**Actividad.** Análisis de programación dinámica

Nombre: Maria Guadalupe Soria Velazquez, Emilio Leví Díaz Abarde, Daniel Aguilar Matrícula: A01710797, A01620887

# Objetivo:

* Poner en práctica los conceptos y términos de programación dinámica (PD).

**Instrucciones:** Realiza lo que se solicita en cada sección:

1. Supón que vives en un país donde sólo están disponibles las monedas de 1, 4 y 9. Se debe diseñar un algoritmo para pagar una cantidad utilizando el menor número posible de monedas.
   * *Muestra la tabla generada en el proceso de solución para una cantidad = 10.*

**Cantidad mínima de monedas necesarias para cambiar una cantidad ‘C’= dp[c]**

Cantidad a cambiar = 0: dp [0] = 0

Cantidad a cambiar = 1: dp [1] = 1 (una moneda de 1)

Cantidad a cambiar = 2: dp [2] = 2 (dos monedas de 1)

Cantidad a cambiar = 3: dp [3] = 3 (tres monedas de 1)

Cantidad a cambiar = 4: dp [4] = 1 (una moneda de 4)

Cantidad a cambiar = 5: dp [5] = 2 (una moneda de 4 y una moneda de 1)

Cantidad a cambiar = 6: dp [6] = 3 (una moneda de 4 y dos monedas de 1)

Cantidad a cambiar = 7: dp [7] = 4 (una moneda de 4 y tres monedas de 1)

Cantidad a cambiar = 8: dp [8] = 2 (dos monedas de 4)

Cantidad a cambiar = 9: dp [9] = 1 (una moneda de 9)

Cantidad a cambiar = 10: dp [10] = 2 (una moneda de 9 y una moneda de 1)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Valor** | **Cantidad a cambiar** | | | | | | | | | | |
| **moneda** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **v1 = 1** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| **v2 = 4** | 0 | 1 | 2 | 3 | 1 | 2 | 3 | 4 | 2 | 3 | 4 |
| **v3 = 9** | 0 | 1 | 2 | 3 | 1 | 2 | 3 | 4 | 2 | 1 | 2 |

1. Supón que vives en un país donde sólo están disponibles las monedas de 10, 7 y 1. Se debe diseñar un algoritmo para pagar una cantidad utilizando el menor número posible de monedas.
   * *Muestra la tabla generada en el proceso de solución para una cantidad = 15.*

**Cantidad mínima de monedas necesarias para cambiar una cantidad ‘C’= dp[c]**

Cantidad a cambiar = 0: dp [0] = 0

Cantidad a cambiar = 1: dp [1] = 1 (una moneda de 1)

Cantidad a cambiar = 2: dp [2] = 2 (dos monedas de 1)

Cantidad a cambiar = 3: dp [3] = 3 (tres monedas de 1)

Cantidad a cambiar = 4: dp [4] = 4 (cuatro monedas de 1)

Cantidad a cambiar = 5: dp [5] = 5 (cinco monedas de 1)

Cantidad a cambiar = 6: dp [6] = 6 (seis monedas de 1)

Cantidad a cambiar = 7: dp [7] = 1 (una moneda de 7)

Cantidad a cambiar = 8: dp [8] = 2 (una moneda de 7 y una de 1)

Cantidad a cambiar = 9: dp [9] = 3 (una moneda de 7 y dos de 1)

Cantidad a cambiar = 10: dp [10] = 1 (una moneda de 10)

Cantidad a cambiar = 11: dp [11] = 2 (una moneda de 10 y una de 1)

Cantidad a cambiar = 12: dp [12] = 3 (una moneda de 10 y dos de 1)

Cantidad a cambiar = 13: dp [13] = 4 (una moneda de 10 y tres de 1)

Cantidad a cambiar = 14: dp [14] = 2 (una moneda de 7 y una moneda de 7)

Cantidad a cambiar = 15: dp [15] = 3 (una moneda de 10 y una moneda de 7 y una de 1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Valor** | **Cantidad a cambiar** | | | | | | | | | | | | | | | |
| **moneda** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **v1 = 1** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| **v2 = 7** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 | 10 | 11 | 12 | 13 | 2 | 3 |
| **v3 = 10** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 | 1 | 2 | 3 | 4 | 2 | 3 |

1. Considera la variante de los números de Fibonacci, que denominaremos “números de burronacci”, definida a continuación. El *n-ésimo* número de burronacci es igual a 4 veces el número *(n-1)-ésimo*, más 2 veces el *(n-2)-ésimo*, menos el *n-ésimo* número de burronacci. El primer y el segundo número de burronacci valen 1 y 2, respectivamente. Se pide lo siguiente:
   * Escribe un algoritmo con PD para el cálculo del *n-ésimo* número de burronacci.
   * Indica claramente la complejidad del algoritmo.
   * Muestra las salidas esperadas del algoritmo en la tabla anexa.

int fibonacci\_PD (int n) { int i;

int[] f=new int[n+1]; f[0] = 1;

if (n > 0) { f[1] = 1

for (i=2 ; i<=n ; i++) f[i] = f[i‐1] + f[i‐2];

}

return f[n];

}

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Burronacci** | **Resultados** | | | | | | | | |
| Valores | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| **Serie** | 1 | 2 | 10 | 44 | 196 | 872 | 3880 | 17264 | 76816 |

# Algoritmo:

Algoritmo de los números de burronacci:

Int Burronacci\_PD(int n){

If n is less than 3

Return n

Else

Return 4 \* Burronacci (n – 1) + 2 \* Burronacci(n – 2)

Complejidad: O(2^n)

1. El número de permutaciones de *n* objetos se puede expresar usando el sistema recurrente.

# Ejemplo:

|  |  |
| --- | --- |
| **P(0) = 1** | Ya que solamente hay una permutación de cero elementos, y … |
| **P(n) = n\*P(n-1)** | Porque podemos elegir un elemento de las *n* posibles y después cualquier secuencia que sea una permutación de los *n-1* restantes. |

Si A = {a, b, c} para P(3) se tiene: P(0) = 1

P(1) = 1\*(1) = 1 {a}

P(2) = 2\*(1) = 2 {ab, ba}

* + Se desea diseñar un algoritmo de PD que devuelva el número de permutaciones que se pueden obtener de *n* objetos.
  + Indica claramente la complejidad del algoritmo.

P(3) = 3\*(2) = 6 {abc, acb, bca, bac, cab, cba}

P(4) = ? {abcd, abdc, acdb, acbd, … }

…

Algoritmo:

#include <iostream>

using namespace std;

// Función para calcular el número de permutaciones usando Programación Dinámica

int permutaciones(int n) {

// Complejidad Espacial: O(n)

// Crear un array para almacenar los resultados de subproblemas

int P[n + 1];

// Caso base: P(0) = 1

P[0] = 1;

// Complejidad Temporal: O(n)

// Rellenar el array usando la relación de recurrencia P(i) = i \* P(i-1)

for (int i = 1; i <= n; i++) {

P[i] = i \* P[i - 1];

}

// Retornar el resultado para P(n)

return P[n];

}

int main() {

// Definir el valor de n

int n = 4;

// Llamar a la función permutaciones y mostrar el resultado

cout << "El número de permutaciones de " << n << " objetos es: " << permutaciones(n) << endl;

return 0;

}