**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

**Modulio P170B400 „Algoritmų sudarymas ir analizė“**

Laboratorinio darbo ataskaita

**Trečias laboratorinis darbas**

|  |  |
| --- | --- |
|  | **Dėstytojas**  Lekt. Makackas Dalius |
|  |
| **Studentas**  Rokas Puzonas |

**KAUNAS, 2023**

**Turinys**

[1. Pirmo programinio kodo analizė 3](#_Toc132670814)

[1.1. Sąlyga ir analizė 3](#_Toc132670815)

[1.2. Eksperimentas 3](#_Toc132670816)

[2. Antro programinio kodo analizė 5](#_Toc132670817)

[2.1. Sąlyga ir analizė 5](#_Toc132670818)

[2.2. Eksperimentas 5](#_Toc132670819)

[3. Rekursinės užduoties sprendimas 7](#_Toc132670820)

[3.1. Sąlyga 7](#_Toc132670821)

[3.2. Rekurentinio sprendimas ir analizė 7](#_Toc132670822)

[3.3. Dinaminio sprendimas ir analizė 8](#_Toc132670823)

[3.4. Rekurentinio ir dinamio eksperimentai 9](#_Toc132670824)

# Pirma užduotis

* Pateikite rekursinį uždavinio sprendimo algoritmą (rekursinis sąryšis su paaiškinimais), bei realizuokite programinį kodą sprendžiantį nurodytą uždavinį (rekursinis sprendimas netaikant dinaminio programavimo).
* Pritaikykite dinaminio programavimo metodologiją pateiktam uždaviniui (pateikti paaiškinimą), bei realizuokite programinį kodą sprendžiantį nurodytą uždavinį (taikant dinaminį programavimą).
* Atlikite realizuotų programinių kodų analizę ir apskaičiuokite įverčius „iš viršaus“ ir „iš apačios“. Atlikite našumo analizę (skaičiuojant programos vykdymo laiką arba veiksmų skaičių) ir patikrinkite, ar apskaičiuotas metodo asimptotinis sudėtingumas atitinka eksperimentinius rezultatus.

## Sąlyga

Sandėlyje laikoma n pavadinimų produktų, kurių svoriai wi ir vertės qi žinomos, i = 1,n. Į lėktuvą galima pakrauti W svorio vienetų. Kokia maksimali galimų pakrauti daiktų vertė? Produktų kiekis gali būti aprašomas tik natūraliaisiais skaičiais.

## Rekursyvus sprendimas

|  |  |  |
| --- | --- | --- |
|  | Kiekis | Laikas |
| public static int knapsack(int weightLimit, int n, int[] weights, int[] values) |  |  |
| { |  |  |
| if (n == 1) | 1 | c1 |
| { |  |  |
| return weights[0] <= weightLimit ? values[0] : 0; | 1 | c2 |
| } |  |  |
| if (weights[n - 1] > weightLimit) | 1 | c3 |
| { |  |  |
| return knapsack(weightLimit, n - 1, weights, values); | 1 | c4 |
| } |  |  |
|  |  |  |
| int case1 = knapsack(weightLimit - weights[n - 1], n - 1, weights, values) + values[n - 1]; | 1 | T(n-1) |
| int case2 = knapsack(weightLimit, n - 1, weights, values); | 1 | T(n-1) |
| return Math.Max(case1, case2); | 1 | c5 |
| } |  |  |

Blogiausiu atvėju: O(n) = 2n

Geriausiu atvėju: Ω(n) = 1

## Dinaminis sprendimas

|  |  |  |
| --- | --- | --- |
|  | Kiekis | Laikas |
| public static int knapsackDP(int weightLimit, int n, int[] weights, int[] values) |  |  |
| { |  |  |
| int[,] memo = new int[n + 1, weightLimit + 1]; | 1 | c1 |
| for (int i = 1; i <= n; i++) | n+1 | c3 |
| { |  |  |
| for (int w = 1; w <= weightLimit; w++) | (W+1)n | c4 |
| { |  |  |
| if (weights[i-1] <= w) | Wn | c5 |
| { |  |  |
| memo[i, w] = Math.Max(values[i - 1] + memo[i - 1, w - weights[i - 1]], memo[i - 1, w]); | Wk | c6 |
| } |  |  |
| else |  |  |
| { |  |  |
| memo[i, w] = memo[i - 1, w]; | W(n-k) | c7 |
| } |  |  |
| } |  |  |
| } |  |  |
|  |  |  |
| return memo[n, weightLimit]; | 1 | c8 |
| } |  |  |

Blogiausiu atvėju: O(n) = Wn

Geriausiu atvėju: Ω(n) = Wn

## Eksperimentas

# Antra užduotis

* Atlikite pateiktų procedūrų lygiagretinimą.
* Įvertinkite teorinį nelygiagretintų ir lygiagretintų procedūrų sudėtingumą.
* Atlikite realizuotų programinių kodų analizę ir apskaičiuokite įverčius „iš viršaus“ ir „iš apačios“. Atlikite našumo analizę (skaičiuojant programos vykdymo laiką arba veiksmų skaičių) ir patikrinkite, ar apskaičiuotas metodo asimptotinis sudėtingumas atitinka eksperimentinius rezultatus.

## Sąlyga ir analizė paprasto

|  |  |  |
| --- | --- | --- |
| public static long methodToAnalysis1(int[] arr) | Kiekis | Laikas |
| { |  |  |
| long n = arr.Length; |  |  |
| long k = n; |  |  |
|  |  |  |
| for (int i = 0; i < n; i++) | n+1 | c1 |
| { |  |  |
| if (arr[i] / 7 == 0) | n | c2 |
| { |  |  |
| k -= 2; | k1 | c3 |
| } |  |  |
| else |  |  |
| { |  |  |
| k += 3; | n-k1 | c4 |
| } |  |  |
|  |  |  |
| } |  |  |
|  |  |  |
| if (arr[0] > 0) | 1 | c5 |
| { |  |  |
| for (int i = 0; i < n \* n; i++) | k2\*(n2+1) | c6 |
| { |  |  |
| if (arr[0] > 0) |  |  |
| { |  |  |
| k += 3; | k2\*n2 | C7 |
| } |  |  |
| } |  |  |
| } |  |  |
|  |  |  |
| return k; | 1 | c8 |
| } |  |  |

Blogiausiu atvėju: O(n) = n2

Geriausiu atvėju: Ω(n) = n

## Paralelizuoto analizė

## Eksperimentas

# Rekursinės užduoties sprendimas

## Sąlyga

Ant Žaidimo lentos (1 x n) langelių surašyti atsitiktiniai teigiami skaičiai (taškai). Žaidėjas vieno

ėjimo metu gali pasirinkti - pereiti į kitą langelį (l -> 2) ir gauti tiek taškų, kiek yra tame langelyje,

ar peršokti vieną langelį (l -> 3) ir gauti du kartus daugiau taškų, nei absoliutinis peršokto ir naujo

langelio taškų skirtumas. Žaidėjas pradeda nuo starto linijos, t. y. jis pirmu ėjimu gali pereiti į

pirmą arba antrą langelį. Kokia yra didžiausia taškų suma, kurią žaidėjas gali surinkti pasiekęs

paskutinį langelį?

## Rekurentinio sprendimas ir analizė

|  |  |  |
| --- | --- | --- |
|  | Kiekis | Laikas |
| static uint Recursive(Span<uint> board) |  |  |
| { |  |  |
| if (board.Length <= 1) | 1 | c1 |
| return 0; | 1 | c2 |
| else if (board.Length == 2) | 1 | c3 |
| return board[1]; | 1 | c4 |
|  |  |  |
| uint score1 = board[1] + Recursive(board[1..]); | 1 | c5+T(n-1) |
| uint score2 = 2 \* (uint)Math.Abs((int)board[0] - (int)board[2]) + Recursive(board[2..]); | 1 | c6+T(n-2) |
|  |  |  |
| return Math.Max(score1, score2); | 1 | c7 |
| } |  |  |

## Dinaminio sprendimas ir analizė

|  |  |  |
| --- | --- | --- |
| static uint DP(uint[] board) | Kiekis | Laikas |
| { |  |  |
| uint[] cache = new uint[board.Length]; | 1 | c1 |
| cache[0] = 0; | 1 | c2 |
| cache[1] = board[1]; | 1 | c3 |
|  |  |  |
| for (int i = 2; i < board.Length; i++) | n-1 | c4 |
| { |  |  |
| uint score1 = board[i] + cache[i - 1]; | n-2 | c5 |
| uint score2 = 2 \* (uint)Math.Abs((int)board[i] - (int)board[i - 2]) + cache[i - 2]; | n-2 | c6 |
|  |  |  |
| cache[i] = Math.Max(score1, score2); | n-2 | c7 |
| } |  |  |
|  |  |  |
| return cache[board.Length - 1]; | 1 | c8 |
| } |  |  |

## Rekurentinio ir dinamio eksperimentai