Instituto Politécnico Nacional
Escuela Superior de Cómputo
Algoritmos Bioinspirados
Practica 1 – Programación Dinámica
Tinoco Videgaray Sergio Ernesto
Jorge Luis Rosas Trigueros
14/09/2022

Marco teórico:

La programación dinámica es un conjunto de métodos que permiten resolver un problema de manera óptima. Estos métodos consisten en la mezcla de elementos como la recursividad que permite reutilizar fragmentos de código para una operación, el paradigma divide y conquista que consiste en dividir el problema en varios subproblemas (llamados etapas) y un término denominado “memoización”, el cual consiste en almacenar el resultado de una operación realizada previamente en un búfer para utilizarla posteriormente cuando sea requerida, de tal forma que no sea necesario realizar dicha operación nuevamente, es decir, una especie de memoria cache.

Algunas características de la programación dinámica son:

- **ETAPAS:** Son cada uno de los pasos que se deben seguir para llegar a la solución. Se representan por medio de líneas discontinuas (Ver figura 1).

- **ESTADOS:** Son las diversas condiciones posibles que el sistema podría presentar en cada etapa del problema. Se representan por círculos (Ver figura 1).

- **POLÍTICA:** Es cualquiera de los caminos que llevan de la primera a la última etapa (Ver figura 1).

- **SUBPOLÍTICA:** Es un subconjunto de la política.

Algunas características de los problemas de la programación dinámica son las siguientes:

- El problema se puede dividir en etapas; cada etapa requiere una decisión. En muchos problemas de programación dinámica, la etapa es la cantidad de tiempo que pasa desde el inicio del problema, en ciertos casos no se necesitan decisiones en cada etapa.

- Cada etapa tiene un número de estados asociados con ella. Cada estado posee información que se necesita en cualquier etapa para tomar una decisión óptima.

- La decisión tomada en cualquier etapa indica cómo se transforma el estado de la etapa actual en el estado de la siguiente etapa. En muchos problemas, una decisión no determina con certeza el estado de la siguiente etapa; en lugar de ello, la decisión actual sólo determina la distribución de probabilidad del estado en la etapa siguiente.

- Dado el estado actual, la decisión óptima para cada una de las etapas restantes no debe depender de estados previamente alcanzados o de decisiones previamente tomadas. (principio de optimalidad)

A continuación, se presentan algunos ejemplos de problemas de programación dinámica.

Problema de la mochila

Para este problema se tienen N objetos cada uno con un peso w y un valor v asociado, además se tiene una mochila que puede transportar un peso máximo W (Ver figura 2).

El problema de la mochila consiste en seleccionar los objetos que van a entrar a la mochila de tal forma que se maximice el valor introducido de los N objetos sin superar el peso máximo soportado por la mochila.

Una generalización de la solución a este problema consiste en dividir el problema en 3 posibles casos o etapas:

- $m[0,w] = 0$; $m[i,0] = 0$

- $m[i,w] = m[i-1,w]$ si $w_i > w$

- $m[i,w] = \max(m[i-1,w], m[i-1, w-w_i] + v_i)$

Problema del cambio de monedas

En este problema se plantean M monedas de diferentes denominaciones y una cantidad de cambio ‘N’ que se debe proporcionar utilizando las diferentes monedas con las que se cuenta, de tal forma que se minimice la cantidad de monedas a utilizar.

La solución, utilizando la técnica de programación dinámica, consiste en completar una tabla de n filas y v+1 columnas donde: n es la cantidad de denominaciones diferentes (Ver figura 3).

Una generalización de la solución a este problema consiste en dividir el problema en 3 posibles casos o etapas:

- m[1,n] = n; m[i,0] = 0
- m[i,n] = m[i-1,n] if di > n
- m[i,n] = min(m[i-1,n], m[i, n-di] + 1)

Material y equipo:

-Equipo: Computadora con conexión a internet.

-IDE: Google Colab.

-Lenguaje de programación: Python 3.6

Desarrollo de la practica:

Problema de la mochila

Implementación en Python.

Para la implementación de este algoritmo se definieron 2 tuplas para los valores y el peso de los elementos, así como las constantes del peso máximo y el tamaño de datos.

En este ejemplo se tienen elementos con valor de 2, 3, 5, 5, 6 y peso de 2, 4, 4, 5, 7 respectivamente, así como un peso máximo de 9 (Ver tabla 1).

```
import numpy as np

v=(2,3,5,5,6) #Tupla v
w=(2,4,4,5,7) #Tupla w
W=9 #Peso máximo W
N=len(v)  #Tamaño de la tupla v (5)
```

De igual forma se definió una matriz nula de 6x10 celdas con la cual se va a trabajar.

```
m=np.zeros((N+1,W+1)) #Matriz de 0's de 6x10
```

Para la implementación del algoritmo se definen dos bucles for anidados para recorrer la matriz nula a partir del elemento (1,1) hasta el final de la matriz.

Aplicando una estructura condicional se determina si el peso del elemento w_i es mayor a w (en este caso la columna), se va a asignar el valor m_{i-1} es decir el elemento arriba del actual.

En caso contrario se va a buscar el máximo entre el elemento de arriba o bien el elemento de la fila de arriba recorrido $w-w_i$ a la izquierda y sumándole el valor de ese elemento.

Finalmente imprimimos la matriz resultante.

```
#Implementacion del algoritmo
for row in range(1,N+1):
  for col in range(1,W+1):
    if w[row-1]>col:
      m[row][col]=m[row-1][col]
    else:
      m[row][col]=max(m[row-1][col],m[row-1][col-w[row-1]]+v[row-1])
```

```
m
```

Resultados de la ejecución:

```
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.],
       [ 0.,  0.,  2.,  2.,  3.,  3.,  5.,  5.,  5.,  5.],
       [ 0.,  0.,  2.,  2.,  5.,  5.,  7.,  7.,  8.,  8.],
       [ 0.,  0.,  2.,  2.,  5.,  5.,  7.,  7.,  8., 10.],
       [ 0.,  0.,  2.,  2.,  5.,  5.,  7.,  7.,  8., 10.]])
```

Problema del cambio de monedas:

Implementación en Python.

Para esta implementación se va a crear únicamente una tupla con los valores v de las monedas de diferentes denominaciones, así como las constantes del tamaño del la tupla y el cambio a devolver.

En este ejemplo se tienen monedas de 1, 2 y 5 unidades y un cambio de 7 (Ver tabla 2).

```
import numpy as np

v=(1,2,5) #Tupla v
N=7
W=len(v)  #Tamaño de la tupla v (3)
```

De igual forma se definió una matriz nula de 6x10 celdas con la cual se va a trabajar.

```
m=np.zeros((W,N+1)) #Matriz de 0's de 3x8
```

Primeramente, se recorre la primera fila de la matriz asignando los valores i- ésimo a cada celda por medio de un bucle for.

```
for i in range(N+1):
    m[0][i]=i
```

Posteriormente se va a recorrer el resto de la matriz usando 2 bucles for anidados empezando por la celda (1,1).

Posteriormente se va a evaluar si el valor del i-ésimo elemento es mayor a la columna en la que nos encontramos y de ser el caso se asigna el valor i-1 de la matriz, es decir el valor de la celda de arriba.

En caso de no cumplirse la condición se va a buscar el valor mínimo entre el valor i-1 de la matriz y el valor de la misma fila recorrido n-di unidades a la izquierda y sumando 1 al final.

Finalmente imprimimos la matriz resultante.

```
#Implementacion del algoritmo
for row in range(1,W):
  for col in range(1,N+1):
    if v[row]>col:
      m[row][col]=m[row-1][col]
    else:
      m[row][col]=min(m[row-1][col],m[row][col-v[row]]+1)

m
```

Resultados de la ejecución:

```
array([[0., 1., 2., 3., 4., 5., 6., 7.],
       [0., 1., 1., 2., 2., 3., 3., 4.],
       [0., 1., 1., 2., 2., 1., 2., 2.]])
```

Diagramas:

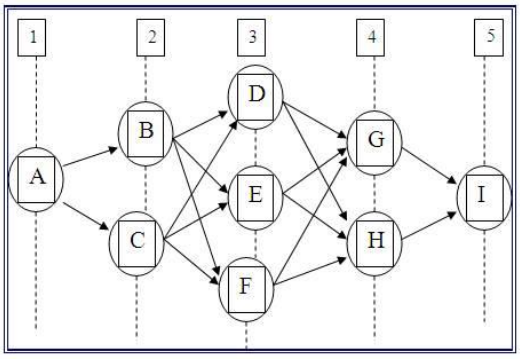

Figura 1. Representación de una solución en programación dinámica.

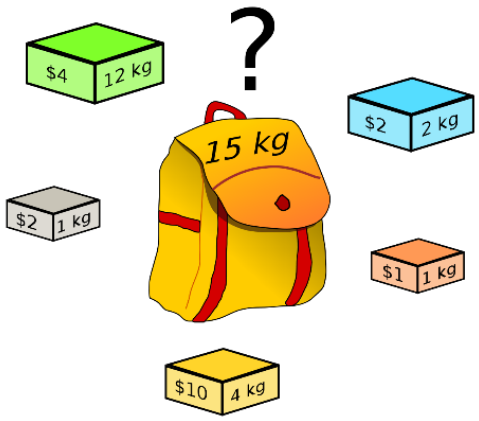

Figura 2. Representación gráfica del problema de la mochila.

Figura 3. Representación gráfica del problema del cambio de monedas

Tablas:

0	0	0	0	0	0
0	6	6	6	6	6
0	6	10	16	16	16
0	6	10	16	18	22

Tabla 1. Resultados del ejemplo 1.

0	1	2	3	4	5	6	7
0	1	1	2	2	3	3	4
0	1	1	2	2	1	2	2

Tabla 2. Resultados del ejemplo 2.

Conclusiones:

La programación dinámica es un conjunto de técnicas para la optimización de soluciones de distintos problemas donde una solución requiere una serie de decisiones de forma secuencial. La implementación de estos problemas nos permite dar solución a problemas de la vida real abstrayendo los elementos que los componen como en el caso del problema de la mochila donde cada valor representa un elemento de la vida real. O como es el caso del problema del cambio de monedas en donde se requiere utilizar la menor cantidad de monedas para devolver una cantidad especifica de cambio en contraste con el problema de la mochila donde se busca maximizar una cantidad, en el problema del cambio de monedas se busca minimizar una cantidad, pero a fin de cuentas lo que se busca es optimizar una solución, ya sea buscando un mínimo o un máximo. Y de igual forma se hace uso de la memoización al utilizar los valores previamente calculados y almacenados en las matrices para realizar los cálculos posteriores y aplicar el algoritmo en base a estos valores sin necesidad de volver a hacer estas operaciones.

Como practica personal decidí modificar los valores de las tuplas en ambos ejemplos añadiendo 2 valores al final, así como también las constantes del peso y el cambio en ambos problemas, de tal forma que los resultados obtenidos fueron los mismos realizados manualmente dado por concluido que el algoritmo funciona para cualquier conjunto de valores dado.

Bibliografía:

Turmero, P. (2015, 6 mayo). *Programación dinámica*. Monografias.com. Recuperado 14 de septiembre de 2022, de https://www.monografias.com/trabajos104/la-programacion-dinamica/la-programacion-dinamica

Hidalgo, U. A. D. E. de. (s. f.). *Boletín Científico :: UAEH*. Recuperado 14 de septiembre de 2022, de https://www.uaeh.edu.mx/scige/boletin/tlahuelilpan/n6/e2.html