KAUNO TECHNOLOGIJOS UNIVERSITETAS

FAKULTETAS

Algoritmų sudarymas ir analizė  
Laboratorinis darbas Nr. 2

Atliko:

Joris Gagilas

Priėmė:

Doc. Dalius Makackas

KAUNAS 2022

TURINYS

[1. Užduotis Nr. 1 3](#_Toc101857228)

[2. Algoritmai ir analizė 5](#_Toc101857229)

[2.1. Kodas Nr. 1 5](#_Toc101857230)

[2.2. Kodas Nr. 2 8](#_Toc101857231)

[3. Kodas 9](#_Toc101857232)

[4. Užduotis Nr. 2 11](#_Toc101857233)

[5. Algoritmai 12](#_Toc101857234)

[6. Rezultatai 13](#_Toc101857235)

[7. Kodas 14](#_Toc101857236)

# Užduotis Nr. 1

**1 užduoties 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).*

**Duoti kodai:**

* Graphical user interface, text, application

  Description automatically generated
* Graphical user interface, text, application, email

  Description automatically generated

# Algoritmai ir analizė

## Kodas Nr. 1

|  |  |  |
| --- | --- | --- |
| **Kodas** | **Kaina** | **Kiekis** |
| public static long methodToAnalysis1(int[] arr) |  |  |
|  |  |  |
| { |  |  |
| long n = arr.Length; | c1 | 1 |
| long k = n; | c2 | 1 |
| for (int i = 0; i < n \* n; i++) | c3 | n­2 |
| { |  |  |
| if (arr[0] > 0) | c4 | n2 |
| { |  |  |
| for (int j = 0; j < n; j++) | c5 | n2 \* (n \* γ); |
| { |  |  |
| k -= 2; | c6 | n3 \* γ; |
| } |  |  |
| for (int j = 0; j < n \* n / 2; j++) | c7 | n2 \* (n2/2 \* γ); |
| { |  |  |
| k += 3; | c8 | n4/2 \* γ; |
| } |  |  |
| } |  |  |
| } |  |  |
|  |  |  |
| return k; |  |  |
|  |  |  |
| } |  |  |

jei arr[0] > 0, tai γ=1, kitu atveju γ=0.

* Kai γ=0:
* Kai γ = 1:

## Kodas Nr. 2

|  |  |  |
| --- | --- | --- |
| **Kodas** | **Kaina** | **Kiekis** |
| public static long methodToAnalysis2(int n) |  |  |
|  |  |  |
| { |  |  |
|  |  |  |
| long k = 0; | c1 | 1 |
| int[] arr = new int[n]; | c2 | 1 |
| Random randNum = new Random(); | c3 | 1 |
| for (int i = 0; i < n; i++) | c4 | n |
| { |  |  |
| arr[i] = randNum.Next(0, n); | c5 | n |
| k += arr[i] + FF1(i); | F(n) | n |
| } |  |  |
| return k; |  |  |
|  |  |  |
| } |  |  |
|  |  |  |
| public static long FF1(int n) |  |  |
| { |  |  |
| if (n > 0) | c6 | 1 |
| { |  |  |
| return FF1(n - 1); | F(n-1) | 1 |
| } |  |  |
| return n; |  |  |
| } |  |  |

# Kodas

using System.Diagnostics;

namespace Pirma

{

class Program

{

static void Main(string[] args)

{

int size = 2000;

int n = 10;

long k;

Console.WriteLine("1 - arr[0] = 0");

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

{

Stopwatch stopwatch = new Stopwatch();

int[] A = new int[size];

A[0] = 0;

stopwatch.Start();

k = methodToAnalysis1(A);

stopwatch.Stop();

Console.WriteLine("{0} {1} {2}", size, stopwatch.ElapsedMilliseconds, k);

GC.Collect();

size += 2000;

}

size = 20;

Console.WriteLine("1 - arr[0] = 1");

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

{

Stopwatch stopwatch = new Stopwatch();

int[] A = new int[size];

A[0] = 1;

stopwatch.Start();

k = methodToAnalysis1(A);

stopwatch.Stop();

Console.WriteLine("{0} {1} {2}", size, stopwatch.ElapsedMilliseconds, k);

GC.Collect();

size += 20;

}

n = 8;

size = 2000;

Console.WriteLine("2");

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

{

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

k = methodToAnalysis2(size);

stopwatch.Stop();

Console.WriteLine("{0} {1} {2}", size, stopwatch.ElapsedMilliseconds, k);

GC.Collect();

size += 2000;

}

}

public static long methodToAnalysis1(int[] arr)

{

long n = arr.Length;

long k = n \* n;

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

{

if (arr[0] > 0)

{

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

{

k -= 2;

}

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

{

k += 3;

}

}

}

return k;

}

public static long methodToAnalysis2(int n)

{

long k = 0;

int[] arr = new int[n];

Random randNum = new Random();

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

{

arr[i] = randNum.Next(0, n);

k += arr[i] + FF1(i);

}

return k;

}

public static long FF1(int n)

{

if (n > 0)

{

return FF1(n - 1);

}

return n;

}

}

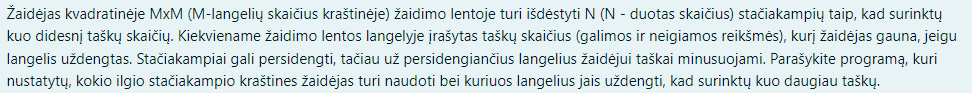
}

# Užduotis Nr. 2

**2 užduoties dalis:** Pateiktas žodinis uždavinys, kurio pagrindu:

* Realizuokite programą skirtingoms pateikto uždavinio duomenų imtims generuoti. Jei duomenų aibė priklauso nuo kelių parametrų – šie parametrai turi būti parametrizuojami. Pvz. jei turi būti sugeneruotas jungusis grafas kurio viršūnių skaičius *v*, o briaunų skaičius gali kisti nuo  *v* -1 (minimalaus jungusis grafas)  iki *v(v -1)/2* (pilnasis grafas) – grafo užpildymo koeficientas turi būti vienas iš duomenų generavimo parametrų. Sugeneruotos duomenų imtys turi būti saugomos atskiruose failuose, kurie toliau bus naudojami atliekant algoritmų testavimą.*(2 balai)*
* Realizuokite algoritmą optimaliam pateikto uždavinio sprendimui rasti. Laisva forma aprašykite algoritmo logiką ir pateikite kelis testinius atvejus parodant jog sudarytas algoritmas veikia teisingai. Atlikite algoritmo analizę ir pateikite apskaičiuotą sudaryto algoritmo sudėtingumą. Atlikite eksperimentinį tyrimą (skaičiuojant programos vykdymo laiką ir veiksmų skaičių) ir nustatykite, ar pateikto algoritmo sudėtingumas atitinka apskaičiuotą algoritmo sudėtingumą, bei pateikite duomenų kiekius prie kurių pavyksta rasti optimalius sprendinius per pasirinktą programos vykdymo laiką (pvz. 30 sek.).*(2 balai)*
* Sudarykite strategiją ir realizuokite algoritmą kaip galima geresniam pateikto uždavinio sprendimui rasti, kurio sudėtingumas turėtų polinominę priklausomybę nuo įvedamų duomenų kiekio, (šiuo atveju algoritmas turi rasti ne būtinai optimalų, bet kaip galima geresnį sprendinį). Atlikite eksperimentinį tyrimą (skaičiuojant programos vykdymo laiką ir veiksmų skaičių) ir nustatykite, ar pateikto algoritmo sudėtingumas atitinka apskaičiuotą algoritmo sudėtingumą. Palyginkite rezultatus su optimalų sprendinį randančiu algoritmu problematikos ir našumo prasme.*(2 balai)*

**Užduotis:**

****

# Algoritmai

**Algoritmas optimaliam sprendiniui:**

/// <summary>

///

/// </summary>

/// <param name="grid"></param>

/// <param name="n">height</param>

/// <param name="m">width</param>

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

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

/// <param name="M"></param>

/// <param name="Best"></param>

/// <returns></returns>

static Rectangle BestRect(GridPoint[,] grid, int n, int m, int y, int x, int M)

{

Rectangle Current = new Rectangle(x, y, m, n);

Current.CalculatePoints(grid);

Rectangle Best1 = new Rectangle();

Rectangle Best2 = new Rectangle();

if(n > 1)

{

Rectangle rect1 = BestRect(grid, n-1, m, y, x, M);

Rectangle rect2 = BestRect(grid, n-1, m, y+1, x, M);

rect1.CalculatePoints(grid);

rect2.CalculatePoints(grid);

if (rect1 > rect2)

{

Best1 = rect1;

}

else

{

Best1 = rect2;

}

}

if (m > 1)

{

Rectangle rect1 = BestRect(grid, n, m-1, y, x, M);

Rectangle rect2 = BestRect(grid, n, m-1, y, x+1, M);

rect1.CalculatePoints(grid);

rect2.CalculatePoints(grid);

if (rect1 > rect2)

{

Best2 = rect1;

}

else

{

Best2 = rect2;

}

}

if(Best1 > Best2)

{

if(Best1 > Current)

{

return Best1;

}

else

{

return Current;

}

}

else

{

if(Best2 > Current)

{

return Best2;

}

else

{

return Current;

}

}

}

}

**Algoritmas geresniam sprendiniui:**

static Rectangle BestRectangleFor(GridPoint[,] grid, int M)

{

Rectangle max = new Rectangle();

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

{

for (int j = M - i; j > 0; j--)

{

for (int k = 0; k < M; k++)

{

for(int l = M - k; l > 0; l--)

{

Rectangle rect = new Rectangle(k, i, l, j);

rect.CalculatePoints(grid);

if (rect > max) { max = rect; }

}

}

}

}

return max;

}

# Rezultatai

N-1; M-3:

Text

Description automatically generated

N-1; M-10:

A screen shot of a computer

Description automatically generated with low confidence

N-3; M-10:

A screen shot of a computer

Description automatically generated with low confidence

# Kodas

using System.Text;

using System.IO;

using System.Diagnostics;

namespace Antra

{

class Program

{

static void Main(string[] args)

{

int N = 1;

int M = 10;

string filename = "square.txt";

CreateSquareFile(filename, M);

GridPoint[,] grid = ReadSquareFile(filename, M);

GridPoint[,] grid2 = ReadSquareFile(filename, M);

for (int i = 0; i < grid.GetLength(0); i++)

{

for (int j = 0; j < grid.GetLength(1); j++)

{

Console.Write(new String(' ', 4-grid[i,j].ToString().Length));

Console.Write(grid[i, j]);

}

Console.WriteLine();

}

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

for (int o = 0; o < N; o++)

{

Rectangle best = BestRectangle(grid, M);

Console.WriteLine(best.ToString());

for(int i = best.y; i<best.y+best.height; i++)

{

for(int j = best.x; j<best.x+best.width; j++)

{

grid[i, j].AddOverlap();

}

}

}

stopwatch.Stop();

Console.WriteLine(stopwatch.ElapsedMilliseconds.ToString());

Stopwatch stopwatch2 = new Stopwatch();

stopwatch2.Start();

for (int o = 0; o < N; o++)

{

Rectangle best = BestRectangleFor(grid2, M);

Console.WriteLine(best.ToString());

for (int i = best.y; i < best.y + best.height; i++)

{

for (int j = best.x; j < best.x + best.width; j++)

{

grid2[i, j].AddOverlap();

}

}

}

stopwatch2.Stop();

Console.WriteLine(stopwatch2.ElapsedMilliseconds.ToString());

}

static void CreateSquareFile(string filename, int M)

{

int seed = M;

string[] lines = new string[M];

Random random = new Random(seed);

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

{

StringBuilder line = new StringBuilder();

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

{

if (j != 0)

{

line.Append(" ");

}

line.Append(random.Next(-10, 10));

}

lines[i] = line.ToString();

}

File.WriteAllLines(filename, lines);

}

static GridPoint[,] ReadSquareFile(string filename, int M)

{

GridPoint[,] result = new GridPoint[M, M];

string[] lines = File.ReadAllLines(filename);

int x = 0;

foreach (string line in lines)

{

string[] values = line.Split(' ', StringSplitOptions.RemoveEmptyEntries);

for(int i = 0; i < values.Length; i++)

{

result [x, i] = new GridPoint(int.Parse(values [i]));

}

x++;

}

return result;

}

static Rectangle BestRectangleFor(GridPoint[,] grid, int M)

{

Rectangle max = new Rectangle();

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

{

for (int j = M - i; j > 0; j--)

{

for (int k = 0; k < M; k++)

{

for(int l = M - k; l > 0; l--)

{

Rectangle rect = new Rectangle(k, i, l, j);

rect.CalculatePoints(grid);

if (rect > max) { max = rect; }

}

}

}

}

return max;

}

static Rectangle BestRectangle(GridPoint[,] grid, int M)

{

return BestRect(grid, M, M, 0, 0, M);

}

/// <summary>

///

/// </summary>

/// <param name="grid"></param>

/// <param name="n">height</param>

/// <param name="m">width</param>

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

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

/// <param name="M"></param>

/// <param name="Best"></param>

/// <returns></returns>

static Rectangle BestRect(GridPoint[,] grid, int n, int m, int y, int x, int M)

{

Rectangle Current = new Rectangle(x, y, m, n);

Current.CalculatePoints(grid);

Rectangle Best1 = new Rectangle();

Rectangle Best2 = new Rectangle();

if(n > 1)

{

Rectangle rect1 = BestRect(grid, n-1, m, y, x, M);

Rectangle rect2 = BestRect(grid, n-1, m, y+1, x, M);

rect1.CalculatePoints(grid);

rect2.CalculatePoints(grid);

if (rect1 > rect2)

{

Best1 = rect1;

}

else

{

Best1 = rect2;

}

}

if (m > 1)

{

Rectangle rect1 = BestRect(grid, n, m-1, y, x, M);

Rectangle rect2 = BestRect(grid, n, m-1, y, x+1, M);

rect1.CalculatePoints(grid);

rect2.CalculatePoints(grid);

if (rect1 > rect2)

{

Best2 = rect1;

}

else

{

Best2 = rect2;

}

}

if(Best1 > Best2)

{

if(Best1 > Current)

{

return Best1;

}

else

{

return Current;

}

}

else

{

if(Best2 > Current)

{

return Best2;

}

else

{

return Current;

}

}

}

}

class Rectangle

{

int points;

public int x, y;

public int width, height;

public Rectangle()

{

this.points = -1000000;

this.x = 0;

this.y = 0;

this.width = 0;

this.height = 0;

}

public Rectangle(int x, int y, int width, int height)

{

this.points = 0;

this.x = x;

this.y = y;

this.width = width;

this.height = height;

}

public void CalculatePoints(GridPoint [,] grid)

{

points = 0;

for (int i = y; i < y+height; i++)

{

for (int j = x; j < x+width; j++)

{

points += grid[i,j].Points();

}

}

}

public static bool operator > (Rectangle lhs, Rectangle rhs)

{

return lhs.points > rhs.points;

}

public static bool operator <(Rectangle lhs, Rectangle rhs)

{

return lhs.points <= rhs.points;

}

public override string ToString()

{

return String.Format("x:{0}; y:{1}; width:{2}; height:{3}; points{4}", x, y, width, height, points);

}

}

class GridPoint

{

int points;

int overlaps;

public GridPoint(int points)

{

this.points = points;

this.overlaps = 0;

}

public int Points()

{

return points;

}

public void AddOverlap()

{

overlaps++;

if (overlaps == 1)

{

points = Math.Abs(points) \* -1;

}

else if (overlaps > 1)

{

points = 0;

}

}

public override string ToString()

{

return points.ToString();

}

}

}