INTERESES BANCARIOS

Enunciado:

Dadas n funciones f1, f2, ..., fn y un entero positivo M, deseamos maximizar la función f1(x1) + f2(x2) + ... + fn(xn) sujeta a la restricción x1 +x2 +...+ xn = M, donde fi(0) = 0 (i=1,..,n), xi son números naturales, y todas las funciones son monótonas crecientes, es decir, x ≥ y implica que fi(x) > fi(y). Supóngase que los valores de cada función se almacenan en un vector.

Este problema tiene una aplicación real muy interesante, en donde fi representa la función de interés que proporciona el banco i, y lo que deseamos es maximizar el interés total al invertir una cantidad determinada de dinero M. Los valores xi van a representar la cantidad a invertir, en miles de pesos, en cada uno de los n bancos.

Para todas las técnicas empleadas en este trabajo, se usará una matriz denotada como *matrizF* de n filas y M+1 columnas, siendo n la cantidad de bancos y M el dinero a invertir (cabe aclarar que si M=6, por ejemplo, significa que M representa 6000 pesos). Las celdas que se encuentran en la fila i representan los valores que el banco i otorga para distintas cantidades de dinero (desde cero hasta una cantidad específica de dinero), es decir que matrizF[i][j] indica el monto total que se obtiene del banco i si se le entrega j pesos. A continuación, se adjunta una matrizF de 3x7 (n = 3 y M = 6)

	0 ($0)	1 ($1000)	2 ($2000)	3 ($3000)	4 ($4000)	5 ($5000)	6 ($6000)
Banco 1	0	1148	2641	3842	4426	5472	6666
Banco 2	0	1484	2583	3054	4251	5640	6945
Banco 3	0	1256	2150	3253	4524	5556	6346

A modo de ejemplo, del banco 2 se obtendrán $4251 si en él se invierten $4000

Técnica de resolución #1: Programación dinámica

La solución a este problema puede ser alcanzada a través de una secuencia de decisiones, una en cada etapa. Denotamos $I_n(M)$ al interés máximo obtenido al invertir M pesos distribuido en n bancos, donde: $I_n(M)$ = f1(x1) + f2(x2) … + fn(xn) con la restricción x1 + x2 … + xn = M. $I_n(M)$ resulta de una serie de decisiones y es óptimo para el problema de invertir en n bancos con M pesos, entonces cualquier subsecuencia de decisiones será optima, entonces para el problema de invertir (M – xn) pesos entre n -1 bancos, el valor óptimo será $I_{n-1}(M-xn)$ = f1(x1) + f2(x2) … + fn(xn).

De esta forma, se utilizará una matriz denominada *matrizI* de n filas y M+1 columnas (nótese que tiene las mismas dimensiones que *matrizF*). donde matriz[i][j] representa el total obtenido al invertir j pesos entre i bancos. El resultado solicitado por el problema en cuestión estará indicado en la celda matrizF [n] [M+1]

Relleno de la tabla:

La primera fila será igual que la primera fila de matrizF. Como solo existe un banco *(el banco 1)*, la única opción posible es invertir todo el dinero j (0 <= j <= M) en ese banco para obtener alguna ganancia.

En la segunda fila, para cada columna j, se toma el máximo entre invertir todo el dinero j en uno de los dos bancos o depositar (j-p) pesos en un banco y p pesos en el otro banco tal que 1 <= p <= j-1. Así se obtiene el óptimo de invertir j pesos entre dos bancos (banco1 y banco2).

Ahora bien, en la fila r (tal que 1 < r <= n), cada celda j será completada tomando el máximo valor entre los valores correspondientes a:

- el valor óptimo de invertir los j pesos en los r-1 bancos anteriores y, por ende, no invertir en el banco r (este valor se ubica en **matrizI [r-1][j]**),
- invertir todo el dinero j en el banco r,
- los valores óptimos de invertir (j-p) pesos en los r-1 bancos anteriores mas el total obtenido de invertir p pesos en el banco r (p=1, 2… j -1)

Este procedimiento es reflejado en la siguiente ecuación de recurrencia:

$$I_n(x) = \begin{cases} f_1(x) & \text{si } n = 1 \\ \underset{0 \le t \le x}{Max}\{I_{n-1}(x-t) + f_n(t)\} & \text{en otro caso.} \end{cases}$$

Finalmente, a partir de esta secuencia de sub-decisiones, donde cada una es óptima, se llega a una solución óptima que aparece en la última posición de la tabla

Técnica de resolución #2: Backtracking/Vuelta atrás

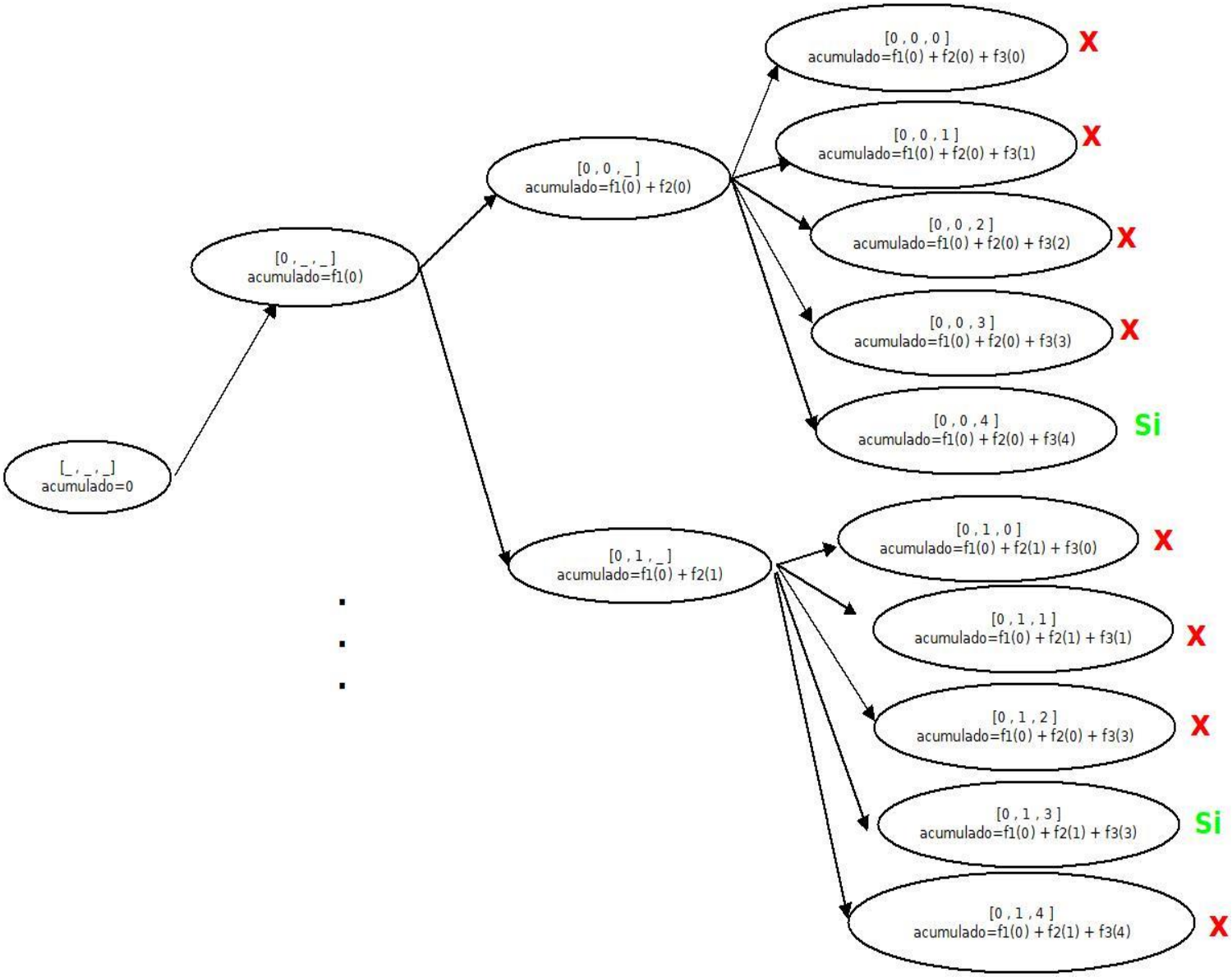
Mediante esta técnica se pueden ir obteniendo las distintas formas de distribuir los m pesos entre los n bancos, dado que se puede ir formando una solución (o mejor dicho una combinación) a partir de etapas, éstas se pueden comparar entre si para poder determinar la solución óptima.

La solución se puede presentar como una lista [x1, x2, … xn] donde

- cada xi debe cumplir que 0<=xi<=M, siendo M el dinero disponible (restricción explícita)
- se debe respetar la restricción x1+x2…xn = M. (restricción explícita)
- el resultado de f1(x1) + f2(x2) + … + fn(xn) debe ser lo máximo posible

A continuación, se presenta una parte del árbol de expansión que simula el procedimiento de este algoritmo.

Cada nodo estará compuesto por el arreglo solución que indica cuánto depositar en cada banco y también de un valor llamado *acumulado* que indica en cada etapa *i* cuanto se ha acumulado distribuyendo el dinero entre el banco 1, banco 2, … banco i



Será necesario un valor óptimo global para compararlo con cada solución encontrada y actualizarlo en caso de ser necesario.

Finalmente, se llegará al interés máximo

Ahora bien, el algoritmo propuesto y adjunto solo indica el interés máximo a obtener

Técnica de resolución #3: Algoritmo voraz

El inconveniente que presenta esta técnica es elegir una función de selección adecuada para llegar a la solución óptima global (el cual ya lo obtuvimos con la programación dinámica), dado que este algoritmo seguirá un rumbo específico sin la posibilidad de retroceder etapas en caso de que la solución no conduzca a un óptimo global.

Se utilizará una matriz denotada como *matrizF2.* A modo de ejemplo: reutilzaremos a *matrizF*

	0 ($0)	1 ($1000)	2 ($2000)	3 ($3000)	4 ($4000)	5 ($5000)	6 ($6000)
Banco 1	0	1148	2641	3842	4426	5472	6666
Banco 2	0	1484	2583	3054	4251	5640	6945
Banco 3	0	1256	2150	3253	4524	5556	6346

Esta última nos indica el dinero total obtenido al invertir j pesos en un banco. MatrizF2 nos indicará la ganancia resultante de invertir j pesos en el banco i:

MatrizF2:

	0 ($0)	1 ($1000)	2 ($2000)	3 ($3000)	4 ($4000)	5 ($5000)	6 ($6000)
Banco 1	0	148	641	842	426	472	666
Banco 2	0	484	583	054	251	640	945
Banco 3	0	256	150	253	524	556	346

En este caso, los candidatos serían las cantidades de dinero a invertir j=(0, 1, 2… M) que se debe destinar en cada banco

La función de selección consistirá en buscar la fila de matrizF2 que cumpla la condición de que la suma de sus celdas sea máxima. Una vez encontrada esa fila, de ésta se encontrará el máximo elemento, el cuál será nuestro “óptimo local” y será acumulado en la solución a mostrar. Una vez hecho esto, a las celdas de la fila en cuestión se les asignará el valor 0 (cero). Este procedimiento continúa hasta las (n-1) filas de matrizF2 sean “anuladas”.

CONCLUSIÓN:

Las técnicas de programación dinámica y backtracking brindan el interés máximo, la diferencia entre éstas es el tiempo de ejecución. Backtracking consiste en un proceso de prueba y error que trabaja por etapas y así construye gradualmente la solución. El inconveniente radica en que en cada etapa las combinaciones a analizar aumentan exponencialmente. De forma tal que este algoritmo analiza M^n combinaciones posibles para poder determinar el óptimo global. Aunque cabe aclarar que esta técnica puede ser mejorada ignorando de alguna forma algunas combinaciones que no llevan a una solución

En cambio, con programación dinámica, aprovechando el principio de óptimo, el tiempo de ejecución reduce en cierta forma porque el algoritmo trabaja por etapas tomando una decisión óptima en cada una y, por ende, en cada etapa tiene en cuenta la decisión óptima de la etapa anterior para decidir sobre la actual.

Con respecto a la técnica de algoritmos ávidos, dado que la función de selección nos lleva a seguir un camino determinado sin la posibilidad de retroceder, es difícil llegar al interés máximo. De ser posible, la función de selección será muy compleja y requiere de una justificación matemática para comprobar que llegará al óptimo global