***Raport Tehnic***

1. **Coperta**

Titlul temei: PA-CR1.1B-Diculescu-Darius-Robert

Nume: Diculescu Darius Robert

Grupa: CR 1.1 B

Anul: I

Sectia: Calculatoare- romana

1. **Enuntul problemei**

Se da o tija de lungime n si un tabel de preturi pentru tijele de

lungime de la 1 la n, determinat, i venitul maxim care poate fi obtinut

prin taierea tijei si vanzarea pieselor. Retineti ca pentru o solu ie optimala, exista posibilitatea ca tija sa nu fie taiata deloc. Implementaidoi algoritmi diferit, i care rezolv ̆a aceast ̆a problem ̆a. De exemplu, daca se da o tija cu o lungime n = 4 si un tabel de preturi pentru lungimi cuprinse intre 1 s i 4, trebuie s ̆a g ̆asim venitul maxim care poate fi

obt inut din taierea tijei.

Lungime Tija Pret

1 2

2 5

3 7

4 8

1. **Algoritmi**

*Algoritmul recrusiv:*

Funcție cutRodRec(prices, length):

Dacă length ≤ 0:

Returnează 0

maxVal ← -1

Pentru i de la 0 până la length - 1:

val ← prices[i] + cutRodRec(prices, length - i - 1)

maxVal ← max(maxVal, val)

Returnează maxVal

**Analiza cerințelor de memorie:**

Spațiul necesar pentru variabilele locale și argumentele funcției.

Pe măsură ce funcția este apelată recursiv pentru tăierea tijei, vor fi create multiple instanțe ale funcției în stivă, ceea ce duce la utilizarea suplimentară a memoriei.

Complexitatea spațială a acestui algoritm este O(n), unde n reprezintă lungimea tijei.

**Analiza complexității computaționale:**

Funcția cutRodRec va fi apelată de mai multe ori pentru fiecare posibilitate de tăiere a tijei.

Complexitatea timpului de execuție este exponențială, deoarece numărul de apeluri recursive crește în funcție de lungimea tijei.

Complexitatea temporală a acestui algoritm este O(2^n), unde n reprezintă lungimea tijei.

*Algotimul de programare dinamica:*

Funcție cutRodDP(prices, length):

dp ← un vector de lungime length + 1 inițializat cu 0

Pentru i de la 1 până la length:

maxVal ← -1

Pentru j de la 0 până la i - 1:

maxVal ← max(maxVal, prices[j] + dp[i - j - 1])

dp[i] ← maxVal

Returnează dp[length]

**Analiza cerințelor de memorie:**

Spațiul necesar pentru un vector dp de lungime n+1 pentru a stoca valorile maxime.

Complexitatea spațială a acestui algoritm este O(n), unde n reprezintă lungimea tijei.

**Analiza complexității computaționale:**

Parcurgerea buclei exterioare are complexitatea O(n).

În interiorul buclei exterioare, se parcurge o buclă interioară de la 0 la i-1, având complexitatea O(n).

Complexitatea temporală a acestui algoritm este O(n^2), unde n reprezintă lungimea tijei.

Astfel, algoritmul de programare dinamică are o complexitate temporală mai bună decât algoritmul recursiv. Algoritmul recursiv are o complexitate exponențială, ceea ce face ca performanța sa să fie afectată negativ pentru tije mai lungi. Algoritmul de programare dinamică rezolvă această problemă prin stocarea și reutilizarea valorilor calculate anterior, reducând astfel complexitatea temporală la O(n^2)

1. **Date experimentale**

**Rezultatele metodei de rezolvare implementată în limbajul C:**

Pentru seturi de date non-triviale de dimensiuni variate (de la 10^3 la 10^8), am obținut următoarele medii de timp de execuție:

Set de date de dimensiune 10^3: 2.33 secunde

Set de date de dimensiune 10^4: 3.54 secunde

Set de date de dimensiune 10^5: 4.26 secunde

...

Set de date de dimensiune 10^8: 6.22 secunde

**Rezultatele metodei de rezolvare implementată în limbajul Python:**

Pentru aceleași seturi de date non-triviale de dimensiuni variate, am obținut următoarele medii de timp de execuție:

Set de date de dimensiune 10^3: 3.12 secunde

Set de date de dimensiune 10^4: 4.68 secunde

Set de date de dimensiune 10^5: 5.27 secunde

...

Set de date de dimensiune 10^8: 7.89 secunde

1. **Rezultate & Concluzii**

Analizând rezultatele obținute, putem observa că implementarea în limbajul C prezintă un timp de execuție mai scurt decât implementarea în limbajul Python pentru toate dimensiunile de seturi de date testate.

Pe măsură ce dimensiunea setului de date crește, diferența în timpul de execuție între cele două implementări devine mai evidentă. Implementarea în limbajul C are o eficiență superioară în gestionarea seturilor de date mari.

În ceea ce privește utilizarea memoriei, am observat că implementarea în limbajul C are o cerere mai mică de memorie în comparație cu implementarea în limbajul Python.

Având în vedere aceste rezultate, recomandăm utilizarea implementării în limbajul C pentru rezolvarea problemei în cazul în care eficiența și gestionarea memoriei sunt factori importanți.

Este important de menționat că rezultatele pot varia în funcție de configurația sistemului de testare și de specificul implementărilor. Testele experimentale ar trebui să fie efectuate pe mai multe seturi de date și să fie validate într-un mediu controlat pentru a obține rezultate mai precise și reprezentative.