



Instituto Politécnico Nacional

Escuela Superior de Cómputo



Unidad de académica:
Análisis y diseño de algoritmos


Alumno:

Solares Velasco Arturo Misael 2023630538

Solis Lugo Mayra 2023630449

Grupo: 3CV1

Profesor : Floriano García Andres



La programación dinámica es una técnica ampliamente utilizada para resolver una variedad de problemas de optimización en informática. Se basa en la idea de descomponer un problema en subproblemas más pequeños y resolverlos de forma recursiva, almacenando las soluciones intermedias para evitar recálculos innecesarios. En este reporte, se presenta un análisis detallado de la implementación y el rendimiento de algoritmos de programación dinámica para resolver dos problemas comunes: el problema del cambio y el problema de la mochila 0/1. Se comparan los resultados obtenidos al implementar los algoritmos en tres lenguajes de programación populares: Python, Java y C.

Se implementaron programas en los tres lenguajes para resolver los problemas del cambio y la mochila 0/1 utilizando la técnica de programación dinámica. Cada programa fue diseñado para aceptar diferentes escenarios de prueba y calcular el tiempo de ejecución y el consumo de memoria para cada uno. Se utilizaron 5 escenarios diferentes para cada problema, con conjuntos de datos variados para evaluar el rendimiento de los algoritmos en situaciones diversas. Los programas fueron ejecutados y los resultados fueron registrados para su análisis.

```
import time
import tracemalloc


def cambio_monedas(monedas, cantidad):
    dp = [float('inf')] * (cantidad + 1)
    dp[0] = 0
    for i in range(1, cantidad + 1):
        for moneda in monedas:
            if i - moneda >= 0:
                dp[i] = min(dp[i], dp[i - moneda] + 1)
    return dp[cantidad] if dp[cantidad] != float('inf') else -1

escenarios = [
    (11, [1, 2, 5]),
    (30, [1, 2, 5, 10]),
    (99, [1, 5, 10, 25, 50]),
    (3, [2, 5, 10]),
    (7, [2, 4])
]
```

```
import java.util.Arrays;

public class CambioMonedas {
    public static int cambioMonedas(int[] monedas, int cantidad) {
        int[] dp = new int[cantidad + 1];
        Arrays.fill(dp, cantidad + 1);
        dp[0] = 0;
        for (int i = 1; i <= cantidad; i++) {
            for (int moneda : monedas) {
                if (i - moneda >= 0) {
                    dp[i] = Math.min(dp[i], dp[i - moneda] + 1);
                }
            }
        }
        return dp[cantidad] > cantidad ? -1 : dp[cantidad];
    }

    public static void main(String[] args) {
        int[][] escenarios = {
            {11, 1, 2, 5},
            {30, 1, 2, 5, 10},
            {99, 1, 5, 10, 25, 50},
            {3, 2, 5, 10},
            {7, 2, 4}
        };
    }
}
```



```

import time
import tracemalloc

def mochila_01(pesos, valores, capacidad):
    n = len(pesos)
    dp = [[0 for _ in range(capacidad + 1)] for _ in range(n + 1)]
    for i in range(1, n + 1):
        for w in range(1, capacidad + 1):
            if pesos[i-1] <= w:
                dp[i][w] = max(dp[i-1][w], dp[i-1][w-pesos[i-1]] + valores[i-1])
            else:
                dp[i][w] = dp[i-1][w]
    return dp[n][capacidad]

escenarios = [
    (4, [1, 2, 3], [10, 20, 30]),
    (10, [2, 3, 5, 7], [1, 4, 5, 7]),
    (15, [3, 4, 6, 8], [2, 3, 1, 5]),
    (10, [5, 5, 5], [10, 20, 30]),
    (5, [6, 7, 8], [10, 12, 15])
]

```

```

public class Mochila01 {
    public static int mochila01(int[] pesos, int[] valores, int capacidad) {
        int n = pesos.length;
        int[][] dp = new int[n + 1][capacidad + 1];
        for (int i = 1; i <= n; i++) {
            for (int w = 1; w <= capacidad; w++) {
                if (pesos[i - 1] <= w) {
                    dp[i][w] = Math.max(dp[i - 1][w], dp[i - 1][w - pesos[i - 1]] + valores[i - 1]);
                } else {
                    dp[i][w] = dp[i - 1][w];
                }
            }
        }
        return dp[n][capacidad];
    }

    public static void main(String[] args) {
        int[][][] escenarios = {
            {{4}, {1, 2, 3}, {10, 20, 30}},
            {{10}, {2, 3, 5, 7}, {1, 4, 5, 7}},
            {{15}, {3, 4, 6, 8}, {2, 3, 1, 5}},
            {{10}, {5, 5, 5}, {10, 20, 30}},
            {{5}, {6, 7, 8}, {10, 12, 15}}
        };
    }
}

```

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/resource.h>
#include <limits.h>

int cambio_monedas(int monedas[], int n, int cantidad) {
    int *dp = (int *)malloc((cantidad + 1) * sizeof(int));
    for (int i = 0; i <= cantidad; i++) {
        dp[i] = INT_MAX;
    }
    dp[0] = 0;
    for (int i = 1; i <= cantidad; i++) {
        for (int j = 0; j < n; j++) {
            if (i - monedas[j] >= 0 && dp[i - monedas[j]] != INT_MAX) {
                dp[i] = dp[i] < dp[i - monedas[j]] + 1 ? dp[i - monedas[j]] + 1 : dp[i];
            }
        }
    }
    int resultado = dp[cantidad] == INT_MAX ? -1 : dp[cantidad];
    free(dp);
    return resultado;
}

int main() {
    int escenarios[5]
}

```

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/resource.h>

int mochila_01(int pesos[], int valores[], int n, int capacidad) {
    int **dp = (int **)malloc((n + 1) * sizeof(int *));
    for (int i = 0; i <= n; i++) {
        dp[i] = (int *)malloc((capacidad + 1) * sizeof(int));
        for (int j = 0; j <= capacidad; j++) {
            dp[i][j] = 0;
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int w = 1; w <= capacidad; w++) {
            if (pesos[i - 1] <= w) {
                dp[i][w] = dp[i - 1][w] > dp[i - 1][w - pesos[i - 1]] + valores[i - 1] ? dp[i - 1][w - pesos[i - 1]] + valores[i - 1] : dp[i - 1][w];
            } else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }
    return dp[n][capacidad];
}

```

Resultados programa de la mochila

main.c

```
5 #include <string.h>
6
7 int mochila01(int* pesos, int* valores, int n, int capacidad) {
8     int** dp = (int**)malloc((n + 1) * sizeof(int*));
```

input

Capacidad	Pesos	Valores	Resultado	Tiempo	Memoria
4	[1, 2, 3]	[10, 20, 30]	40	0.01 ms	0 KB
10	[2, 3, 5, 7]	[1, 4, 5, 7]	11	0.00 ms	0 KB
15	[3, 4, 6, 8]	[2, 3, 1, 5]	10	0.00 ms	0 KB
10	[5, 5, 5]	[10, 20, 30]	50	0.00 ms	0 KB
5	[6, 7, 8]	[10, 12, 15]	0	0.00 ms	0 KB

main.py

```
3
4 def mochila_01(pesos, valores, capacidad):
5     n = len(pesos)
6     dp = [[0 for _ in range(capacidad + 1)] for _ in range(n + 1)]
```

input

Capacidad	Pesos	Valores	Resultado	Tiempo	Memoria
4	[1, 2, 3]	[10, 20, 30]	40	0.03 ms	0.29 KB
10	[2, 3, 5, 7]	[1, 4, 5, 7]	11	0.02 ms	0.32 KB
15	[3, 4, 6, 8]	[2, 3, 1, 5]	10	0.02 ms	0.32 KB
10	[5, 5, 5]	[10, 20, 30]	50	0.01 ms	0.29 KB
5	[6, 7, 8]	[10, 12, 15]	0	0.00 ms	0.29 KB

Mochila01.java :

```
1
2
3
4 valores, int capacidad) {
```

input

Capacidad	Pesos	Valores	Resultado	Tiempo	Memoria
4	[1, 2, 3]	[10, 20, 30]	40	0 ms	499 KB
10	[2, 3, 5, 7]	[1, 4, 5, 7]	11	0 ms	645 KB
15	[3, 4, 6, 8]	[2, 3, 1, 5]	10	0 ms	645 KB
10	[5, 5, 5]	[10, 20, 30]	50	0 ms	645 KB
5	[6, 7, 8]	[10, 12, 15]	0	0 ms	646 KB

Resultados programa del cambio de monedas

main.c

```

6
7- int cambioMonedas(int* monedas, int tam, int cantidad) {
8    int* dp = (int*)malloc((cantidad + 1) * sizeof(int));
9-    for (int i = 0; i <= cantidad; i++) {

```

input

Cantidad	Monedas	Resultado	Tiempo	Memoria
11	[1, 2, 5, 0]	3	0.01 ms	0 KB
30	[1, 2, 5, 10]	3	0.00 ms	0 KB
99	[1, 5, 10, 25]	9	0.00 ms	0 KB
3	[2, 5, 10, 0]	-1	0.00 ms	0 KB
7	[2, 4, 0, 0]	-1	0.00 ms	0 KB

main.py

```

4- def cambio_monedas(monedas, cantidad):
5    dp = [float('inf')] * (cantidad + 1)
6    dp[0] = 0
7-    for moneda in monedas:

```

input

Cantidad	Monedas	Resultado	Tiempo	Memoria
11	[1, 2, 5]	3	0.03 ms	0.26 KB
30	[1, 2, 5, 10]	3	0.04 ms	0.41 KB
99	[1, 5, 10, 25, 50]	8	0.13 ms	0.96 KB
3	[2, 5, 10]	-1	0.00 ms	0.20 KB
7	[2, 4]	-1	0.01 ms	0.21 KB

CambioMonedas.ja...

```

57- public static String padRight(String s, int n) {
58    return String.format("%-" + n + "s", s);
59    }
60
61- public static void main(String[] args) {
62-    int[][] escenarios = {
63        {11, 1, 2, 5},
64        {30, 1, 2, 5, 10},

```

input

Cantidad	Monedas	Resultado	Tiempo	Memoria
11	[1, 2, 5]	3	0 ms	499 KB
30	[1, 2, 5, 10]	3	0 ms	645 KB
99	[1, 5, 10, 25, 50]	8	0 ms	645 KB
3	[2, 5, 10]	-1	0 ms	645 KB
7	[2, 4]	-1	0 ms	645 KB



Eficiencia de Tiempo

C demostró ser el más eficiente en términos de tiempo de ejecución para ambos problemas, con tiempos prácticamente inmediatos. Esto se debe a la naturaleza de bajo nivel de C y su capacidad para manejar operaciones directamente en el hardware sin la sobrecarga de la administración de tiempo de ejecución y compilación de bytecode presentes en Python y Java.

Python mostró un excelente rendimiento con tiempos de ejecución muy rápidos, lo que lo hace adecuado para prototipos y desarrollos rápidos donde la rapidez de implementación y facilidad de uso son importantes.

Java tuvo tiempos de ejecución ligeramente más altos, pero aún dentro de un rango muy eficiente, beneficiándose de su robusto ecosistema y características de programación orientada a objetos.

Eficiencia de Memoria

C también lideró en términos de eficiencia de memoria, utilizando la menor cantidad de memoria entre los tres lenguajes debido a su capacidad para gestionar la memoria de manera más precisa y sin la sobrecarga de los entornos de ejecución.

Python y Java tuvieron un uso de memoria similar, con Python ligeramente más eficiente que Java, lo que podría ser una consideración en sistemas con restricciones de memoria.

