Laboratorio 6 | Algoritmos Codiciosos y Programación Dinámica

Geordie Quiroa

September 20, 2018

1 – Algortimo de fracciones egipcias

Cada fraccion positiva se puede representar como una suma de fracciones unitarias; las fracciones unitarias son aquellas donde el numerador es 1 y el denominador es un entero positivo.

Escribirlo en forma de pseudocodigo.

Pseudocódigo

Algorithm 1 Fracciones Egipcias

procedure FRACCEGIPCIAS($nume, deno$)
  $fraccUnit \leftarrow (nume/deno).cieling$
  $nume \leftarrow nume * fraccUnit\ -\ deno$
  $deno \leftarrow deno * fraccUnit$
  $Print(1/deno)$
 if $nume\ != \ 0$ **then**
   $\mathbf{fraccEgipcias(nume, deno)}$

Justificación
Este algoritmo es codicioso ya que no calcula todas las opciones posibles para determinar las fracciones unitarias (bottom up), sino que, al contrario, cada cálculo que se realizó fue a partir de un problema general (la fracción inicial) y cada uno de los valores resultantes del numerador y denominador se fueron reduciendo hasta que el valor del numerador fuera igual a cero. De la misma forma, no se creó una estructura de memoria auxiliar.

2 – Algoritmo Knapsack Fraccionado

Observaciones generales

Los valores definidos para el problema son los siguientes.

Item	Valor	Peso
Cobre	60	10
Plata	100	20
Oro	120	30

Figure 1: Valores iniciales del problema.

Para desarrollar el pseudocódigo de cada versión, se representó la figura 1 en una matriz M = Filas(3) x Columnas(3):

- M = [["Cobre", 60, 10], ["Plata", 100, 20], ["Oro", 120, 30]]
- El peso máximo para este problema es W = 50
- Los indices para esta matriz inician en 1.
- El total de filas, es el largo del arreglo.
- Ej: M[1][3] = 10; es decir hace referencia al peso del cobre.

El valor que este algoritmo retorna es el valor máximo que se puede obtener en base al peso límite definido.

2.1 – Knapsack Fraccionado Programación Dinámica

Knapsack Fraccionado

1. Para este algoritmo, se agregó una columna adicional a la matriz M, que almacena el Valor Por Unidad <VPU> (Valor/Peso) de cada fila con el fin de que sea fraccionario.
2. Uno de los subproblemas a resolver, es el escenario en el que la matriz no ordenada descendentemente respecto al Valor Por Unidad (VPU) calculado, en este caso, la matriz ya está ordenada.
3. La matriz resultante es: M = [["Cobre", 60, 10, 6], ["Plata", 100, 20, 5], ["Oro", 120, 30, 4]]

4. Cada fila en la matriz de KnapSack (KS), va a tener los valores: metal, Peso Acumulado (QuantAcum) y Valor Acumulado (ValorAcum).

Algorithm 2 KnapSack Programación Dinámica

procedure KNAPSACK(M, W)
  M **es la matriz base.**
  $Filas \leftarrow M.length$
  for $i \leftarrow 0$ to $Filas$ **do**
    $M[i][VPU] \leftarrow M[i][Valor]$ / $M[i][Peso]$
  $i \leftarrow 1$
  $KS \leftarrow [\ ][\ ]$
  while $KS[i-1][QuantAcum] < W$ **do**
    if $M[i][Peso] + KS[i-1][QuantAcum] <= W$ **then**
      $KS[i][Metal] \leftarrow M[i][Metal]$
      $KS[i][QuantAcum] \leftarrow M[i][Peso] + KS[i-1][QuantAcum]$
      $KS[i][ValorAcum] \leftarrow M[i][VPU] * quantity2take$
  $FilasKS \leftarrow KS.length$
  return KS[FilasKS][ValorAcum]

2.2 – Knapsack Fraccionado Codicioso

Algorithm 3 KnapSack Codicioso

procedure KNAPSACK($Array, W$)
  W **es el peso máximo variable.**
  $metal \leftarrow max(Array, Punidad)$
  if $W - metal.peso >= 0$ **then**
    $W \leftarrow W - metal.peso$
    $Valor \leftarrow metal.valor$
    $Array.remove(metal)$
    $return\ \ Valor + KnapSack(Array, W)$
  else
    $W \leftarrow (W - metal.peso) + metal.peso$
    $Valor \leftarrow W * metal.Punidad$
    $returnW$