Solución al problema de Criptoaritmética

Alejandra Campo Archbold, Juan Pablo Sierra Useche

September 22, 2021

Contents

1 Introducción

La criptoaritmética es un problema que se deriva de la criptología; aunque a diferencia de la criptología, que busca implementar sistemas seguros, la criptoaritmética tiene un caracter más de entretenimiento.

Este ejercicio tiene tres componentes principales:

- Dos sumandos
- Un resultado

Como cualquer suma, esta operación binaria toma dos elementos y al sumar sus valores obtiene uno representativo como la suma de los dos. A diferencia de una suma convencional, en la criptoaritmetica se inicia con las tres componentes en forma de letras tal que cada caracter $(A_i, B_j, C_k \in \{A, B, C, \ldots, X, Y, Z\}$ para todo $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$ y $k \in \{1, \ldots, o\}$de la forma:

$$\begin{array}{cccccc} & A_m & A_{m-1} & \ldots & A_2 & A_1 \\ + & B_n & B_{n-1} & \ldots & B_2 & B_1 \\ \hline & C_o & C_{o-1} & \ldots & C_2 & C_1 \end{array}$$

Las tres componentes no deben ser de igual tamaño, es decir (teniendo en cuenta la notación de la ecuación anterior) m, n y o no deben ser iguales; pero deben cumplir tres reglas:

1. $A_i + B_i = C_i$
2. Si $A_i + B_i \neq C_i$ existen 3 posibilidades:
   - Si $A_i + B_i >= 10$ entonces $A_i + B_i = 10 + C_i$
   - Si $A_i + B_i < 10$ entonces $A_{i-1} + B_{i-1} = 10 + C_{i-1}$
   - Si $A_i + B_i >= 10$ y $A_{i-1} + B_{i-1} >= 10$ entonces $A_i + B_i = 10 + C_i$ y $A_{i-1} + B_{i-1} = 10 + C_{i-1}$.
3. Dos letras distintas no puedes ser asignadas a un mismo número.

Ahora, teniendo estas reglas como base, la finalidad del problema es encontrar una combinación donde todas las reglas se cumplan. Un ejemplo posible sería el siguiente:

$$\begin{array}{cccc} & A_2 & A_1 & \\ + & B_2 & B_1 & \\ \hline & C_2 & C_1 & \end{array} \rightarrow \begin{array}{cccc} & A & B & \\ + & C & D & \\ \hline & E & F & \end{array} \rightarrow \begin{array}{cccc} & 3 & 2 & \\ + & 5 & 7 & \\ \hline & 8 & 9 & \end{array}$$

2 Métodos

Con la finalidad de encontrar una forma para calcular las soluciones a este problema, a conticuación, se van a explicar las dinamicas del ejercicio de foma más rigurosa:

2.1 Estado Inicial

El estado inicial del problema consite del las tres componentes (ya descritas en la introducción) como conjunto de caracteres entre la A y la Z.

2.2 Posibles Acciones

Las posibles acciones se definen como una función del conjunto de letras (aquellas que conforman a las tres componentes) a las que no se les ha asignado un valor numérico apuntando al conjunto de números naturales en el intervalo $[0-9]$ que aún no han sido asignados a una letra.

2.3 Función de Transición

Esta función retorna un estado del problema, donde se le a agregado la nueva asignación letra→número y a los conjuntos de letras y números disponibles se les extrae los valores respectivos a la relación recien formada.

2.4 Prueba de objetivo

Esta función determina si el estado actual cumple la relación entre los dos sumandos y el resultado.

2.5 Función de costo

La función de costo está orientada a la implementación de una lista prioritaria, donde se calcula el costo dependiendo de la frecuencia de cada una de las letras que faltan por reemplazar, entre más frecuente menor es el costo (es decir más prioritario segun lista prioritaria).

2.6 Algoritmos de busqueda

Vale la pena recalcar que el diseño del problema se implementó en formato nodal (al igual que en clase), Por lo que el problema funciona con los algoritmos ya programados desde la clase. Por lo tanto decidimos hacer pruebas utilizando los siguientes algoritmos de busqueda:

- **Depth First Search (DFS)**: El DFS es un algoritmo que funciona en arboles, y funciona recorriendo de un extremo al otro del arbol, buscando siempre analizar el nodo más profundo.

- **Breadth First Search (BFS)**: El algoritmo tambien hace un barrido de un lado al otro de un arbol, pero a diferencia del DFS, el BFS le da como privilegio a los niveles del arbol, por lo que revisa todos los nodos de cada nivel antes de bajar al siguiente.
- **Depth Limited Search (DLS)**: Este algoritmo funciona de la misma forma que el **Depth First Search**, pero estableciendo un límite de profundidad.
- **Iterative Deepening Search (IDS)**: Este algoritmo es la combinación de los dos anteriores, buscando repetidamente hasta cierta profundidad por todo el nivel.
- **Best First Search (BestFS)**: Con este algoritmo se busca el nodo según la lista prioritaria lo determine.

3 Resultados

- **Prueba 1**: Resolver problemas con sumandos y resultados de igual tamaño (igual a dos) y seis letras diferentes.
- **Prueba 2**: Resolver `SEND + MORE = MONEY`.

Prueba	Algoritmo	Resultados
1	DFS	Promedio de `7 ms`.
1	BFS	Nunca nos entregó resultados.
1	DLS	Promedio de `9.80 ms`.
1	IDS	Promedio de `23.7 s`.
1	BestFS	Promedio de `1 min`.
2	DFS	Promedio de `886 ms`.
2	BFS	Nunca nos entregó resultados.
2	DLS	Promedio de `298 ms`.
2	IDS	Promedio de `11 mins`.
2	BestFS	Nunca nos entregó resultados.

4 Discución

El algoritmo de **BFS** tuvo problemas de tiempo, y hay dos hipótesis:

1. Se cree que el problema está relacionado con la definición de prioridad, implicando que no encontramós la mejor de las formas para medir el problema.

2. Se cree que el problema se vincula con el mismo rendimiento de la función de costo, agregando demasiada complejidad al problema.

Dejando abierta la mejor forma de solucionar el problema, pero entregando buenas soluciones con algoritmos más simples.

5 Conclusiones

Con problemas más simples, funciona mejor el DFS ya que el algoritmo es bastante simple y es justo para lo que necesita el problema. Pero cuando la complejidad del estado inicial va aumentando, el DLS resulta con mucho mejores promedios ya que administra mejor la busqueda, asumiendo la cantidad de letras y así dejando de intentar soluciones cuando estas se vuelven muy largas.