Tecnológico de Monterrey

ESPY

Proyecto final

Omar David Hernández A01383543
Bernardo García A00570682

23 de noviembre 2022
Diseño de compiladores

Contents

1 Introducción

La programación dinámica es una técnica para resolver problemas con subproblemas superpuestos.

Por lo general, estos subproblemas surgen de una recurrencia que relaciona la solución de un problema dado con las soluciones de sus subproblemas más pequeños. En vez de resolver subproblemas superpuestos una y otra vez, la programación dinámica sugiere resolver cada uno de los subproblemas más pequeños solo una vez. (Rodríguez E, 2021).

2 Definición formal del problema

Formalmente se plantea de la siguiente manera; dada una cantidad N, que es un entero positivo, encontrar un conjunto finito de números enteros no negativos (las monedas) $x1, x2, ..., x_n$, donde se busca minimizar el número total de monedas que se utilizan para sumar N. (Kozen, D. et.al, s.f)

3 Descripción detallada

Algoritmo 1: Dynamic Programming Coin Change

Autor: Bernardo García, Carlos Arroyo

Input: Número entero n con la cantidad de dinero que se espera, número entero q con la cantidad de monedas disponibles, y arreglo de enteros C[ ] con las cantidades de cada moneda disponible.

Output: Arreglo de enteros $coinsUsed[]$ con las monedas usadas en el mínimo cambio para la cantidad n. El número de operaciones básicas empleadas para encontrar la solución, el tiempo en segundos de la ejecución.

Contar Monedas:
for i $\leftarrow$ 0 to q
    for j $\leftarrow$ 0 to n
        if j - C[i] $\geq$ 0
            amount[j] $\leftarrow$ menor(amount[j], amount[j-C[i]] +1)
            coinsChecked [j] $\leftarrow$ c[i]
if amount[n] = 0 $\rightarrow$ print "No full solution"
else $\rightarrow$ Checar monedas

4 Análisis de complejidad

Algoritmo 1: Dynamic Programming Coin Change

Contemplando las siguientes observaciones:

1. q es la cantidad de valores de monedas
2. n es el monto de cambio solicitado
3. La comparación if es la operación básica, pasando $n * q$ veces

4. Se contempla también el recorrido de *Checar monedas* donde *pos* es un iterador y *amount* es la cantidad de monedas que se necesitan para el monto n

5. Formulación:

$$\sum_{i=0}^{q}\sum_{j=0}^{n}1+\sum_{pos=0}^{amount}1 \tag{1}$$

La complejidad temporal es de orden: $O(n*q)+O(amount)$

Contemplando el uso de un arreglo de montos, un arreglo de monedas iteradas, y un arreglo de monedas usadas, la complejidad espacial es de orden: $O(2q+amount)$

5 Experimentación

Ambos algoritmos fueron probados con 15 diferentes instancias de tamaños incrementables, cada uno ejecutado 10 veces para llevar un registro de tiempo y operaciones menos variable. (Anexo 1)

Notación

n	Monto solicitado de cambio en monedas
q	Cantidad de monedas disponibles (tamaño de entrada)
T	Promedio del tiempo de ejecución en segundos
B	Promedio de operaciones básicas de ejecución

Los resultados son los siguientes:

5.1 Algoritmo 1: Dynamic Programming Coin Change

5.1.1 Condiciones

- Lenguaje de programación: C++
- Banderas de compilación: NA
- Equipo: Windows 10 v 2005 x-64 bit, Processor: i5-8250U. RAM: 8GB.
- IDE: VSCode

5.1.2 Ejecución

*Las notas en amarillo hacen referencia a errores anotados del algoritmo Greedy

Instancia	*n*	*q*	T	B	Notes
test01	11	3	0.0052	28	
test02	98	7	0.0058	506	
test03	38	10	0.0186	157	
test04	147	25	0.0043	3348	
test05	266	30	0.0047	5755	
test06	1365	50	0.0059	61550	
test07	5000	100	0.0085	474713	Had a successful attempt with n 4690
test08	6750	130	0.0135	844734	Had a successful attempt with n 1344
test09	9467	150	0.0164	1383868	
test10	9500	200	0.018	1796319	Had a successful attempt with n 9000
test11	15266	250	0.0294	3693577	
test12	13000	300	0.0302	3751352	
test13	263	370	0.0053	33076	
test14	23456	420	0.0719	9705035	
test15	45150	500	0.1645	22450250	

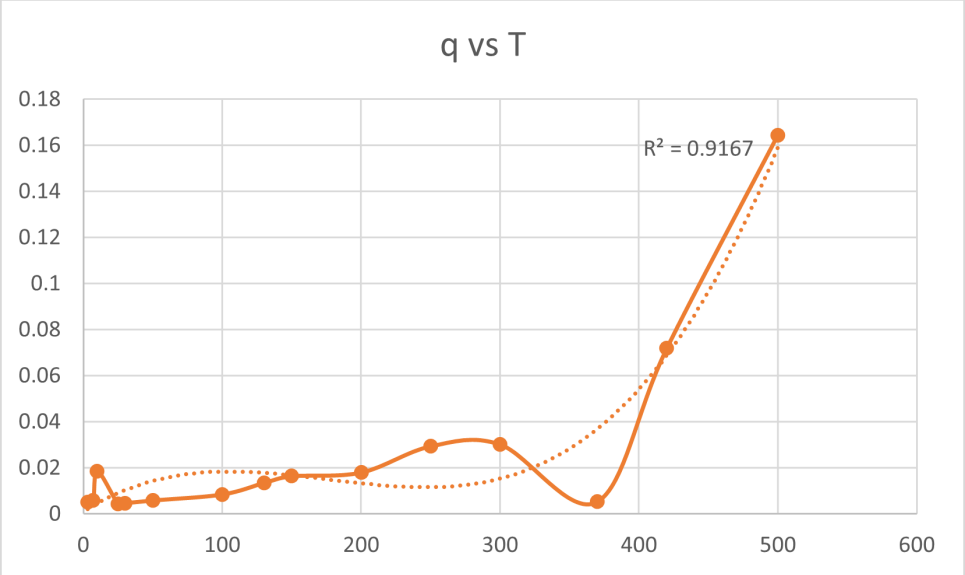

Figure 1: q vs T. Línea de tendencia polinómica clase 3; $y = 5E-09x^3 - 3E-06x^2 + 0.0004x + 0.0011$, $R^2 = 0.9167$

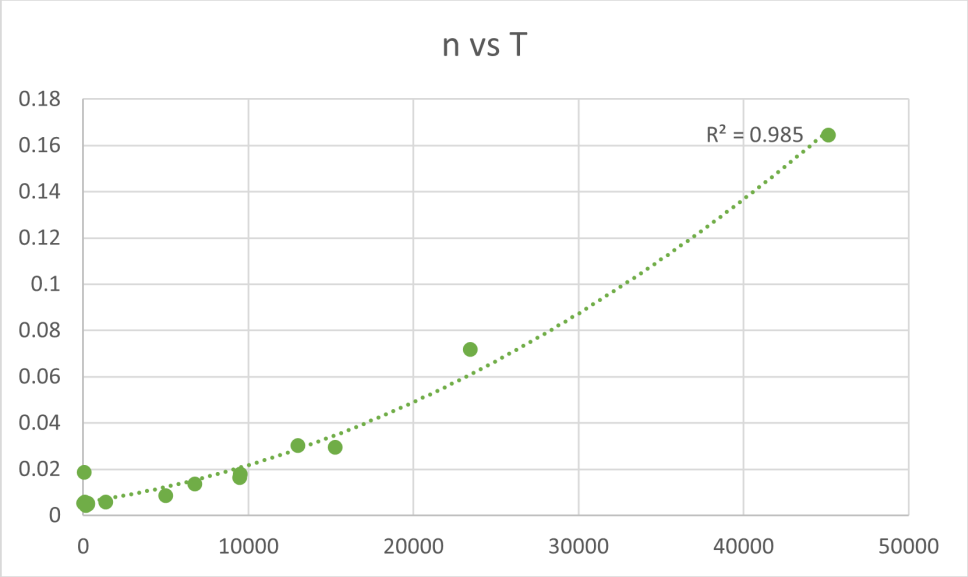

Figure 2: n vs T. Línea de tendencia polinómica clase 2; $y = 6E-11x^2 + 1E-06x + 0.0057$, $R^2 = 0.985$

5.1.3 Análisis de resultados

Es notable la enorme cantidad de operaciones que realiza este algoritmo conforme el tamaño de entrada de datos aumenta. Tomando esto en cuenta, el tiempo de ejecución es notablemente bajo en tamaños de entrada de menos de 130 datos, menor que los resultados del algoritmo Greedy aunque no por mucho, pero entradas mayores a dicho número su tiempo va creciendo exponencialmente (con una línea de tendencia polinómica).

5.2 Video demostración

Link al video de demostración del proyecto: https://youtu.be/-5Ckw4Ocauk

6 Aportaciones

Aportaciones	
Carlos	Investigación, documentación, pruebas de código
Bernardo	Planificación, desarrollo y depuración de códigos, pruebas de código, ejecuciones de algoritmos, tablas y análisis de resultados

7 Aprendizajes

Aprendizajes			
	Aprendizaje del curso	Lo que más le agradó	Lo que no le agradó
Carlos	La variedad y la lógica detrás de algunos algoritmos muy interesantes.	Ciertos algoritmos y la disponibilidad y actitud del maestro.	Ver tantos algoritmos por encima, en vez de ver pocos de mayor importancia y profundizar.
Bernardo	Aprender la importancia de eficientar un algoritmo, y la lógica de varios ejemplos concretos.	El orden y la organización de las clases, la flexibilidad y disponibilidad del profesor a todas horas.	En ocasiones la falta de imágenes y ejemplos, y la tremenda dificultad de ciertas tareas tras poca práctica en las clases.

8 Conclusiones

9 Referencias bibliográficas

Rodriguez Eduardo, (2021). Técnicas para el diseño de algoritmos. Algoritmos voraces (greedy) Sesión 15, p 4.

Rodriguez Eduardo, (2021). Técnicas para el diseño de algoritmos. Programación dinámica Sesión 14, p 3.

Kozen, D. Zaks, S. (s.f) Optimal Bounds for the ChangeMaking Problem. Recuperado de: https://www.cs.cornell.edu/ kozen/Papers/change.pdf

Khov, Try (2020). Solving Minimum Coin change. Recuperado de: https://trykv.medium.com/how-to-solve-minimum-coin-change-f96a758ccade

Back To Back SWE (2019). The Change Making Problem - Fewest Coins To Make Change Dynamic Programming. Video from: https://www.youtube.com/watch?v=jgiZlGzXMBw

10 Anexos

10.1 Anexo 1

Tabla de tiempos de ejecución

Programación dinámica

	Tiempos														
# Ejecucic	test01	test02	test03	test04	test05	test06	test07	test08	test09	test10	test11	test12	test13	test14	test15
1	0.004	0.005	0.002	0.002	0.003	0.009	0.008	0.009	0.015	0.016	0.029	0.031	0.003	0.07	0.159
2	0.002	0.005	0.002	0.003	0.002	0.003	0.006	0.017	0.014	0.015	0.029	0.028	0.002	0.07	0.16
3	0.006	0.006	0.009	0.003	0.003	0.003	0.009	0.012	0.017	0.019	0.028	0.03	0.002	0.072	0.163
4	0.003	0.008	0.004	0.003	0.003	0.009	0.007	0.015	0.014	0.015	0.028	0.03	0.003	0.071	0.161
5	0.004	0.003	0.002	0.003	0.007	0.004	0.009	0.017	0.014	0.02	0.029	0.032	0.008	0.07	0.162
6	0.007	0.004	0.001	0.002	0.007	0.004	0.01	0.011	0.014	0.015	0.03	0.029	0.003	0.073	0.166
7	0.005	0.002	0.003	0.002	0.004	0.01	0.009	0.01	0.018	0.019	0.029	0.029	0.002	0.073	0.173
8	0.005	0.014	0.012	0.006	0.011	0.008	0.007	0.019	0.017	0.021	0.031	0.03	0.01	0.072	0.168
9	0.006	0.008	0.011	0.015	0.003	0.004	0.009	0.012	0.024	0.021	0.029	0.032	0.011	0.073	0.165
10	0.01	0.003	0.14	0.004	0.004	0.005	0.011	0.013	0.017	0.019	0.032	0.031	0.009	0.075	0.168
Promedio	0.0052	0.0058	0.0186	0.0043	0.0047	0.0059	0.0085	0.0135	0.0164	0.018	0.0294	0.0302	0.0053	0.0719	0.1645
Op basica	28	506	157	3348	5755	61550	474713	844734	1383868	1796319	3693577	3751352	33076	9705035	22450250