FACULTAD DE INGENIERÍA Y CIENCIAS EXACTAS

DEPARTAMENTO DE TECNOLOGÍA INFORMÁTICA

Trabajo Practico Obligatorio

“Backtracing”

Profesor/es:

Cuadrado Estrebou, Maria Fernanda

Foglino, Alejandro Luis

Grupo:

Ares, Juan Manuel – LU: 1176654

Borassi, Ignacio Ezequiel – LU: 1173963

Ferrari, Juan Ignacio – LU: 1176815

Rupcic, German – LU: 1173030

Buenos Aires, 03 de Noviembre de 2024.-

Tabla de Contenidos

Introducción

El siguiente trabajo consta en la implementación de un algoritmo que construya una blockchain válida, considerando todas las combinaciones posibles de bloques. Cada bloque debe cumplir con un conjunto más amplio de reglas, que incluyen validaciones adicionales como la dependencia entre transacciones, transacciones con firmas múltiples, priorización y restricciones de prueba de trabajo*

** una blockchain es una estructura de datos distribuida que garantiza la integridad de las transacciones mediante reglas criptográficas y consensos entre los nodos*

Descripción del Problema

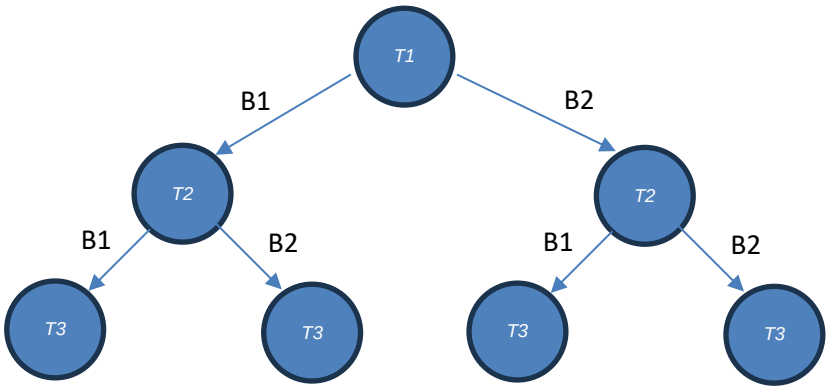
Estrategia de Resolución

Técnica: Backtracking

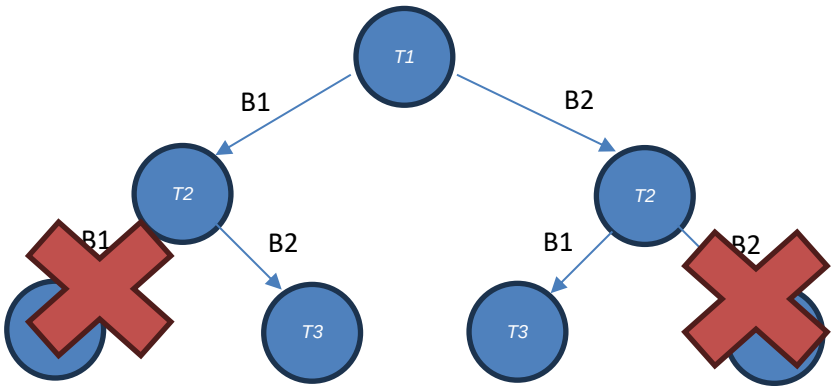
Estrategia:

- *Por nivel: Transacción*
- *Opciones a evaluar por nivel: en que bloque asigno la transacción evaluada en el nivel.*
- *Poda: si la solución actual se pasa en el valor máximo por bloque o el máximo de transacciones por bloque o el tamaño máximo de bloque con respecto a la "Mejor Solución" no continuo en profundidad.*

  Si se arma un árbol completo del ejemplo para Transacciones = {T1, T2, T3} y Bloques = {B1, B2} el árbol será el siguiente:



Si se utiliza la poda el árbol quedaría de la siguiente forma siendo
T1 : {tamaño: 200kb, valor: 30, firmas: 2, dependencia: -}
T2 : {tamaño: 150kb, valor: 40, firmas: 1, dependencia: -}
T3 : {tamaño: 300kb, valor: 50, firmas: 2, dependencia: T1}
Limitaciones: {tamaño: 500kb, valor: 80, firmas: 3}
:



EXCEDE VALOR Y TAMAÑO TOT PERMITIDO EN BLOQUE 1

EXCEDE VALOR TOT PERMITIDO EN BLOQUE 2

Pseudocódigo del Algoritmo de Resolución del Problema

Algoritmo: AsignarTransacciones

Entrada: transacciones: Vector <objeto>, bloquesActuales: Vector <objeto>, reglas: objeto, mejorSolucion: Vector <objeto>, cantidadTransacciones: entero, nroTransaccion: entero

Salida: -

```
Entero i = 0;
Mientras i < cantidadTransacciones/reglas.cantMaxTransacciones
       agregarTransaccion(bloquesActuales, i, transaccion[nroTransaccion])

       si (nroTransaccion = longitud(transacciones) - 1)

              si cumpleReglas(bloquesActuales, reglas) y
maxEspacio(bloquesActuales) < maxEspacio(mejorSolucion) y
maxValor(bloquesActuales) < maxValor (mejorSolucion)

                     mejorSolucion = bloquesActuales

              fin si

       sino

              si cumpleReglas(bloquesActuales, reglas) y
maxEspacio(bloquesActuales) < maxEspacio(mejorSolucion) y
maxValor(bloquesActuales) < maxValor (mejorSolucion)

                     AsignarTransacciones(transacciones, bloquesActuales, reglas,
mejorSolucion, cantidadTransacciones, nroTransaccion +1)

              Fin si

       fin si

       removerTransaccion(bloquesActuales, i, transaccion[nroTransaccion])

       i = i + 1

fin mientras
```

O(p^n)

a = p; b= 1; k = 1;

Algoritmo: cumpleReglas

Entrada: bloques: Vector <objeto>, reglas: objeto

Salida: resultado: booleano

```
Entero i = 0;
Entero resultado = true;
Mientras i < longitud(bloques)
	Si bloques[i].espacioOcupado > reglas.espacioMaximo o bloques[i].valor >
reglas.valorMaximo o bloques[i].cantidadTransacciones >
reglas.cantMaxTransacciones o !cumplePruebaTrabajo(bloques[i])
		Resultado = false;
	fin si
	i = i + 1
fin mientras
devolver resultado
```

O(n)

Siendo n la cantidad de bloques

Algoritmo: maxEspacio

Entrada: bloques: Vector <objeto>

Salida: resultado: entero

```
Entero i = 0;
Entero resultado = 0;
Mientras i < longitud(bloques)
	Si bloques[i].espacioOcupado > resultado
		resultado = bloques[i]
	fin si
	i = i + 1
fin mientras
devolver resultado
```

O(n)

Siendo n la cantidad de bloques

Algoritmo: maxValor

Entrada: bloques: Vector <objeto>

Salida: resultado: entero

```
Entero i = 0;
Entero resultado = 0;
Mientras i < longitud(bloques)
	Si bloques[i].valor > resultado
		resultado = bloques[i]
	fin si
	i = i + 1
fin mientras
devolver resultado
```

O(n)

Siendo n la cantidad de bloques

Análisis de Complejidad Temporal

Complejidad Temporal Teórica

La complejidad temporal teórica respecto del pseudocodigo desarrollado es la siguiente:

Siendo de caso substracción

a= p; (se llama p veces a la función recursiva, siendo p la cantidad de bloques obtenida mediante la cantidad de transacciones sobre la cantidad máxima de transacciones por bloque)

b= 1; (caso de substracción)

k= 1; (grado del polinomio, sentencias ejecutadas fuera de la llamada recursiva)

además sabiendo que es un caso de substraccion, podemos decir que:

$$T(n) = \begin{cases} c & \text{si } 0 \leq n < b \\ aT(n-b) + p(n) & \text{si } n \geq b \end{cases}$$

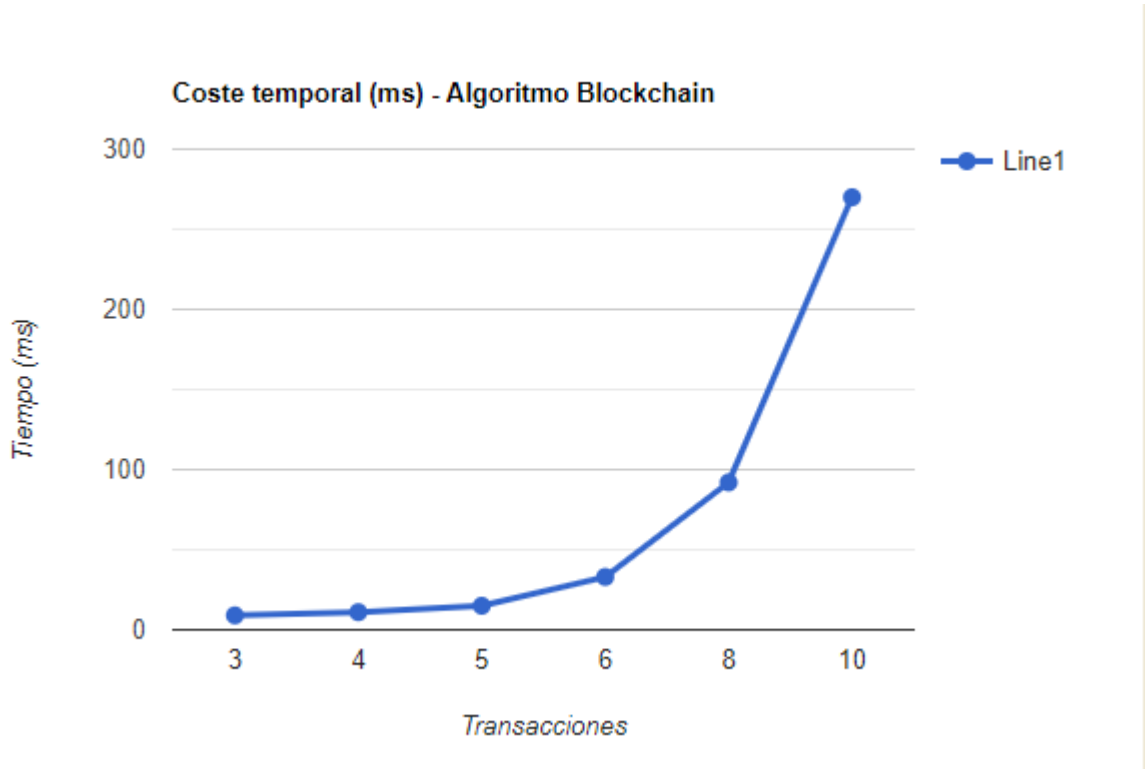
$$T(n) \in \begin{cases} \Theta(n^k) & \text{si } a < 1 \\ \Theta(n^{k+1}) & \text{si } a = 1 \\ \Theta(a^{n \; div \; b}) & \text{si } a > 1 \end{cases}$$

a = p; b= 1; k = 1; => a > 1 => O(a^(n / b)) => O(p^n)

T(n) = O(p^n)

Complejidad Temporal Práctica

Cuando calculamos el coste temporal en la practica, probando con datos de múltiples ejemplos y llegamos a graficarlo de la siguiente forma:



La misma tiene una tendencia similar a una función exponencial que cumple con similitud a la función del coste temporal teórico que es una función exponencial (O(p^n)).

**Cabe aclarar que los datos mostrados en la tabla son promedios de tiempo obtenidos de ejecutar múltiples veces el algoritmo con los mismos parámetros.*

Conclusiones

En conclusión, el objetivo de este trabajo es implementar un algoritmo que construya una blockchain válida, considerando diversas combinaciones posibles de bloques. Cada bloque debe cumplir una serie de reglas, que incluyen validaciones relacionadas con las transacciones como puede ser las dependencias, firmas múltiples, restricciones de tamaño o valor. La estrategia utilizada para el desarrollo del algoritmo fue el uso de backtracking, donde el problema se resuelve de manera recursiva evaluando las transacciones y asignándolas a los bloques posibles (podando las ramas que no cumplan con las reglas previamente mencionadas). La misma es la elegida en comparación a otras estrategias debido a la naturaleza de esta que permite resolver problemas combinatorios (explorando todas las soluciones posibles) de manera eficiente al aplicar poda.
Finalmente luego del desarrollo practico del algoritmo pudimos comparar los costos temporales teóricos y prácticos y llegamos a la conclusión de que la misma tiene un coste temporal similar al de una función exponencial, esto se sabía de la obtención de la formula de coste temporal teórico pero se confirmo a la hora de probar el algoritmo (coste temporal practico). Esto también nos lleva a pensar que si este algoritmo se utiliza para grandes problemas el coste seria poco eficiente en comparación a otros algoritmos.