# Proyecto 3: Suma de Subconjuntos

Ricardo Castro Jiménez
Carmen Hidalgo Paz
Jorge Guevara Chavarría

*Escuela de Ingeniería en Computación, Tecnológico de Costa Rica, San José, Costa Rica*

riccastro@estudiantec.cr
carmenhidalgopaz@estudiantec.cr
joguevara@estudiantec.cr

Este proyecto consiste en resolver el problema de la Suma de Subconjuntos utilizando backtracking. El objetivo es implementar distintas variantes del problema y presentar los resultados en una interfaz gráfica hecha en C con GTK y Glade. Se permiten subconjuntos de un conjunto A cuyos elementos son números enteros positivos que están ordenados de menor a mayor, y se debe comparar la suma de dichos subconjuntos con una meta W definida por el usuario. A lo largo del proyecto, aprendimos a aplicar recursión de manera eficiente, validar datos de entrada, y presentar visualmente grandes volúmenes de datos.

## Variantes del Problema

El programa resuelve cuatro variantes del problema clásico de Suma de Subconjuntos. Cada variante modifica la condición bajo la cual un subconjunto es considerado solución válida. Todas se implementaron mediante backtracking, y están disponibles para el usuario desde la interfaz gráfica.

### Variante 1: Suma Exacta

Esta es la forma más clásica del problema. Se busca encontrar todos los subconjuntos del conjunto A cuya suma sea exactamente igual a W. Usamos una función recursiva que va probando si incluir o no cada elemento, y cuando la suma llega a W, se guarda como solución. Si la suma se pasa, se deja de explorar esa rama.

### Variante 2: Rango [W - Δ, W + Δ]

En esta variante se permite un margen de error. El usuario define un parámetro Δ, y se aceptan subconjuntos cuya suma esté entre W - Δ y W + Δ. Se adapta la función clásica de backtracking para seguir explorando incluso si la suma ya pasó de W, mientras no supere el límite superior del rango. Esto permite encontrar más soluciones útiles en casos donde una coincidencia exacta es poco probable.

### Variante 3: Mayor o Igual

Aquí se buscan subconjuntos cuya suma sea mayor o igual que W. A diferencia de las variantes anteriores, no se detiene cuando se encuentra una solución: se sigue explorando porque se podrían formar subconjuntos más largos que también cumplen. Esto genera más soluciones, incluyendo algunas que son extensiones de otras ya válidas.

**Variante 4: Mayor o Igual Acotado**

Esta variante es parecida a la anterior, pero con una restricción importante: una vez que un subconjunto ya cumple la condición (suma ≥ W), no se le pueden agregar más elementos. De esta forma, cada solución es mínima y no hay supersets de otras soluciones. El algoritmo se detiene justo cuando se encuentra una suma válida, para no seguir expandiendo.

## Interfaz Gráfica

Se diseñó una interfaz amigable usando GTK y Glade [4]. El usuario puede ingresar el tamaño del conjunto A (entre 3 y 12), los valores de cada elemento (estos en forma ascendente), la meta W, y elegir la variante que desea resolver. Si escoge la variante 2, se le pide también el valor de Δ (que debe ser menor que el valor mínimo de todos los elementos). Al presionar el botón de ejecutar, el programa procesa la información y muestra los subconjuntos válidos dentro de una zona con scroll [5]. Cada subconjunto se representa como una línea con checkboxes que indican qué elementos forman parte de la solución. También se muestra la suma de cada subconjunto. La interfaz informa cuántas soluciones se encontraron y cuántos nodos fueron explorados.

## Resultados y Validaciones

Probamos el programa con distintos tamaños de conjunto y diferentes metas W. Validamos que cada variante generara soluciones correctas y que el número de nodos aumentara con la complejidad del problema. Se hicieron validaciones para evitar valores negativos, campos vacíos o caracteres no numéricos. Además, se aseguró que el programa no crasheara por entradas fuera del rango o combinaciones inválidas.

## Conclusión

Este proyecto nos permitió aplicar recursión de manera eficiente y entender cómo pequeñas variaciones en un problema pueden cambiar completamente el comportamiento del algoritmo. Trabajamos en equipo dividiendo bien las tareas, lo cual nos permitió avanzar rápido y cubrir tanto la parte lógica como la gráfica. La experiencia con GTK y Glade nos ayudó a reforzar el desarrollo de interfaces profesionales y el manejo de eventos desde código en C.

## Fuentes Consultadas


[1] GeeksforGeeks, “Subset Sum Problem”, [En línea]. Disponible en: https://www.geeksforgeeks.org/subset-sum-problem-dp-25/ [Accesado: 10 de mayo del 2025].

[2] GeeksforGeeks, “Backtracking with Constraints”, [En línea]. Disponible en: https://www.geeksforgeeks.org/backtracking-algorithms/ [Accesado: 10 de mayo del 2025].

[3] Enunciado del Proyecto 3, Análisis de Algoritmos, Tecnológico de Costa Rica, 2025.

[4] K. O’Kane, “Linux Gtk Glade Programming Part 1”, YouTube, [video en línea]. Disponible en: https://www.youtube.com/watch?v=g-KDOH_uqPk [Accesado: 10 de mayo del 2025].

[5] K. O’Kane, “Linux Gtk Glade Programming Part 25 Dynamic Scrollable Grid of Buttons”, YouTube, [video en línea]. Disponible en: https://www.youtube.com/watch?v=9SHHjzc9wqA [Accesado: 11 de mayo del 2025].