain,
                                    int maxTamanioBloque,
                                    int maxValorBloque,
                                    int maxTransacciones) {
    if (indice == transacciones.size()) {
        solucion.add(blockchain);
        return;
    }

    for (int i = indice; i < transacciones.size(); i++) {
        Transaccion transaccion = transacciones.get(i);
        if (!blockchain.isEmpty()) {
            for (int j = 0; j < blockchain.size(); j++) {
                Bloque bloqueBlockchain = blockchain.get(j);
                if (transaccionValidaEnBloque(blockchain, bloqueBlockchain, transaccion, maxTamanioBloque, maxValorBloque, maxTransacciones)) {
                    //agregamos la transaccion al bloque
                    List<Transaccion> transaccionesDeBloque = bloqueBlockchain.getTransacciones();
                    transaccionesDeBloque.add(transaccion);
                    bloqueBlockchain.setTamanioTotal(bloqueBlockchain.getTamanioTotal() + transaccion.getTamanio());
                    bloqueBlockchain.setValorTotal(bloqueBlockchain.getValorTotal() + transaccion.getValor());
                    bloqueBlockchain.setTransacciones(transaccionesDeBloque);
                    //agregamos el bloque a la transaccion y llamamos recursivamente
                    blockchain.add(bloqueBlockchain);
                    backtrackingBlockchain(transacciones,  indice: i + 1, solucion, blockchain, maxTamanioBloque, maxValorBloque, maxTransacciones);
                    //removemos la transaccion del bloque
                    blockchain.remove(bloqueBlockchain);
                    //removemos los datos del bloque que agregamos
                    bloqueBlockchain.setTamanioTotal(bloqueBlockchain.getTamanioTotal() - transaccion.getTamanio());
                    bloqueBlockchain.setValorTotal(bloqueBlockchain.getValorTotal() - transaccion.getValor());
                    transaccionesDeBloque.remove(transaccion);
                }
            }
        }
    }
```

En la primera imagen, nuestro método privado transaccionValidaEnBloque nos muestra que tiene un costo de O(nc), donde n son la cantidad de transacciones posibles y c son la cantidad de bloques posibles.

En la segunda imagen, se muestra como esta función se repite en un ciclo for c cantidad de veces, dentro de otro ciclo for que itera n cantidad de veces. Esto nos da como resultado de a = **n^3*c^3**

b = 1, ya que cada llamada recursiva aumenta el índice + 1

k = 0, considerando que los métodos, getters y setters de los objetos transacción, bloque y listas son de costo constante (de no ser así, no encontramos forma de verificar estos costos)

$$T(n) \in \begin{cases} \Theta(n^k) & \text{si } a < 1 \\ \Theta(n^{k+1}) & \text{si } a = 1 \\ \Theta(a^{n \text{ div } b}) & \text{si } a > 1 \end{cases}$$

Utilizando el siguiente teorema para funciones recursivas de sustracción, nos encontramos con un costo final de:

O((n^3*c^3)^n)

Conclusiones

La implementación del algoritmo de blockchain presentada en la clase del Algoritmo de Blockchain representa un enfoque sólido y estructurado para construir una cadena de bloques a partir de un conjunto de transacciones. Mediante el uso de un enfoque de backtracking, el algoritmo explora exhaustivamente todas las combinaciones posibles de bloques y transacciones, así se asegura que cada bloque cumpla con los criterios establecidos, como el tamaño máximo, el valor total y el número de transacciones permitidas.

No obstante, su implementación actual presenta limitaciones importantes, principalmente en términos de eficiencia y escalabilidad. El alto costo temporal observado restringe su viabilidad en escenarios con grandes conjuntos de datos o aplicaciones del mundo real. Esto destaca la necesidad de optimizar tanto la lógica de backtracking como la estructura general del algoritmo, buscando estrategias que permitan reducir la cantidad de combinaciones exploradas sin comprometer la validez de las soluciones. Además, se requiere una revisión exhaustiva de su comportamiento en diversos escenarios, lo cual podría realizarse mediante pruebas más amplias y métricas que evalúen su desempeño.

Adicionalmente, es crucial que la implementación sea mejor documentada. La ausencia de comentarios claros y explicaciones detalladas de cada método puede dificultar la comprensión, el mantenimiento y la evolución del código. Una buena práctica sería incluir documentación que detalle el propósito de cada parte del algoritmo, los parámetros utilizados y los resultados esperados. Esto no solo facilitará el trabajo de otros desarrolladores, sino que también ayudará a identificar áreas de mejora y a garantizar que el algoritmo sea fácilmente extensible y adaptable.

En resumen, aunque el algoritmo logra alcanzar los resultados esperados asegurando la generación de soluciones válidas, su eficiencia y escalabilidad son aspectos a mejorar. La implementación actual es prometedora, pero para ser verdaderamente robusta y efectiva en el largo plazo, requiere optimizaciones técnicas, una documentación más completa y pruebas exhaustivas que validen su comportamiento bajo diferentes condiciones. Esto abriría la puerta a una mayor aplicabilidad y utilidad en escenarios prácticos de blockchain.

Bibliografía

- *Blockchain: la nueva tecnología para la confianza | SAP*. (s. f.). SAP. https://www.sap.com/latinamerica/products/artificial-intelligence/what-is-blockchain.html#:~:text=El%20blockchain%20es%20un%20registro,que%20es%20inmutable%20o%20inmodificable