Comparación de algoritmos dinámicos y voraces en la solución del problema de la devolución de cambio

Resumen: El problema de devolver el cambio óptimo (con el menor número de monedas) consiste en: Dado un sistema monetario de un país formado por monedas de valores v_1, v_2,..., v_n, el problema del cambio consiste en descomponer cualquier cantidad dada "M" en monedas de ese país utilizando el menor número posible de monedas, de tal manera que se optimice el proceso.

Normalmente este proceso es simple y se puede resolver con un algoritmo voraz, pero no todos los sistemas monetarios son iguales o algunas veces no se cuenta con monedas de cierta denominación, lo cual dificulta aún más el proceso.

Un ejemplo claro es cuando se tienen monedas de 1, 4 y 6 unidades y se requiere devolver 8 unidades; el algoritmo voraz propondría devolver una moneda de 6 y dos de 1, lo cual implicaría darle tres monedas, pero fácilmente podemos darnos cuenta de que dos monedas de 4 unidades es una mejor solución.

En este trabajo se presentan dos soluciones una utilizando un enfoque de algoritmos voraces y la otra con un enfoque de algoritmo dinámico y compararemos sus resultados al devolver cantidades desde 1 hasta 100, con monedas ficticias de 60, 40, 10, 6, 4, 1 unidades.

Palabras clave: Algoritmos dinámicos, Algoritmos voraces, Análisis de algoritmos, Comparación, Problema del cambio

Colaboración
Isaac Alberto Aldave Rojas, Instituto Tecnológico Superior de Ciudad Serdán

Abstract: The problem of returning the optimal change (with the least number of coins) consists of: Given a monetary system of a country consists of currency values v_1, v_2, ..., vn, the problem of change is to break any given amount "M" in the currencies of the country using the fewest number of coins, so that the process is optimized.
Normally this process is simple and can be solved with a greedy algorithm, but not all monetary systems are equal or sometimes do not have coins of a certain denomination, which makes the process even more.

A clear example is when you have coins of 1, 4 and 6 units and 8 units are required to return; the greedy algorithm would propose returning a one coin of 6 and two coins of 1, implying give three coins, but we can easily realize that two coins of 4 units is a better solution.
In this work two solutions have a voracious approach using algorithms and the other with a focus on dynamic algorithm and compare its results to return amounts from 1-100, with fake coins 60, 40, 10, 6, 4, 1 units.

Keywords: Dynamic algorithms, Greedy algorithms, Algorithm analysis, Comparison, Problem of change.

INTRODUCCIÓN

Los algoritmos de planificación son de vital importancia en muchas áreas prácticas del conocimiento, ya que nos permiten prever la administración de un conjunto de recursos que por lo general son limitados o en su defecto su uso se encuentra restringido a algún tipo de norma que no se puede romper.

En las ciencias computacionales el estudio de este tipo de problemas ha dado lugar a diferentes interpretaciones algorítmicas para su solución tratando de optimizar los recursos computacionales primarios como son el tiempo de respuesta y la utilización de la memoria principal.

A continuación se realiza una breve descripción de las técnicas estudiadas haciendo hincapié en que no son las únicas formas de abordad este tipo de problemas.

1.1 Algoritmos Voraces

En el área de la Informática este tipo de Algoritmos se consideran una técnica de diseño general a pesar del hecho de que es aplicable únicamente a problemas de optimización. El enfoque voraz sugiere la construcción de una solución a través de una secuencia de pasos, donde en cada ampliación se obtiene una solución parcialmente construida obtenida hasta el momento, esto continua hasta que se alcanza una solución completa al problema.

En cada paso, y este es el punto central de esta técnica, la opción elegida debe ser:

- Factible, es decir, tiene que satisfacer las limitaciones del problema
- Localmente óptima, es decir, tiene que ser la mejor opción local entre todas las opciones viables disponibles en ese paso
- Irrevocable, es decir, una vez hecho, se no se puede cambiar en etapas posteriores del algoritmo

Estos requisitos explican el nombre de la técnica: en cada paso, sugiere un robo "codiciosos" de la mejor alternativa disponible en la esperanza de que una secuencia de opciones localmente óptimas producirá una solución óptima para todo el problema.

Como regla general, los algoritmos voraces son intuitivamente atractivos y simples. Dado un problema de optimización, por lo general es fácil de encontrar la manera de proceder de una manera voraz, posiblemente después de considerar algunas pequeñas instancias del problema. Lo que es generalmente es más difícil, es demostrar que un algoritmo voraz produce una solución óptima (cuando lo hace). Una de las formas más comunes de hacer esto es el uso de la inducción matemática, que muestra que una solución parcialmente construida obtenida por el algoritmo voraz en cada iteración se puede extender a una solución óptima al problema. [1]

Dado un conjunto finito de entradas C, un algoritmo voraz devuelve un conjunto S (seleccionados) tal que $S \subseteq C$ y que además cumple con las restricciones del problema inicial. A cada conjunto S que satisfaga las restricciones se le suele denominar prometedor, y si este además logra que la función objetivo se minimice o maximice (según corresponda). Diremos que S es una solución óptima.

Elementos de los que consta la técnica:

El conjunto C de candidatos, entradas del problema.

- Función solución. Comprueba, en cada paso, si el subconjunto actual de candidatos elegidos forma una solución (no importa si es ´ óptima o no lo es).
- Función de selección. Informa de todas las soluciones cuál es el elemento más prometedor para completar la solución. Este no puede haber sido escogido con anterioridad. Cada elemento es considerado una sola vez. Luego, puede ser rechazado o aceptado y pertenecerá a C \ S.
- Función de factibilidad. Informa si a partir de un conjunto se puede llegar a una solución. Lo aplicaremos al conjunto de seleccionados unido con el elemento más prometedor.
- Función objetivo. Es aquella que queremos maximizar o minimizar, el núcleo del problema. [3].

1.2. Programación Dinámica

La programación dinámica es una técnica de diseño de algoritmos con una historia bastante interesante. Fue inventada por el matemático Richard Bellman, en la década de 1950 como un método general para la optimización de los procesos de toma de varias etapas. Por lo tanto, la palabra "programación" en el nombre de esta técnica es sinónimo de "planificación" y no se refiere a la programación de computadoras.

La programación dinámica es una técnica para resolver problemas utilizando subproblemas, la programación dinámica sugiere la solución de cada uno de los subproblemas más pequeños sólo una vez y registrando los resultados en una tabla de la que a continuación se puede obtener la solución al problema original.

Si uno usa la versión bottom-up clásica de la programación dinámica o su variación up-bottom, el paso crucial en el diseño de un algoritmo sigue siendo el mismo: derivar una recurrencia relacionada al problema de las soluciones a sus subproblemas más pequeños.

A pesar de su aplicabilidad cuando se aplica a un problema en particular necesita ser comprobada, por supuesto, por lo general dicha comprobación no es la dificultad principal en el desarrollo de una programación basada en Algoritmos dinámicos. [2].

En general, se pueden resolver problemas con subestructuras óptimas siguiendo estos tres pasos:

- Dividir el problema en subproblemas más pequeños.
- Resolver estos problemas de manera óptima usando este proceso de tres pasos recursivamente.
- Usar estas soluciones óptimas para construir una solución óptima al problema original. [3]

Los subproblemas se resuelven a su vez dividiéndolos en subproblemas más pequeños hasta que se alcance el caso fácil, donde la solución al problema es trivial.

Decir que un problema tiene subproblemas superpuestos es decir que se usa un mismo subproblema para resolver diferentes problemas mayores. En una mala implementación puede acabar desperdiciando tiempo recalculando las soluciones óptimas a problemas que ya han sido resueltos anteriormente.

Esto se puede evitar guardando las soluciones que ya hemos calculado. Entonces, si necesitamos resolver el mismo problema más tarde, podemos obtener la solución de la lista de soluciones calculadas y reutilizarla.

Este acercamiento al problema se llama memoizacion (no confundir con memorización; en inglés es llamado memoization,). Si estamos seguros de que no volveremos a necesitar una solución en concreto, la podemos descartar para ahorrar espacio. En algunos casos, se pueden calcular las soluciones a problemas que de antemano sabemos que vamos a necesitar.

MATERIAL Y MÉTODOS

El Problema de la devolución del cambio.

Aunque este problema puede parecernos a nosotros trivial, el explicarle a un equipo de cómputo la forma en que debe devolver el cambio de una transacción (por ejemplo para una máquina expendedora) y que sea además con el menor número de monedas posible, con el fin de optimizar sus recursos, que por lo general son limitados, es importante, no solo para este problema en particular, sino también para aquellos que se basen en los mismo principios. Este problema se ocupa a menudo en los ámbitos académicos para mostrar como la elección de la técnica algorítmica a utilizar tiene influencia sobre el resultado final de una aplicación informática.

El enunciado general de este problema es el siguiente:

En un país X se tienen monedas con las siguientes denominaciones M1, M2,…, Mn dichas monedas pueden representar cualquier cantidad real. Cuando se realiza una transacción monetaria con una cantidad superior a lo pactado y se tiene que devolver el sobrante (cambio), se espera que dicho vuelto tenga la menor cantidad posible de monedas con el fin de que sean fácilmente contadas y transportadas. Sin importar la cantidad a devolver. Exprese una solución a dicho problema asumiendo en este primer ejemplo (caso de estudio) que la cantidad de monedas de cada denominación es ilimitada.

Una aplicación práctica a este problema son las máquinas expendedoras, las cuales deben devolver cambio en la mayoría de las transacciones que se realizan optimizando los recursos que posee ya que a diferencia del problema académico clásico, no existen en esta monedas ilimitadas de cada valor. Otra forma de utilizar estas implementaciones es la realización de aplicaciones que permitan evaluar que tan eficientemente un niño de educación básica comprende el uso de su sistema monetario cuando debe devolver cambio tras una transacción comercial, siguiendo la regla de menor cantidad de monedas, fortaleciendo tanto la realización de operaciones como el análisis del problema para dar su solución, y el reforzamiento positivo que debe dar la aplicación tanto cuando se acierta como cuando se comete algún error.

A continuación se presenta la descripción de la implementación voraz y dinámica a este problema.

Solución voraz

Es fácil implementar un algoritmo voraz para resolver este problema, que es el que sigue el proceso que usualmente utilizamos en nuestra vida diaria. Sin embargo, tal algoritmo va a depender del sistema monetario utilizado y por ello vamos a plantearnos dos situaciones para las cuales deseamos conocer si el algoritmo ávido encuentra siempre la solución óptima:

Suponiendo que cada moneda del sistema monetario del país vale al menos el doble que la moneda de valor inferior, que existe una moneda de valor unitario, y que disponemos de un número ilimitado de monedas de cada valor.

Suponiendo que el sistema monetario está compuesto por monedas de valores 1, p, p2, p3,..., pn, donde $p > 1$ y $n > 0$, y que también disponemos de un ´ número ilimitado de monedas de cada valor.

La mejor solución siguiendo la metodología de análisis y planteamiento del problema de forma voraz es definiendo los siguientes puntos:

- Conjunto de candidatos.- Conjunto de monedas
- Función de solución.- el valor total del conjunto de monedas elegidas es exactamente igual a la cantidad a pagar.
- Función objetivo.- minimizar el número de monedas utilizadas en la solución.

- Función de selección.- elegir la moneda de mayor valor posible tal que satisfaga las restricciones de factibilidad.

El algoritmo resultante de este proceso es de orden n con respecto al número de monedas.

Sin embargo este algoritmo depende del sistema monetario que se ocupa en cada país, pudiendo no dar el mejor resultado bajo ciertas características de dicho sistema. Por ejemplo {6, 4, 1} y {11, 5, 1}. En estos sistemas monetarios este algoritmo produciría resultados no óptimos.

Solución Dinámica

El algoritmo dinámico se basa en llenar una tabla C conformada por tantas filas como denominaciones de monedas hay, y las columnas serán las cantidades entre 1 y el valor a devolver, de esta manera se calcularan todas las formas posibles de devolver entre 1 y la cantidad requerida haciendo uso de las diferentes denominaciones que se tienen.

Por lo tanto devolver una cantidad estará relacionado con las denominaciones utilizadas y con la soluciones de devolver cantidades inferiores.

Donde el numero almacenado en C[i][j] será el número mínimo de monedas necesarias para devolver j unidades, haciendo uso de las primeras i denominaciones y T[i], es la i-ésima denominación, y se calcula de la forma que se observa en la figura 1.
La complejidad del algoritmo anterior es la complejidad de construir la tabla de tamaño i X j donde i es el número de monedas en el sistema y j el cambio a repartir.

$$C[i][j] = \begin{cases} \infty & \text{si } (i = 1) \text{ Y } (1 <= j <= T[i]) \\ 0 & \text{si } (j = 0) \\ 1 + C[i][j - t[i]] & \text{si } (i = 1) \text{ Y } (j >= T[i]) \\ C[i-1][j] & \text{si } (i > 1) \text{ Y } (j < T[i]) \\ \min(C[i-1][j],\ 1 + C[i][j - t[i]]) & \text{de lo contrario} \end{cases}$$

Figura 1. Cálculo de denominaciones

De esta tabla para devolver el cambio óptimo diciendo cuales son y en qué cantidad son las denominaciones de las monedas a devolver se debe hacer uso de un arreglo auxiliar del mismo tamaño que el arreglo de monedas y un recorrido de la matriz de C de la siguiente forma: i=número de monedas j=cambio a devolver Mientras j diferente de 0 haz: Si c[i,j]≥C[i-1,j] Mc[i]= Mc[i]+1; i=número de monedas aux=aux+Mc[i] j=cambio a devolver-aux en otro caso i=i-1 //cambio de moneda a la siguiente denominación mayor.

La metodología que se utiliza para verificar cuál de las dos implementaciones ofrece mejores soluciones fue la siguiente:

RESULTADOS Y CONCLUSIONES

Al realizar la ejecución de estos algoritmos por utilizando como herramienta Matlab ®. Con un sistema monetario ficticio {60, 40, 10, 6, 4, 1} se obtuvieron los siguientes resultados:

Ambos algoritmos se comportan de una forma aceptable, sin embargo el algoritmo voraz con este sistema monetario produjo resultados no óptimos, mientras que el algoritmo dinámico, siempre devolvió mejores soluciones.

El algoritmo dinámico es más barato en tiempo de ejecución que el algoritmo voraz para el cálculo de múltiples valores a devolver.
Este algoritmo en su implementación de forma general utiliza dos tablas una para el cálculo y una copia de esta para la búsqueda de las soluciones, lo que implica un gran requerimiento de memoria RAM, sin embargo con una pequeña modificación y calculando por métodos estadísticos cuál es el mayor cambio que puede devolver la máquina expendedora, es posible utilizar la misma tabla para ambas tareas. Devolviendo el tipo de las monedas a devolver y la cantidad de cada una de ellas.

La función de recorrido la cual tiene un orden de ejecución menor a n siendo n la cantidad a devolver. Este hecho compensa la cantidad de memoria que ocupa la tabla en el sistema.

Para este trabajo se realizaron las pruebas para la devolución con monedas de las siguientes cantidades en el rango de 1 hasta la cantidad 100 con un sistema monetario {60, 40, 10, 6, 4, 1}, los algoritmos se comportaron como se expresa en la gráfica 2:

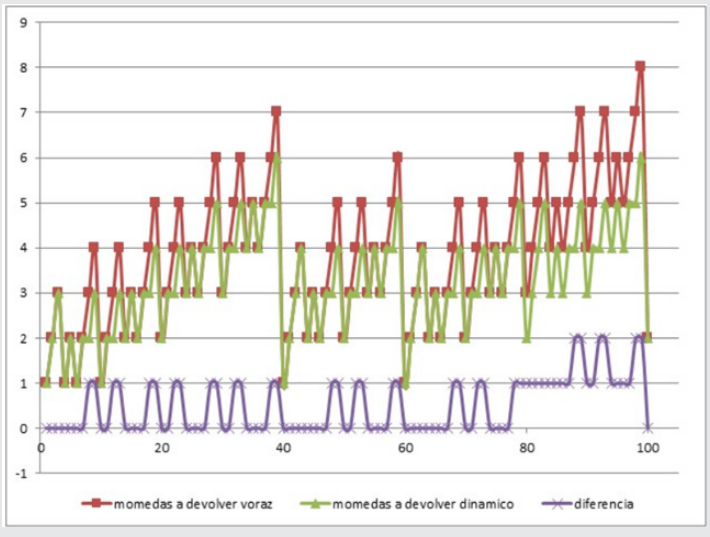

Figura 2 Número de monedas que entregan ambos algoritmos y la diferencia entre estas cantidades.

CONCLUSIONES

- En el caso del problema de la devolución del cambio, el algoritmo dinámico se comporta de forma óptima ya que no varía su eficiencia en relación directa con el sistema monetario, tal y como la hace el algoritmo voraz que para el caso del sistema monetario propuesto, fallo en dar el cambio de forma óptima en un 46%. Al cual al analizar su comportamiento se incrementara conforme aumente n.

- Con respecto a la utilización de la memoria el algoritmo dinámico consume un arreglo de ´ mXn posiciones más dos vectores de tamaño m como elementos principales de su funcionamiento, lo cual se compensa con su asertividad y con el hecho de que la misma tabla sirve para calcular cualquier cantidad en el rango calculado, con lo cual solo se realiza el procedimiento de llenado una sola vez a diferencia del voraz que rehace todos los cálculos en cada ocasión.

- Con respecto a otras implementaciones para la solución de este tipo de problemas tal como pueden ser los algoritmos genéticos por ejemplo, aunque los algoritmos genéticos reducen en gran medida la cantidad de cálculos a realizar para la solución de un problema de optimización, estos no garantizan encontrar la solución óptima y en muchas ocasiones ni siquiera encuentran una solución factible, al depender en su desarrollo de un alto componente probabilístico, que no está presente en las implementaciones de los algoritmos seleccionados.

REFERENCIAS

[1] Levitin, Anany. Introduction to the design & analysis of algorithms, 3rd ed. (2012) chapter 9.

[2] Levitin, Anany. Introduction to the design & analysis of algorithms, 3rd ed. (2012) chapter 8.

[3] Cormen, Thomas H. Introduction to Algorithms, Second Edition (2001).