Centro de Investigación en Matemáticas A.C.
Programación y Algoritmos 1 (2019)

Tarea 09 - Programación dinámica

Marco Antonio Esquivel Basaldua

Pregunta 1.

Este programa da solución al problema de knapsack tomando los valores a partir de un archivo de texto .txt que se lee como argumento de entrada del programa.

En el archivo .txt se obtiene lo siguiente:

Capacidad máxima de la mochila (V)

Total de objetos disponibles (n)

Dos columnas de n valores: la primera de ellas corresponde al volumen de cada objeto, la segunda al valor de cada objeto. Estos datos de van a guardar en dos arreglos, $volumen[]$ y $valores[]$ respectivamente.

En la carpeta compartida se incluyen cuatro ejemplos para este problema.

A partir de la programación dinámica que utiliza una matriz de $(n+1) \times (V+1)$ se aplica lo siguiente para llenar las casillas de la matriz $f[][]$:

$$f(i,j) = \begin{cases} f(i-1,j), & \text{si volumen(i)} > \text{j} \\ max(f(i-1,j), valores(i) + f(i-1, j-volumen(i))), & \text{en otro caso} \end{cases}$$

Ya que la fórmula anterior solo hace uso de la fila i y la fila $(i-1)$, en esta aplicación del problema se remplaza la matriz indicada por dos vectores de tamaño $(V+1)$ en los cuales se van guardando los valores de las filas i e $(i-1)$ correspondientes a la matriz. Estos vectores llevan por nombre $newrow[]$ y $oldrow[]$, $oldrow$ se inicializa con ceros y al final de cada iteración de i se actualiza con los valores de $newrow$. Al hacer este cambio, se pasa de tener una complejidad $O(nV)$ por una complejidad $O(V)$, ya que se tienen dos vectores de tamaño $V+1$.

Problema 2.

Este programa toma como entrada una secuencia de caracteres dentro de un archivo de texto .txt que se lee como argumento de entrada del programa. El primer dato de se lee de este archivo es un entero que indica el número de caracteres que serán leídos dentro del archivo. Al tener la cadena de caracteres como entrada se calcula el tamaño de la mayor secuencia palindrómica dentro de la cadena de caracteres.

La complejidad en tiempo del problema es $O(n^2)$ ya que sf(i,j) =

\begin{cases}
f(i-1,j),&\quad\text {si volumen(i) > j}\\
max(f(i-1,j),valores(i)+f(i-1,j-volumen(i))),&\quad\text{en otro caso}
\end{cases}f(i,j) =

\begin{cases}
f(i-1,j),&\quad\text {si volumen(i) > j}\\
max(f(i-1,j),valores(i)+f(i-1,j-volumen(i))),&\quad\text{en otro caso}

\end{cases}f(i,j) =
\begin{cases}
f(i-1,j),&\quad\text {si volumen(i) > j}\
max(f(i-1,j),valores(i)+f(i-1,j-volumen(i))),&\quad\text{en otro caso}
\end{cases}e trabaja con dos ciclos for anidados que recorren una matriz de tamaño $n \times n$. De igual forma la complejidad de memoria es $O(n^2)$.

Problema 3.

A partir de un archivo de texto .txt que se lee como argumento de entrada se toman los valores en el siguiente orden:

n: cantidad de partidos que deben ganar los Pumas para ganar el campeonato
p: partidos ganados por los Pumas hasta el momento
a: partidos ganados por las Águilas hasta el momento

Se calcula la probabilidad de que los Pumas ganen el campeonato a partir de los datos anteriores y suponiendo que en cada enfrentamiento cada equipo tiene la misma probabilidad de ganarlo (0.5).

El usuario puede modificar el archivo .txt para probar con distintos de n, p y a. Por default en la carpeta se incluye el archivo "data.txt" con los valores $n = 5$, $p = 1$ y $a = 0$.

Suponiendo que se dan como datos de entrada $p = j$ y $a = i$, para el partido en cuestión se pueden presentar dos resultados:

1. Que el pumas gane
    - En este caso los datos se actualizan de la siguiente forma: $p = (j + 1)$, $a = i$.
2. Que el pumas pierda
    - En este caso los datos se actualizan de la siguiente forma: $p = j$, $a = (i + 1)$.

Cada uno de los casos tiene la misma probabilidad de ocurrir (0.5) y estos se van a presentar cada vez hasta que p o a sean iguales a n con lo que se puede lograr f(i,j) =
\begin{cases}
f(i-1,j),&\quad\text {si volumen(i) > j}\
max(f(i-1,j),valores(i)+f(i-1,j-volumen(i))),&\quad\text{en otro caso}
\end{cases}organizar los datos para p y a que vayan desde 0 hasta n en una tabla de tamaño $(n + 1) \times (n + 1)$.

Para solucionar este problema se hace uso de la programación dinámica guardando los valores en una matriz llamada $table$ de tamaño $(n + 1) \times (n + 1)$ tomando como casos base:

- $a = p \neq n$, en este caso el valor cada equipo tiene la misma probabilidad de ganar (0.5).
- $p = n$, en este caso la probabilidad de que el pumas gane es 1 ya que ya habrá ganado los n partidos necesarios para coronarse campeón.
- $a = n$, en este caso los Pumas ya no pueden ser campeones ya que las Águilas han ganado n partidos.
- $a = p = n$, este caso genera un error ya que ambos partidos no pueden llegar a los n partidos al mismo tiempo.

Sólo es necesario llenar la forma triangular superior de la tabla ya que el resto corresponde al complemento para llegar a 1 de su posición transpuesta. La fórmula para llenar la tabla es la siguiente donde i corresponde a los partidos asociados a las Águilas y j a los Pumas:

$$

```
table(i,j) =
    \begin{cases}
    0.5,&\quad\text i=j\neq n\\
    1,&\quad\text j=n, i\neq n\\
    0,&\quad\text i=n,j\neq n\\
    -1,&\quad\text a=p=n\\
    0.5*table(i,j+1)+0.5*table(i+1,j),&\quad\text {e.o.c}
    \end{cases}
```

$$

$$table(i,j) = \begin{cases} 0.5, & \text{i} = j \neq n \\ 1, & \text{j} = n, i \neq n \\ 0, & \text{i} = n, j \neq n \\ -1, & \text{a} = p = n \\ 0.5 * table(i, j+1) + 0.5 * table(i+1, j), & \text{e.o.c} \end{cases}$$

$$table(i,j) = \begin{cases} 0.5, & \text{i} = j \neq n \\ 1, & \text{j} = n, i \neq n \\ 0, & \text{i} = n, j \neq n \\ -1, & \text{a} = p = n \; 0.5 * table(i, j+1) + 0.5 * table(i+1, j), \quad \text{e.o.c} \end{cases}$$