FACULTAD DE INGENIERÍA Y CIENCIAS EXACTAS

DEPARTAMENTO DE TECNOLOGÍA INFORMÁTICA

TPO - BACKTRACKING

Profesor/es:

Cuadrado, Nombre

Foligno, Nombre

Grupo:

Lo Faro, Bruno– LU: 1157409

Grimoldi, Facundo Javier - LU:1143758

Saud, Juan Manuel– LU: 1110992

Buenos Aires, 8 de Noviembre de 2024.-

Tabla de Contenidos

Introducción

A continuación, expondremos la estrategia, pseudocódigo y análisis de complejidad temporal del problema para construir un blockchain válido.Una blockchain es una estructura de datos distribuida que garantiza la integridad de las transacciones mediante reglas criptográficas y consensos entre los nodos. El objetivo es la construcción de una blockchain válida, considerando todas las combinaciones posibles de bloques. Combinando los valores dentro de la lista de transacciones y validando por cada bloque que, se encuentren con una sumatoria de satoshis menor a 100, un máximo de 3 transacciones, la suma de los valores sea divisible por 10 y que el tamaño máximo de cada bloque sea menor de 1MB, se obtendra un una cadena de bloques compleja.

Descripción del Problema

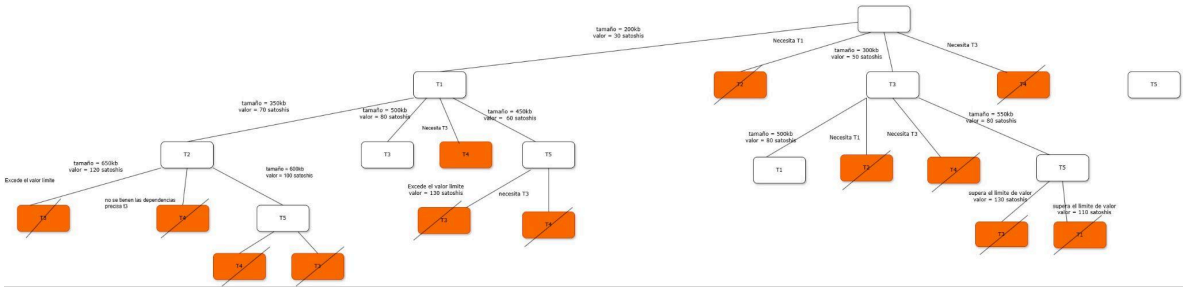
Estrategia de Resolución

Se diseñó un algoritmo que explora de manera exhaustiva todas las combinaciones posibles de transacciones, generando las cadenas de bloques complejas, respetando las validaciones de valor máximo por bloque, máximo de transacciones por bloque, prueba de trabajo y tamaño máximo de bloque. Los parámetros de entrada serán una lista que contendrá el listado de Transacciones. Luego, un índice que comienza en 0, una solución parcial de tipo lista de bloque, el bloque actual de tipo lista de transacciones, una lista de bloques actuales y los datos necesarios para validaciones un tamaño máximo de bloque, un valor máximo de bloque y el máximo de transacciones, los 3 de tipo entero

El algoritmo recorre el listado de transacciones de manera recursiva, seleccionando las transacciones uno a uno de la lista. Se exploran todas las combinaciones posibles (avance de etapa – llamada a la función backtracking()) respetando el orden en el que se encuentran los transacciones en la lista (Transaccion 1, luego Transaccion 2 y así). Si el bloque formado cumple las validaciones(función esSolucion()), se guarda como una solución válida al listado de solución parcial.

El avance de etapa estará dado en el avance de la lista de transacciones a usar (índice+1) y luego estará la exploración de alternativas por la iteración del bucle for para pasar por todos los valores de la lista transacciones actual (ltransacciones[i])

Por último, se añade una poda, validando si la transaccion evaluada tiene dependencias que no pertenecen al bloque antes de llamar a la función bactracking. Si la transaccion no tiene dependencias o la ya se encuentra esa dependencia dentro del bloque, se llama a la función backtracking permitiendo continuar con la combinacionActual como posible solución; si no, se continua con el siguiente elemento en el bucle for.



Pseudocódigo del Algoritmo de Resolución del Problema

AlgoritmoBase(transacciones: Lista<Transaccion>, maxTamanioBloque: Entero, maxValorBloque: Entero, maxTransacciones: Entero)

soluciones = ListaVacia<ListaVacia<Bloque>>

bloquesActuales = ListaVacia<Bloque>

bloqueActual = ListaVacia<Bloque>

indice: Entero = 0

AlgoritmoConstruirBlockchain(transacciones, maxTamanioBloque, maxValorBloque, maxTransacciones, indice, bloqueActual, soluciones, bloquesActuales)

RETORNAR soluciones

FIN AlgoritmoBase

AlgoritmoConstruirBlockchain(transacciones: Lista<Transaccion>, maxTamanioBloque: Entero, maxValorBloque: Entero, maxTransacciones: Entero, indice: Entero, bloqueActual: Lista<Transaccion>, soluciones: Lista<Lista<Bloque>>, bloquesActuales: Lista<Bloque>)

```
SI (esBloqueValido(bloqueActual, maxTamanioBloque, maxValorBloque, maxTransacciones))
    AGREGAR COPIA DE bloqueActual A bloquesActuales
                                    //si el conjunto de bloques es valido, lo agrego al conjunto soluciones

    AGREGAR COPIA DE bloquesActuales A soluciones


    PARA i DESDE indice HASTA LONGITUD(transacciones) - 1

        transaccion = transacciones[i]

        SI sePuedeAgregar(transaccion, bloqueActual)

            AGREGAR transaccion A bloqueActual        //agrego la transaccion al bloque actual
            AlgoritmoConstruirBlockchain(transacciones, maxTamanioBloque, maxValorBloque,
maxTransacciones, i + 1, ListaVacia, soluciones, bloquesActuales)
            ELIMINAR transaccion DE bloqueActual      //deshago el paso de agregar la transaccion al
bloque actual \(backtrack)

        FIN SI

    FIN PARA
                                    //deshago el paso de agregar el bloque al conjunto de bloques
(backtrack)
```

ELIMINAR ÚLTIMO ELEMENTO DE bloquesActuales

FIN SI

FIN AlgoritmoConstruirBlockchain

AlgoritmoEsBloqueValido(bloque: Lista<Transaccion>, maxtamanio: Entero, maxValor: Entero, maxTransacciones: Entero): BOOLEANO

//analizo el bloque entero

PARA cada transaccion EN bloque

bloque.tamanioTotal = bloque.tamanioTotal + transaccion.tamanio

bloque.valorTotal = bloque.valorTotal + transaccion.valor

bloque.numTransacciones = bloque.numTransacciones+1

FIN PARA

//si el bloque es valido agrego una copia de este, al conjunto de bloques

SI bloque.tamanioTotal ≤ maxtamanio Y bloque.valorTotal ≤ maxValor Y bloque.numTransacciones ≤ maxTransacciones Y (bloque.valorTotal MOD 10 = 0)

RETORNAR VERDADERO

SINO

RETORNAR FALSO

FIN SI

FIN AlgoritmoEsBloqueValido

AlgoritmoSePuedeAgregar(transaccion: Transaccion, bloque: Lista<Transaccion>): BOOLEANO

PARA cada dependencia EN transaccion.dependencias

SI dependencia NO ESTÁ en bloques

RETORNAR FALSO

```
    FIN SI
  FIN PARA

  SI transaccion.firmasRequeridas NO se encuentran en bloque
    RETORNAR FALSO
  FIN SI

  RETORNAR VERDADERO
FIN AlgoritmoSePuedeAgregar
```

Análisis de Complejidad Temporal

Complejidad Temporal Teórica

La complejidad temporal teórica de este algoritmo de backtracking depende del tamaño de las listas de transacciones. Suponiendo que la lista de Transacciones tienen longitudes n1, n2…nk y los límites del rango son la cantidad de Transacciones, se debe considerar el largo de la lista de Trasnacciones. Diremos que n es el tamaño de las lista de Transacciones.

El tipo de recurrencia es de sustracción. La función backtracking AlgoritmoConstruirBlockchain() se llama n veces siendo n la cantidad de elementos que tenga la lista (a=n), y eso se repite por la cantidad de transacciones que tenga (PARA i DESDE indice HASTA

LONGITUD(transacciones) - 1). La cantidad de unidades en que se sustrae es 1 AlgoritmoConstruirBlockchain(transacciones, maxTamanioBloque, maxValorBloque, maxTransacciones, **i + 1**, ListaVacia, soluciones, bloquesActuales))(b=1) y el grado de polinomio es lineal (PARA cada dependencia EN transaccion.dependencias)(k=1) pues una de las validaciones recorre una lista. A=n; b=1; k=1 quedaría como n elevado a la n, en el peor de los casos O(n**n)

Complejidad Temporal Práctica

Conclusiones

*Con el análisis realizado sobre el código ejecutado se corrobora la función de la poda en los algoritmos backtracking siendo este efectiva para reducir la complejidad temporal. Teniendo en cuenta que n es el tamaño de la lista, la complejidad del algoritmo en el peor de los casos puede ser n elevado a la n O(n**n) siendo un algoritmo sin poda o bien, con poda pero que todos los elementos sean válidos, ya que la poda no aplicaría efecto.*