**KAUNO TECHNOLOGIJOS UNIVERSITETAS INFORMATIKOS FAKULTETAS**

Algoritmų sudarymas ir analizė Laboratorinis darbas Nr. 3

### Atliko:

### Rokas Marcinkevičius IFF 2-10

### Priėmė:

### Doc. Dalius Makackas

**KAUNAS 2024**

**TURINYS**

[1. UŽDUOTIS 3](#_Toc166070867)

[Rekursinis sprendimas: 3](#_Toc166070868)

[Dinaminis sprendimas: 4](#_Toc166070869)

[2. UŽDUOTIS: 6](#_Toc166070870)

[Rekursinis sprendimas: 6](#_Toc166070871)

[Dinaminis sprendimas 7](#_Toc166070872)

# 1. UŽDUOTIS

## Rekursinis sprendimas:

|  |  |  |
| --- | --- | --- |
| **Kodo eilutė** | **Kaina** | **Kiekis** |
| static int MaxPoints(int x, int y) |  |  |
| { |  |  |
| int rows = board.GetLength(0); | C1 | 1 |
| int cols = board.GetLength(1); | C2 | 1 |
|  |  |  |
| if (x == rows - 1 && y == cols - 1) | C3 | 1 |
| return board[x, y]; | C4 | 1 |
| int right = (y < cols - 1) ? MaxPoints(x, y + 1) : 0; | MaxPoints(x, y + 1) | T(n, m + 1) |
| int up = (x < rows - 1) ? MaxPoints(x + 1, y) : 0; | MaxPoints(x + 1, y) | T(n + 1, m) |
| int diagonal = (x < rows - 1 && y < cols - 1) ? MaxPoints(x + 1, y + 1) : 0; | MaxPoints(x + 1, y + 1) | T(n + 1, m + 1) |
| return board[x, y] + Math.Max(Math.Max(right, up), diagonal); | C5 | 1 |
| } |  |  |

Rekurentinė lygtis:

Našumo testas:

Rekursinis sąryšis:

Kiekviename lentos langelyje (x, y) atliekame tris rekursinius iškvietimus, nebent esame apatiniame dešiniajame kampe. Kadangi iš viso yra m x n langelių ir kiekviename langelyje atliekame pastovų darbo kiekį, laiko sudėtingumą galima išreikšti atliktų rekursinių iškvietimų skaičiumi.

## Dinaminis sprendimas:

Sprendimo kodas:

static int MaxPointsDynamic()

{

int rows = board.GetLength(0);

int cols = board.GetLength(1);

int[,] dp = new int[rows, cols];

for (int i = rows - 1; i >= 0; i--)

{

for (int j = cols - 1; j >= 0; j--)

{

if (i == rows - 1 && j == cols - 1)

dp[i, j] = board[i, j];

else

{

int right = (j < cols - 1) ? dp[i, j + 1] : 0;

int up = (i < rows - 1) ? dp[i + 1, j] : 0;

int diagonal = (i < rows - 1 && j < cols - 1) ?

dp[i + 1, j + 1] : 0;

dp[i, j] = board[i, j] + Math.Max(Math.Max(right, up), diagonal);

}

}

}

return dp[0, 0];

}

* Inicijuojame 2D masyvą dp, kuriame saugomi maksimalūs taškai, kuriuos galima pasiekti pradedant nuo kiekvieno langelio. - Pradedame pildyti dp masyvą nuo apatinio dešiniojo kampo iki viršutinio kairiojo kampo.
* Kiekvienam langeliui (i, j) apskaičiuojame didžiausius galimus taškus, atsižvelgdami į tris galimus ėjimus: į dešinę, aukštyn ir įstrižai į dešinę.
* Didžiausias taškų, pasiekiamų iš langelio (i, j), skaičius yra to langelio taškų ir taškų, pasiekiamų iš gretimų langelių į dešinę, į viršų ir įstrižai į dešinę, suma.
* Galiausiai didžiausias taškų, pasiekiamų iš apatinio kairiojo kampo, skaičius, saugomas dp[0, 0], yra uždavinio sprendinys.

Šio dinaminio programavimo sprendimo laiko sudėtingumas yra O(m × n), kur m reiškia eilučių skaičių, o n - stulpelių skaičių lentoje. Taip yra todėl, kad kiekvieną m x n lentos langelį, norėdami užpildyti dp masyvą, iteruojame lygiai vieną kartą. Erdvės sudėtingumas taip pat yra O(m × n), nes naudojame papildomą m × n masyvą, kad išsaugotume maksimalius taškus, kuriuos galima gauti iš kiekvieno langelio.

Našumo testas:

# 2. UŽDUOTIS:

## Rekursinis sprendimas:

|  |  |  |
| --- | --- | --- |
| Kodo eilutė | Kaina | Kiekis |
| static int DivideIntoBooksRecursive(int[] pagesPerChapter, int totalChapters, int numberOfBooks, int[,] dp) |  |  |
| { |  |  |
| if (totalChapters == 0) | C1 | 1 |
| return 0; | C2 | 1 |
| if (numberOfBooks == 0) | C3 | 1 |
| return int.MaxValue; | C4 | 1 |
|  |  |  |
| if (dp[totalChapters, numberOfBooks] != int.MaxValue) | C5 | 1 |
| return dp[totalChapters, numberOfBooks]; | C6 | 1 |
|  |  |  |
| int sum = 0; | C7 | 1 |
| int minMax = int.MaxValue; | C8 | 1 |
| for (int i = totalChapters; i >= 1; i--) | totalChapters + 1 | n + 1 |
| { |  |  |
| sum += pagesPerChapter[i - 1]; | C9 | 1 |
| minMax = Math.Min(minMax, Math.Max(DivideIntoBooksRecursive(pagesPerChapter, i - 1, numberOfBooks - 1, dp), sum)); | DivideIntoBooksRecursive(  pagesPerChapter, i - 1,  numberOfBooks - 1, dp), sum) | T(n – 1, m – 1) |
| } |  |  |
|  |  |  |
| dp[totalChapters, numberOfBooks] = minMax; | C10 | 1 |
| return minMax; | C11 | 1 |
| } |  |  |

Našumo testas:

Paaiškinimas ir įrodymas:

Funkcija `DivideIntoBooks Recursive` yra rekursyvi funkcija, kuri apskaičiuoja mažiausią didžiausią puslapių skaičių j knygoms su i skyriais. Ji kerta kiekvieną galimą padalijimo tašką, kad rastų mažiausią didžiausią puslapių skaičių iš visų galimų padalijimų. Dinaminio programavimo lentelė dp naudojama paproblemų rezultatams saugoti, kad būtų išvengta perteklinių skaičiavimų. Funkcija `DivideIntoBooks` inicializuoja dinaminio programavimo lentelę, iškviečia rekursyviąją funkciją ir pagal apskaičiuotus rezultatus atkuria puslapių pasiskirstymą knygose.

Asimptotinis sudėtingumas:

* Tegul ʼn yra skyrių skaičius, o m - knygų skaičius.
* Rekursinės funkcijos `DivideIntoBooksRecursive` laiko sudėtingumas be dinaminio programavimo yra O(m2), nes ji ištiria visus įmanomus skyrių skirstymo į knygas derinius.
* Naudojant dinaminį programavimą, laiko sudėtingumas sumažėja iki O(n2 × m), nes turime dvimatę dinaminio programavimo lentelę su n eilučių ir m stulpelių, o kiekvienam įrašui reikia išnagrinėti O(n) galimybių.
* Dinaminio programavimo lentelės erdvinis sudėtingumas yra O(n × m).

## Dinaminis sprendimas

Sprendimo kodas:

static int FindMinimumPages(int[] pagesPerChapter, int numberOfBooks)

{

int left = 0;

int right = 0;

foreach (int pages in pagesPerChapter)

{

right += pages;

}

while (left < right)

{

int mid = left + (right - left) / 2;

if (IsValid(pagesPerChapter, numberOfBooks, mid))

{

right = mid;

}

else

{

left = mid + 1;

}

}

return left;

}

static bool IsValid(int[] pagesPerChapter, int numberOfBooks, int maxPages)

{

int booksCount = 1;

int pagesInCurrentBook = 0;

foreach (int pages in pagesPerChapter)

{

if (pagesInCurrentBook + pages <= maxPages)

{

pagesInCurrentBook += pages;

}

else

{

booksCount++;

pagesInCurrentBook = pages;

if (booksCount > numberOfBooks)

{

return false;

}

}

}

return true;

}

static int[] DivideIntoBooks(int[] pagesPerChapter, int numberOfBooks, int minPages)

{

int[] booksPages = new int[numberOfBooks];

int currentBookIndex = 0;

int pagesInCurrentBook = 0;

foreach (int pages in pagesPerChapter)

{

if (pagesInCurrentBook + pages <= minPages)

{

pagesInCurrentBook += pages;

}

else

{

booksPages[currentBookIndex++] = pagesInCurrentBook;

pagesInCurrentBook = pages;

}

}

booksPages[currentBookIndex] = pagesInCurrentBook;

return booksPages;

}

Našumo testas:

Asimptotinis sudėtingumas:

* Tegul n yra skyrių skaičius, o m - knygų skaičius. - Šio dinaminio programavimo sprendimo laiko sudėtingumas yra O(n2 × m).
* Yra du įterpti ciklai, kurių kiekvienas iteruoja iki n.
* Vidiniame cikle yra dar vienas ciklas, iteruojantis iki m.
* Dinaminio programavimo lentelės dp erdvinis sudėtingumas yra O(n × m).