DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico I

Eligiendo justito

Algoritmos y Estructuras de Datos III
Segundo Cuatrimestre de 201

Integrante	LU	Correo electrónico
Facundo Linlaud	561/16	facundolinlaud@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Resumen

En este documento se analizará el problema de la suma de subconjuntos (*subset sum* en inglés). Dado un conjunto de n elementos V generalizados v_i y un valor objetivo T, decidir si existe un subconjunto de S cuyos elementos sumen exactamente T y, de existir múltiples soluciones, dar el mínimo cardinal.

Es interesante analizar este problema y sus derivados por su importante papel, por ejemplo, en las Ciencias de la Computación. Uno de sus caso de uso es encontrar la mejor distribución de tareas a ejecutar en dos procesadores, minimizando tiempos de *idling* no es mas que una instancia del *subset sum* con $T = \frac{total}{2}$, siendo $total$ la suma de los tiempos totales de todas las tareas a ejecutar.

Este problema nos invita a preguntarnos:

- ¿Todas las instancias de este problema tienen solución?
- ¿Qué técnicas algorítmicas podemos emplear en nuestras soluciones y cuáles son sus complejidades?
- ¿Hay *entry sets* más o menos performantes? ¿Qué forma tienen?

Índice

1. Análisis del problema

Definiremos el conjunto de n elementos I como la estructura que almacena todos los índices de valores asociados $v_i \in \mathbb{N}_0$. Comenzaremos analizando las aristas del problema con una serie de ejemplos:

1.1. Casos descartables

Podemos decidir rápidamente si una instancia tiene solución si esta cumple alguna de las siguientes características:

1.1.1. Elementos mayores al valor objetivo

Si todos los elementos del conjunto son mayores al valor objetivo T, entonces no importa que subconjunto se elija, el valor T será inferior a cualquier elemento del subconjunto exceptuando el caso $T = 0$ y el conjunto vacío.

1.1.2. Suma total impar y valor objetivo equivalente a la mitad del total

Sea $total$ la suma de todos los elementos del conjunto, si se busca obtener un subconjunto que sume $\frac{total}{2}$, es necesario que $total$ sea divisible por dos. De lo contrario, no sería posible encontrar una mitad de $total$ en el conjunto I.

En otras palabras, no existe un subconjunto solución I tal que $\sum_{i \in I} v_i = \frac{T}{2}$ porque $2 \nmid total$ y todos los valores asociados a v_i en I son naturales.

1.2. Casos no descartables

A diferencia de los anteriores, la factibilidad de ciertos casos no puede ser determinada a simple vista. Para ello, es necesario procesar estas instancias del problema mediante diferentes algoritmos que se explayarán más adelante. Ahora, analizaremos algunas posibles formas del problema de la suma de subconjuntos:

Primer ejemplo

Dada una lista indexada desde el cero de valores $V = [10, 15, 5, 10, 5]$ y $T = 25$ podemos encontrar las siguientes cinco soluciones, donde la menor cardinalidad es dos:

- $I = \{0, 1\}$, sumando los valores 10 y 15
- $I = \{1, 3\}$, sumando los valores 15 y 10
- $I = \{0, 2, 3\}$, sumando los valores 10, 2 y 10
- $I = \{0, 3, 4\}$, sumando los valores 10, 10 y 5
- $I = \{1, 2, 4\}$, sumando los valores 15, 5 y 5

Segundo ejemplo

$V = [2, 8, 9, 13, 16]$ y $T = 24$ podemos encontrar sólo una solución de cardinalidad dos:

- $I = \{1, 4\}$, sumando los valores 8 y 16

Tercer ejemplo

$V = [2, 8, 9, 13, 16]$ y $T = 20$ no posee soluciones.

2. Técnicas propuestas

Se nos pidió implementar tres soluciones para el problema utilizando las siguientes técnicas algorítmicas por separado:

- Brute-forcing
- Backtracking
- Programación dinámica

2.1. Brute-forcing

Esta técnica consiste en probar absolutamente todas las combinaciones posibles. Si bien esta técnica es sencilla de implementar y asegura encontrar una solución al problema si es esta existe, su complejidad generalmente es muy cara y existen alternativas más eficientes que veremos a continuación.

2.2. Backtracking

2.3. Programación Dinámica

Esta técnica puede ser resumida como *divide, conquer & memoization*, donde se minimizan las llamadas recursivas al extinguir aquellas que ya han sido calculadas anteriormente. Esto sucede cuando existen recursiones que se invocan más de una vez con los mismos parámetros.

En este documento, se considerarán dos enfoques de la programación dinámica:

- El enfoque top-down
- El enfoque bottom-up

3. Brute-forcing

3.1. Solución

3.2. Implementación

3.3. Complejidad

3.4. Análisis de performance

4. Backtracking

4.1. Solución

4.2. Implementación

4.3. Complejidad

4.4. Análisis de performance

5. Programación dinámica

5.1. Solución

Antes que nada, debemos demostrar que el Principio de Optimalidad de Bellman vale para el problema de la suma de subconjuntos. Es decir, que toda solución óptima de un problema es construida a partir de subsoluciones óptimas de sus respectivos subproblemas. ¿Entonces, se cumple el principio de optimalidad? Verifiquémoslo.

5.1.1. Cumplimiento del Principio de Optimalidad

Asumiendo I un conjunto solución de n índices correspondientes a elementos en V tal que $I = \{1, ..., n\}$ y $\sum_{i \in I} v_i = T$ con T el valor objetivo. Para cada valor posible en V hay dos posibilidades: que sea parte de la solución o no lo sea.

I) $n \notin I \implies I \subseteq \{1, 2, ..., n-1\}$

De esta manera, el valor de T no cambia. Es decir: $\sum_{i \in I} v_i = T$ con I solución óptima por hipótesis. Luego, I remanece solución óptima para el subproblema de valor objetivo T.

II) $n \in I$:

En este caso, se tiene que cumplir $\sum_{i \in I'} v_i = T - v_i$ con $I' = I - \{n\}$ y I' solución óptima.

Si $I' = I - \{n\}$ **no** fuese solución óptima entonces existiría un $I = I''$ mínimo (y óptimo) tal que $|I''| < |I'|$ y $\sum_{i \in I''} v_i = T - v_n$

Pero si I'' fuese solución óptima del subproblema $n-1$ entonces la solución óptima del problema n sería $I = I'' \cup \{n\}$ y recordando la equivalencia $I = I' \cup \{n\}$ que fue obtenida anteriormente, podemos generar un sistema de ecuaciones y luego aplicar álgebra:

$$\begin{cases} I' = I - \{n\} \\ I = I'' \cup \{n\} \end{cases} \iff I' = (I'' \cup \{n\}) - \{n\} \iff I' = I''$$

La equivalencia entre I' e I'' obtenida como conclusión contradice la declaración $|I''| < |I'|$, obteniéndose un absurdo. Luego, I' es subsolución óptima del subproblema.

Finalmente, el problema satisface el principio de optimalidad.

□

5.2. Implementación

5.3. Complejidad

5.4. Análisis de performance