Informe de Práctica 3 de Diseño y Análisis de Algoritmos

Programación Dinámica

Pablo Pastor Martín

Introducción:

En esta práctica nos adentramos en el estudio de los algoritmos basados en la estrategia de programación dinámica. Para ello, hemos implementado soluciones al problema de los menús saludables.

El llamado "Problema de los menús saludables" busca maximizar, dado un conjunto de platos, la aportación calórica de un menú sin superar un umbral de nutrientes pocos saludables. Podríamos considerar este problema como una variante del problema de la mochila, motivo por el cual tanto el pseudocódigo como el análisis de la complejidad de los algoritmos que solucionan el problema se asemejan a los que se hacen para el problema de la mochila.

Estructura óptima de los subproblemas:

Suponiendo una matriz(V) de tamaño m = |alimentos| +1 y n = maxInsanos+1. La entrada de esta matriz para un punto dado (i,j) viene determinado por el máximo valor nutricional total al considerar sólo los primeros i alimentos y con un umbral máximo de nutrientes poco saludables de j.

Generalizando, podremos afirmar que la entrada (|alimentos|+1, maxInsanos+1) contendrá la solución óptima considerando todos los platos y el límite real de alimentos poco saludables.

La afirmación anterior se basa en que, para cada elemento de la matriz existen dos opciones:

- No incluir el alimento i, por lo que la solución será la misma que para los primeros i-1 alimentos.
- Incluir el alimento i, añadiendo el valor nutricional del mismo al valor nutricional acumulado por los primeros i-1 alimentos cuando el umbral estaba en el límite actual(j) sustrayendo los nutrientes insanos del alimento i.

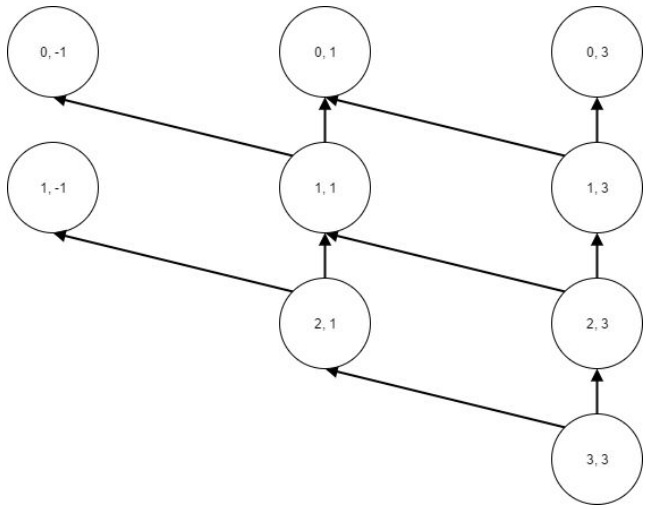
Generalizando obtenemos:

$$V_{i,j} = max\{V_{i-1,j},\ V_{i-1,\,j-insanos[i]} + valoresNutricionales[i]\},\ para\ i = 1...numAlimentos\ y\ j = 0....maxInsanos$$

Grafo de subproblemas:

Desde cada subproblema, como vimos antes, se usan 2 valores, incluyendo el alimento y no incluyéndolo, por tanto, no es difícil darse cuenta de que cada subproblema realiza dos llamadas recursivas a subproblemas. Para el caso propuesto con los datos siguientes y un tamaño máximo de nutrientes insanos de 3, se nos quedaría un problema tal que así:

Alimento	1	2	3
Nutrientes insanos	2	2	2



Leyenda: Cada nodo tiene una notación (x, y), donde 'x' es el alimento a tener en cuenta en esa llamada e 'y' es el máximo de nutrientes insanos para ese subproblema.

Como se puede observar, hay subproblemas que son resueltos más de una vez, por ejemplo (1, 1), motivo por el cual es útil la programación dinámica en este problema.

Pseudocódigos:

Pseudocódigo para la solución del problema mediante recursividad:

```
// Los alimentos se encuentran como atributos de la clase desarrollada
funcion menu(int n, int insanos)
     si n = 0 && insanos >= 0:
          return 0
     si insanos < 0:
          return -inf
     aux1 = menu(n - 1, insanos)
     aux2 = menu(n - 1, insanos -  insanos[n]) + valorNutricional[n]
     return max(aux1, aux2)
```

Pseudocódigo para la solución del problema mediante programación dinámica: Bottom-Up

```
// Los alimentos forman parte de la clase, así como la matriz de soluciones
parciales

función bottomUp():
     for(i = 0 to maxInsanos):
          matrizSoluciones[0][i] = 0;
     for(i = 1 to numPlatos):
          for(j = 0 to maxInsanos):
               if(insanos[i] <= j):
                    aux1 = matrizSoluciones[i-1][j]
                    aux2 = matrizSoluciones[i-1][j-insanos[i]]+valorNutricional[i]
                    matrizSoluciones[i][j] = max(aux1,aux2)
               else:
                    matrizSoluciones[i][j] = matrizSoluciones[i-1][j]
```

Pseudocódigo para la solución del problema mediante programación dinámica: Top-Down

```
// Los alimentos forman parte de la clase, así como la matriz de soluciones
parciales
// La matriz debería estar al principio con sus celdas iniciadas a -1

función TopDown(int n, int insanos):
     if(n = 0 && insanos>=0)
          return 0
     if(insanos < 0)
          return -inf
     if(matrizSoluciones[n][insanos] != -1)
          return matrizSoluciones[n][insanos]
     aux1 = TopDown(n-1, insanos)
     aux2 = TopDown(n-1, insanos - insanos[n]) + valorNutricional[n]
     matrizSoluciones[n][insanos]= max(aux1,aux2)
     return matrizSoluciones[n][insanos]
```

Análisis de la complejidad de los algoritmos implementados:

Denotaremos por n al número de platos y por M al máximo número de nutrientes perjudiciales:

Recursivo:

El algoritmo recursivo analiza todos los posibles subconjuntos de platos a partir de un conjunto con n alimentos, por lo que existen 2^n posibles subconjuntos. Además, dicho algoritmo lleva a cabo la comprobación de cada una de esas comprobaciones (Aunque al final sólo considere aquellas cuya suma de nutrientes insanos sea mayor que el máximo). Por lo que su tiempo de ejecución pertenece a $O(2^n)$.

Bottom-Up:

Si la matriz estuviera correctamente inicializada, el primer bucle del pseudocódigo no sería necesario, así que nos centraremos en el segundo bloque, que tiene dos bucles anidados.

En la parte más interna del bucle nos encontramos con sentencias ejecutadas en tiempo constante, así que la complejidad será aquella resultante de la anidación de los bucles.

Para cada iteración del primer bucle, nos encontramos con M iteraciones del segundo bucle, por lo que, al realizar el primero n iteraciones, nos encontramos con un tiempo de ejecución:
$T(n, M) = O(nM)$

Top-Down:

En el peor caso posible, que es cuando todos los platos tienen 1 sólo nutriente perjudicial (deberá analizar cada posible suma de nutrientes perjudiciales), deberemos comprobar para cada nodo su celda superior en la tabla, y la que tiene arriba a la izquierda. Al estar memorizados los resultados anteriores, cada uno de estos cálculos se realizarán una única vez, calculándose en realidad el triángulo superior de la matriz (incluyendo la diagonal principal).

Al ser de tamaño n*M dicha matriz, y basarnos en notación O. Concluímos que la complejidad pertenece a $O(nM)$.

Ejemplos de soluciones al problema:

- **Caso 1:**

```
8
170
Ensalada         10 10
Arvejas          60 20
CarneCabra       180 50
Rapadura         220 60
Escaldon         280 70
PapasArrugadas   300 75
ChocolateTirma   325 80
HeladoKalise     350 95
```

```
La solución aporta un valor de: 665

Platos:
- ChocolateTirma
- Escaldon
- Arvejas
```

- **Caso 2:**

```
7
170
Espaghetis             180 55
Pizza                  250 95
EnsaladaCesar          100 10
HeladoItaliano         130 30
EscalopeMilanesa       170 50
Focaccia               220 80
Lassagna               240 90
```

```
La solución aporta un valor de: 620

Platos:
- Focaccia
- EscalopeMilanesa
- HeladoItaliano
- EnsaladaCesar

Se tardó en ejecutar 1 milisegundos
```

Conclusiones:

En esta práctica hemos visto cómo la programación dinámica puede dar mejor resultado que el paradigma "Divide y vencerás" visto anteriormente en esta asignatura. Además, podemos ver que el problema genérico de la mochila puede ser variado para adaptarse a otros enunciados de problemas. A su vez, también hemos trabajado con dos formas de utilizar la programación dinámica: Top-Down y Bottom-Up, obteniendo, como de costumbre en programación dinámica, la misma complejidad con los dos enfoques.

Bibliografía utilizada:

- Apuntes de la asignatura en el campus virtual de la asignatura.
- *Dynamic Programming*, Mission-Peace, https://github.com/mission-peace/interview/wiki/Dynamic-Programming