KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

# P170B400 Algoritmų sudarymas ir analizė

(P170B400)

Laboratorinių darbų ataskaita

Atliko:

IFF-1/9 gr. studentas

**Nedas Liaudanskis**

2023 m. kovo 22 d.

Priėmė:

**Doc. Pilkauskas Vytautas**

**Lekt. Kraujalis Tadas**

**Doc. Čalnerytė Dalia**

**Lekt. Makackas Dalius**

KAUNAS 2023

# UŽDUOTIS

## 1 Dalis:

Pateiktiems programinio kodo metodams **„methodToAnalysis(...)“** (gautiems atlikus užduoties pasirinkimo testą):

* Atlikite programinio kodo analizę, bei sudarykite rekurentinę lygį. Jei metodas neturi vidinių rekursinių kreipinių, apskaičiuokite pateikto metodo asimptotinį sudėtingumą. Jei metodo sudėtingumas priklauso nuo duomenų pateikiamų per parametrus – apskaičiuokite įverčius „iš viršaus“ ir „iš apačios“ (2 balai).
* Metodams, kurie turi rekurentinių kreipinių išspręskite rekurentinę lygtį apskaičiuodami jos asimptotinį sudėtingumą (1 balas).
* Atlikti eksperimentinį tyrimą (našumo testus: vykdymo laiką ir veiksmų skaičių) ir patikrinkite ar apskaičiuotas metodo asimptotinis sudėtingumas atitinka eksperimentinius rezultatus. Jei pateikto metodo asimptotinis sudėtingumas priklauso nuo duomenų, atitinkamai atliekant analizę reikia parinkti tokias testavimo duomenų imtis, kad rezultatai atspindėtų įvertinimus iš viršaus ir iš apačios (1 balas).

**Individualaus užduoties varianto lygčių kodas:**

**Pirmas metodas:**

public static long methodToAnalysis (int[] arr)

        {

            long n = arr.Length;

            long k = n;

            for (int i = 0; i < n; i++)

            {

                if (arr[i] > 0)

                {

                    for (int j = 0; j < n \* n / 2; j++)

                    {

                        k -= 2;

                    }

                    for (int j = 0; j < n \* n / 2; j++)

                    {

                        k += 3;

                    }

                }

            }

            return k;

        }

**Antras metodas:**

 public static long methodToAnalysis (int n, int[] arr)

        {

            long k = 0;

            for (int i = 0; i < n; i++)

            {

                k += k;

                k += FF6(i\*2, arr);

            }

            k += FF6(n, arr);

            return k;

        }

        public static long FF6(int n, int[] arr)

        {

            if (n > 0 && arr.Length > n)

            {

                return FF6(n-1, arr) ;

            }

            return n;

        }

## 2 Dalis:

* 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.

# Pirma Dalis. Metodų „methodToAnalysis(...)“ išanalizavimas

## Metodas:

public static long methodToAnalysis (int[] arr)

        {

            long n = arr.Length;

            long k = n;

            for (int i = 0; i < n; i++)

            {

                if (arr[i] > 0)

                {

                    for (int j = 0; j < n \* n / 2; j++)

                    {

                        k -= 2;

                    }

                    for (int j = 0; j < n \* n / 2; j++)

                    {

                        k += 3;

                    }

                }

            }

            return k;

        }

## Programinio kodo analizė

static long methodToAnalysis(int[] arr)

{

long n = arr.Length; // c1 | 1

long k = n; // c2 | 1

for (int i = 0; i < n; i++) // c3 | 1

{ // c4 | n + 1

if (arr[i] > 0) // c5 | n

{

for (int j = 0; j < n \* n / 2; j++) // c6 | n

{ // c7 | (n \* n / 2 + 1) \* n ((n^2 / 2 + 1) \* n) -> (n^3/2 + n)

k -= 2; // c8 | n \* n \* n / 2

}

for (int j = 0; j < n \* n / 2; j++) // c9 | n

{ // c10 | n \* n \* n / 2 + 1 -> (n^3/2 + n)

k += 3; // c11 | n \* n \* n / 2

}

}

}

return k; // c12 | 1

}

## Asimptotinio sudėtingumo apskaičiavimas

T(n) = C1 + C2 + C3 + C4(n + 1) + C5\*n + C6\*n\*x + C7(n3/ 2 + n)\*x + C8(n3/2)\*x + C9\*n\*x + C10(n3/ 2 + n )\*x + C11(n3/2)\*x + C12 =>

* x(C6\*n + C7(n3/ 2 + n) + C8(n3/2) + C9\*n + C10(n3/ 2 + n ) + C11(n3/2)) + C5\*n + C4(n + 1) + C1 + C2 + C3 + C12

Kai if (arr[i] > 0) sąlyga nėra įgyvendinama x = 0, o x = 1 kai sąlyga yra teigiama.

Kai x = 0, bus geriausias atvėjis:

* C5\*n + C4(n + 1) + C1 + C2 + C3 + C12
* b(n + 1) + c (Ω(n + 1) iš apačios)

b, c – konsatantos.

Kai x = 1, bus blogiausias atvėjis:

* (C6\*n + C7(n3/ 2 + n) + C8(n3/2) + C9\*n + C10(n3/ 2 + n) + C11(n3/2) + C5\*n + C4(n + 1) + C1 + C2 + C3 + C12
* an3/2+ bn + c (O(n3/2) iš viršaus)

a, b, c – konsatantos.

## Eksperimentinis tyrimas

## 

## Metodai

 public static long methodToAnalysis (int n, int[] arr)

        {

            long k = 0;

            for (int i = 0; i < n; i++)

            {

                k += k;

                k += FF6(i\*2, arr);

            }

            k += FF6(n, arr);

            return k;

        }

        public static long FF6(int n, int[] arr)

        {

            if (n > 0 && arr.Length > n)

            {

                return FF6(n-1, arr) ;

            }

            return n;

        }

## Programinio kodo analizė

static long methodToAnalysis2(int n, int[] arr)

{

long k = 0; // c1 | 1

for (int i = 0; i < n; i++) // c2 | 1

{ // c3 | n + 1

k += k; // c4 | n

k += FF6(i \* 2, arr); // sum(i = 0; n = n; T(i\*2) ) | n

}

k += FF6(n, arr); // T(n) | 1

return k; // c5 | 1

}

static long FF6(int n, int[] arr)

{

if (n > 0 && arr.Length > n) // c6 | 1

{

return FF6(n - 1, arr); // T(n - 1) | 1

}

return n; // c7 | 1

}

## Rekurentinės lygties sudarymas ir asimptotinio sudėtingumo radimas

Pirmiausia reikia apsiskaičiuoti FF6 metodo sudėtingumą:

static long FF6(int n, int[] arr)

{

if (n > 0 && arr.Length > n) // c6 | 1

{

return FF6(n - 1, arr); // T(n - 1) | 1

}

return n; // c7 | 1

}

c6 + c7, kai neatitinka bent vienos sąlygos

F(n) =

C6 + T(n - 1) + c7, kai atitinka abi sąlygas

Reikia išspręsti lygtį F(n) = C6 + F(n - 1) + c7

Paveikslėlis, kuriame yra diagrama

Automatiškai sugeneruotas aprašymas

Iš lygties gauname jog F(n) = O(n)

Toliau įsistačius F(n) sudėtingumo reikšmę galime surasti viso metodo sudėtingumą:

static long methodToAnalysis2(int n, int[] arr)

{

long k = 0; // c1 | 1

for (int i = 0; i < n; i++) // c2 | 1

{ // c3 | n + 1

k += k; // c4 | n

k += FF6(i \* 2, arr); // sum(i = 0; n = n; T(i\*2) ) | n \* n

}

k += FF6(n, arr); // T(n) | 1

return k; // c5 | 1

}

T(n) = C1 + c2 + c3 + c4\*n + c5 + T(n) + \* n

Kadangi T(n) = O(n), tai galime įstatyti n vietoj T(n)

T(n) = C1 + c2 + c3 + c4\*n + c5 + n + n\*n

Iš lygties gauname:

T(n) = n^2 + n + c, tai galiu teigti, jog T(n) = O(n^2)

Šis sudėtingumas yra gaunamas kai yra žiūrima iš viršaus. Iš apačios sudėtingumas gaunasi tada kai n = 0, tada Ω(0), nes kai n = 0, ciklas nebus prasukamas.

## Eksperimentinis tyrimas

## 

# Antra Dalis:

## Užduoties sąlyga

Stačiakampė žaidimo lenta suskirstyta į n x m langų. Kiekviename lange nurodytas taškų skaičius (sveikas teigiamas skaičius). Reikia sudaryti maršrutą, kurio eidamas žaidėjas, pradėjęs žaidimą apatiniame kairiajame kampe, pasiektų viršutinį dešinį lentos kampą surikęs daugiausia taškų. Žaidėjas gali judėti tik į viršų, į dešinę arba įstrižai į viršutinį dešinį langelį. Jeigu žaidėjas pateko į langelį iš apačios arba kairės, jis gauna tiek taškų, kiek parašyta langelyje. Jeigu žaidėjas juda įstrižai, gauna langelyje esančių taškų skaičių, padaugintą iš dviejų. Žaidėjas negali judėti į apačią, kairę ar kitomis nei nurodyta įstrižomis kryptimis.

## Programos kodas

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Program

{

/// <summary>

/// Board class

/// </summary>

public class Board

{

List<int> Path = new List<int>();

List<int> PathR = new List<int>();

int n = 0;

int m = 0;

int[,] board;

int[,] pboard;

int MaxD = 0;

int MaxR = 0;

public int CountT = 0;

/// <summary>

/// A constructor that creates an empty nxm matrix, that holds all 0 and another matrix with random point values

/// </summary>

/// <param name="x"> size of the matrix in length</param>

/// <param name="y"> size of the matrix in hight </param>

public Board(int x, int y)

{

n = x;

m = y;

board = new int[n,m];

pboard = new int[n, m];

var rand = new Random();

for (int i = 0; i < n; i++)

{

for (int j = 0; j < m; j++)

{

board[i, j] = rand.Next(0, 9);

pboard[i, j] = 0;

}

}

}

/// <summary>

/// Calculates the answer dynamically

/// </summary>

public void CalculateD()

{

CalculateMax();

FindPath();

PrintBoardD();

}

/// <summary>

/// Calculates the answer c

/// </summary>

public void CalculateR()

{

PathR.Add(board[0, 0]);

MaxValue(board[0, 0], 0, 0, PathR);

PrintBoardR();

}

/// <summary>

/// A function used to print out the answer dynamically

/// </summary>

public void PrintBoardD()

{

Console.WriteLine("");

Console.WriteLine("\nStarting Board: ");

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

{

for (int j = 0; j < m; j++)

{

Console.Write(board[i, j] + " ");

}

Console.WriteLine();

}

Console.WriteLine("\nMax points on each block: ");

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

{

for (int j = 0; j < m; j++)

{

Console.Write(pboard[i, j] + " ");

}

Console.WriteLine();

}

Console.WriteLine("\nMax points: " + MaxD);

Console.WriteLine("Path :");

foreach(int i in Path)

{

Console.Write(i + " ");

}

}

/// <summary>

/// A function used to print out the answer recursively

/// </summary>

public void PrintBoardR()

{

Console.WriteLine("");

Console.WriteLine("\nStarting Board: ");

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

{

for (int j = 0; j < m; j++)

{

Console.Write(board[i, j] + " ");

}

Console.WriteLine();

}

Console.WriteLine("\nMax points: " + MaxR);

Console.WriteLine("\nPath :");

foreach (int i in Path)

{

Console.Write(i + " ");

}

}

/// <summary>

/// Finds all max points values in each block of the matrix board

/// </summary>

private void CalculateMax()

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < m; j++)

{

int maxPoints = board[i, j];

if (i > 0)

{

maxPoints = Math.Max(maxPoints, pboard[i - 1, j] + board[i, j]);

}

if (j > 0)

{

maxPoints = Math.Max(maxPoints, pboard[i, j - 1] + board[i, j]);

}

if (i > 0 && j > 0)

{

maxPoints = Math.Max(maxPoints, pboard[i - 1, j - 1] + board[i, j] \* 2);

}

pboard[i, j] = maxPoints;

}

}

MaxD = pboard[n - 1, m - 1];

}

/// <summary>

/// Finds the path that the player has to take in order to get the max amount of points

/// </summary>

private void FindPath()

{

int x = n - 1;

int y = m - 1;

Path.Add(board[x, y]);

while (x > 0 || y > 0)

{

if (x == 0)

{

y--;

}

else if (y == 0)

{

x--;

}

else if (pboard[x, y] - board[x, y] == pboard[x - 1, y])

{

x--;

}

else if (pboard[x, y] - board[x, y] == pboard[x, y - 1])

{

y--;

}

else if (pboard[x, y] - board[x, y] \* 2 == pboard[x - 1, y - 1])

{

x--;

y--;

}

Path.Add(board[x, y]);

}

Path.Reverse();

}

/// <summary>

/// Recursively finds the path that leads to the most points

/// </summary>

/// <param name="max"> point values that have accumulated up to this point </param>

/// <param name="x"> current x position </param>

/// <param name="y"> current y position </param>

/// <param name="path"> the path that gets generated up to the certain point in the recursion </param>

private void MaxValue(int max, int x, int y, List<int> path)

{

CountT++;

if (MaxR < max)

{

MaxR = max;

PathR = path;

}

if (n - 1 > x)

{

List<int> p = Fill(path);

p.Add(board[x + 1, y]);

MaxValue(max + board[x + 1, y], x + 1, y, p);

}

if (m - 1 > y)

{

List<int> p = Fill(path);

p.Add(board[x, y + 1]);

MaxValue(max + board[x, y + 1], x, y + 1, p);

}

if (n - 1 > x && m - 1 > y)

{

List<int> p = Fill(path);

p.Add(board[x + 1, y + 1]);

MaxValue(max + board[x + 1, y + 1] \* 2, x + 1, y + 1, p);

}

}

/// <summary>

/// Fills a list with one more position of the path that the player has to take

/// </summary>

/// <param name="list"> positions up to this point </param>

/// <returns> position list with the new position in the end of it </returns>

private List<int> Fill(List<int> list)

{

List<int> path = new List<int>();

foreach (int i in list)

{

path.Add(i);

}

return path;

}

}

class Program

{

static void Main(string[] args)

{

const int n = 12;

const int m = 12;

Board Dyn = new Board(n, m);

var watch = new System.Diagnostics.Stopwatch();

watch.Start();

Dyn.CalculateD();

watch.Stop();

Console.WriteLine($"\nDinaminio sprendimo skaičiavimų laikas: {watch.ElapsedMilliseconds} ms");

watch.Reset();

watch.Start();

Dyn.CalculateR();

watch.Stop();

Console.WriteLine($"\nRekurentinio sprendimo skaičiavimų laikas: {watch.ElapsedMilliseconds} ms");

Console.WriteLine($"\nRekurentinio sprendimo rekurentinių ciklų skaičius: {Dyn.CountT}");

}

## }

## Programos rezultatai

Starting Board:

5 2 5 2 3 0 2 2 0 5 3 5

2 3 3 6 0 5 4 7 7 0 0 8

8 4 4 6 7 6 2 2 5 6 4 4

2 2 1 6 8 3 6 0 1 8 5 2

0 0 8 0 7 1 2 8 8 8 1 8

5 4 1 2 4 1 7 6 7 7 7 8

8 5 7 5 2 1 1 8 7 4 0 7

8 8 7 5 3 4 2 1 3 0 0 3

8 2 5 7 1 8 1 4 8 8 0 8

6 1 4 5 0 5 5 1 4 0 2 0

1 0 4 7 5 3 3 4 3 3 7 2

3 5 2 5 6 3 0 1 6 2 2 6

Max points on each block:

56 58 72 85 89 97 103 110 115 125 128 141

51 56 67 83 86 97 101 108 115 116 120 136

49 53 64 77 86 92 94 96 101 116 120 124

41 45 60 71 79 82 88 88 95 110 115 117

39 43 59 59 69 70 72 86 94 102 103 115

39 43 49 55 61 62 70 77 85 92 99 107

34 39 48 53 55 56 57 71 78 82 82 89

26 34 41 46 49 53 55 56 64 69 69 80

18 20 25 36 37 47 48 52 61 69 69 77

10 11 20 29 29 39 44 45 49 49 54 54

4 8 16 24 29 32 35 39 42 45 52 54

3 8 10 15 21 24 24 25 31 33 35 41

Max points: 141

Path :

3 1 6 8 8 8 7 7 5 2 1 7 8 8 8 8 6 4 8 5

Dinaminio sprendimo skaičiavimų laikas: 43 ms

Starting Board:

5 2 5 2 3 0 2 2 0 5 3 5

2 3 3 6 0 5 4 7 7 0 0 8

8 4 4 6 7 6 2 2 5 6 4 4

2 2 1 6 8 3 6 0 1 8 5 2

0 0 8 0 7 1 2 8 8 8 1 8

5 4 1 2 4 1 7 6 7 7 7 8

8 5 7 5 2 1 1 8 7 4 0 7

8 8 7 5 3 4 2 1 3 0 0 3

8 2 5 7 1 8 1 4 8 8 0 8

6 1 4 5 0 5 5 1 4 0 2 0

1 0 4 7 5 3 3 4 3 3 7 2

3 5 2 5 6 3 0 1 6 2 2 6

Max points: 141

Path :

3 1 6 8 8 8 7 7 5 2 1 7 8 8 8 8 6 4 8 5

Rekurentinio sprendimo skaičiavimų laikas: 22483 ms

Rekurentinio sprendimo rekurentinių ciklų skaičius: 125797984

## Programos rezultatų greitaveika