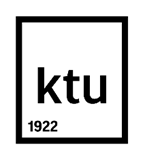
****

**Kauno technologijos universitetas**

Informatikos fakultetas

**Pirmasis laboratorinis darbas**

P170B400 Algoritmų sudarymas ir analizė

|  |  |
| --- | --- |
| **Rokas Marcinkevičius**  Studentas |  |
|  |  |
| **dr. MAKACKAS Dalius**  Dėstytojas |  |
|  |  |



**Turinys**

[**Užduotis 3**](#_heading=h.gjdgxs)

[**1.**](#_heading=h.30j0zll) **Realizacija – programos kodas 4**

[**2.**](#_heading=h.1fob9te) **Programos rezultatai 7**

[**3.**](#_heading=h.3znysh7) **Darbo laiko ir veiksmų skaičiaus priklausomybės – grafikai 8**

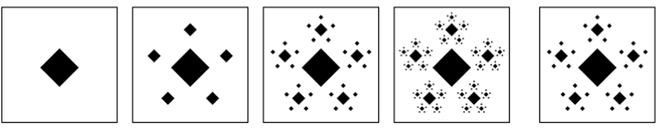
[**Išvados 9**](#_heading=h.2et92p0)

**Užduotis**

Parašyti efektyvias rekursinių procedūrų realizacijas, nenaudojant jokių grafinių bibliotekų, kurios generuotų nurodytos struktūros BMP formato paveikslėlį (1 LD individualios užduoties pasirinkimo teste pateiktas konkretus variantas).

Reikalavimai:

1. Du realizacijos atvejai: 1) Nurodomas rekursijos gylis; 2) Generuojamas maksimalaus detalumo paveikslėlis, nurodytiems paveikslėlio matmenims.
2. Programų rezultatas BMP formato bylos demonstruojančios programos veikimą. (3 balai)
3. Eksperimentiškai nustatykite darbo laiko ir veiksmų skaičiaus priklausomybę nuo rekursijos gylio ar generuojamo paveikslėlio dydžio (taškų skaičiaus). Gautus rezultatus atvaizduokite grafikais (4 grafikai). Grafiką turi sudaryti nemažiau kaip 5 taškai ir paveikslėlio taškų skaičius turi didėti proporcingai (kartais). (2 balai)
4. Analitiškai įvertinkite procedūrų, kurios generuoja paveikslėlius, veiksmų skaičių ir laiką, sudarydami rekurentines lygtis ir jas išspręskite. Gauti rezultatai turi patvirtinti eksperimentinius rezultatus (našumo testus: vykdymo laiką ir veiksmų skaičių). (2 balai)
5. Paruoškite detalią ataskaitą su atliktais skaičiavimais. Ataskaita privalo tenkinti rašto darbų reikalavimus. (3 balai)

Tikslas:

1. **Realizacija – programos kodas**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

using System.Reflection.Metadata;

namespace BMP\_example

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Enter the width of the canvas:");

int width = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Enter the depth of the recursion:");

int depth = Convert.ToInt32(Console.ReadLine());

depth -= 1;

using (FileStream file = new FileStream("sample.bmp", FileMode.Create, FileAccess.Write))

{

int fileSize = CalculateFileSize(width, width);

file.Write(

new byte[62]

{

// Header

0x42, 0x4d,

(byte)(fileSize & 0xff), (byte)((fileSize >> 8) & 0xff), (byte)((fileSize >> 16) & 0xff), (byte)((fileSize >> 24) & 0xff),

0x0, 0x0, 0x0, 0x0,

0x36, 0x0, 0x0, 0x0,

// Information header

0x28, 0x0, 0x0, 0x0,

(byte)(width & 0xff), (byte)((width >> 8) & 0xff), (byte)((width >> 16) & 0xff), (byte)((width >> 24) & 0xff),

(byte)(width & 0xff), (byte)((width >> 8) & 0xff), (byte)((width >> 16) & 0xff), (byte)((width >> 24) & 0xff),

0x1, 0x0,

0x1, 0x0,

0x0, 0x0, 0x0, 0x0,

0x0, 0x0, 0x0, 0x0,

0x0, 0x0, 0x0, 0x0,

0x0, 0x0, 0x0, 0x0,

0x0, 0x0, 0x0, 0x0,

0x0, 0x0, 0x0, 0x0,

// Color palette

0xff, 0xff, 0xff, 0x0,

0x0, 0x0, 0x0, 0x0

});

int l = (width + 31) / 32 \* 4;

var t = new byte[width \* l];

int centerx = width / 2;

int centery = width / 2;

int size = width / 8;

PrintDiamond(t, size, centerx, centery, width, l);

CalculateDiamondCoordinates(t, centerx, centery, size\*7/3, size / 3, depth, l, width);

file.Write(t);

file.Close();

new Process

{

StartInfo = new ProcessStartInfo(@"sample.bmp")

{

UseShellExecute = true

}

}.Start();

}

}

static int CalculateFileSize(int width, int height)

{

int dataSize = ((width + 31) / 32) \* 4 \* height;

return 54 + dataSize;

}

public static void DrawLine(byte[] t, int x0, int y0, int x1, int y1, int size, int l)

{

int dx = Math.Abs(x1 - x0);

int dy = Math.Abs(y1 - y0);

int sx = x0 < x1 ? 1 : -1;

int sy = y0 < y1 ? 1 : -1;

int err = dx - dy;

int x = x0;

int y = y0;

while (true)

{

if (x >= 0 && x < size && y >= 0 && y < size)

{

int index = (y \* l) + (x / 8);

t[index] |= (byte)(0x80 >> (x % 8));

}

if (x == x1 && y == y1)

{

break;

}

int e2 = 2 \* err;

if (e2 > -dy)

{

err -= dy;

x += sx;

}

if (e2 < dx)

{

err += dx;

y += sy;

}

}

}

public static void PrintDiamond(byte[] t, int size, int centerx, int centery, int width, int l)

{

DrawLine(t, centerx, centery - size, centerx + size, centery, width, l); // desine apacia

DrawLine(t, centerx - size, centery, centerx, centery - size, width, l); ; // kaire apacia

DrawLine(t, centerx - size, centery, centerx, centery + size, width, l); // kaire virsus

DrawLine(t, centerx, centery + size, centerx + size, centery, width, l); ; // desine virsus

FillColor(t, centerx, centery, 0, 1, width, l);

}

public static void FillColor(byte[] t, int x, int y, byte targetColor, byte replacementColor, int size, int l)

{

if (x < 0 || x >= size || y < 0 || y >= size)

return;

Stack<(int, int)> stack = new Stack<(int, int)>();

stack.Push((x, y));

while (stack.Count > 0)

{

(int cx, int cy) = stack.Pop();

if (cx < 0 || cx >= size || cy < 0 || cy >= size)

continue;

int index = (cy \* l) + (cx / 8);

int bitIndex = cx % 8;

if (((t[index] >> (7 - bitIndex)) & 1) != targetColor)

continue;

t[index] &= (byte)~(0x80 >> bitIndex);

t[index] |= (byte)((replacementColor & 1) << (7 - bitIndex));

stack.Push((cx + 1, cy));

stack.Push((cx - 1, cy));

if (cy > 0)

stack.Push((cx, cy - 1));

if (cy < size - 1)

stack.Push((cx, cy + 1));

}

}

static void CalculateDiamondCoordinates(byte[] t, int centerX, int centerY, int distance, int size, int depth, int l, int width)

{

if (depth == 0) return;

if (size < 2) return;

double theta = Math.PI / 10;

for (int n = 0; n < 5; n++)

{

int x = centerX + (int)(distance \* Math.Cos(theta));

int y = centerY + (int)(distance \* Math.Sin(theta));

theta += 2 \* Math.PI / 5;

PrintDiamond(t, size, x, y, width, l);

CalculateDiamondCoordinates(t, x, y, distance \* 19/50, (int)(size / 2.5), depth - 1, l, width);

}

}

}

}

1. **Programos rezultatai**

Pirmą kartą paveikslėlis generuojamas įrašant generuojamo paviršiaus dimensijas, bei nustatant rekursijos gylį.

| Gylis = 4 | Dimensijos = 1000x1000 |
| --- | --- |
|  | |

Antrame paveikslėlyje generuojamas įrašant tik generuojamo paviršiaus dimensijas, o rekursijos gylis nustatomas automatiškai – parenkamas maksimalus gylis, nutraukiant rekursiją, kai rombas pasiekia 2 pikselių dydi.

| Gylis = max | Dimensijos = 1000x1000 |
| --- | --- |
|  | |

1. **Darbo laiko ir veiksmų skaičiaus priklausomybės – grafikai**

| **Priklausomybė nuo rekursijos gylio** | **Priklausomybė nuo generuojamo paveikslėlio dydžio (taškų skaičiaus)** |
| --- | --- |
| Darbo laikas  Chart | Darbo laikas  Chart |
| Veiksmų skaičius  Chart | Veiksmų skaičius  Chart |

| **Kodo eilutė** | **Kaina** | **Kiekis** |
| --- | --- | --- |
| static void CalculateDiamondCoordinates(byte[] t, int centerX, int centerY, int distance, int size, int depth, int l, int width, ref int count) |  |  |
| { |  |  |
| if (depth <= 0) return; | C1 | 1 |
| if (size < 2) return; | C2 | 1 |
| double theta = Math.PI / 10; | C3 | 1 |
| for (int n = 0; n < 5; n++) | C4 | 5 |
| { |  |  |
| int x = centerX + (int)(distance \* Math.Cos(theta)); | C5 | 5 |
| int y = centerY + (int)(distance \* Math.Sin(theta)); | C6 | 5 |
| theta += 2 \* Math.PI / 5; | C7 | 5 |
| PrintDiamond(t, size, x, y, width, l);  count++; | C8 | max(abs(x1-x), abs(y1-y)) + stack.Count - 1 |
| CalculateDiamondCoordinates(t, x, y, distance \* 19/50, (int)(size / 2.5), depth - 1, l, width, ref count); | C9 |  |
| } |  |  |

| **Kodo eilutė** | **Kaina** | **Kiekis** |
| --- | --- | --- |
| public static void PrintDiamond(byte[] t, int size, int centerx, int centery, int width, int l) |  |  |
| { |  |  |
| DrawLine(t, centerx, centery - size, centerx + size, centery, width, l); // desine apacia | C1 | max(abs(x1-x), abs(y1-y)) |
| DrawLine(t, centerx - size, centery, centerx, centery - size, width, l); ; // kaire apacia | C1 | max(abs(x1-x), abs(y1-y)) |
| DrawLine(t, centerx - size, centery, centerx, centery + size, width, l); // kaire virsus | C1 | max(abs(x1-x), abs(y1-y)) |
| DrawLine(t, centerx, centery + size, centerx + size, centery, width, l); ; // desine virsus | C1 | max(abs(x1-x), abs(y1-y)) |
| FillColor(t, centerx, centery, 0, 1, width, l); |  | stack.Count - 1 |
| } |  |  |

| **Kodo eilute** | **Kaina** | **Kiekis** |
| --- | --- | --- |
| public static void DrawLine(byte[] t, int x0, int y0, int x1, int y1, int size, int l) |  |  |
| { |  |  |
| int dx = Math.Abs(x1 - x0); | C1 | 1 |
| int dy = Math.Abs(y1 - y0); | C2 | 1 |
| int sx = x0 < x1 ? 1 : -1; | C3 | 1 |
| int sy = y0 < y1 ? 1 : -1; | C4 | 1 |
| int err = dx - dy; | C5 | 1 |
| int x = x0; | C6 | 1 |
| int y = y0; | C7 | 1 |
| while (true) | C8 | max(abs(x1-x), abs(y1-y)) |
| { |  |  |
| if (x >= 0 && x < size && y >= 0 && y < size) | C9 | 1 |
| { |  |  |
| int index = (y \* l) + (x / 8); | C10 |  |
| t[index] |= (byte)(0x80 >> (x % 8)); | C11 |  |
| } |  |  |
| if (x == x1 && y == y1) | C12 | 1 |
| { |  |  |
| break; |  |  |
| } |  |  |
| int e2 = 2 \* err; | C13 | 1 |
| if (e2 > -dy) | C14 | 1 |
| { |  |  |
| err -= dy; |  |  |
| x += sx; |  |  |
| } |  |  |
| if (e2 < dx) | C15 | 1 |
| { |  |  |
| err += dx; |  |  |
| y += sy; |  |  |
| } |  |  |
| } |  |  |
| } |  |  |

| **Kodo eilute** | **Kaina** | **Kiekis** |
| --- | --- | --- |
| public static void FillColor(byte[] t, int x, int y, byte targetColor, byte replacementColor, int size, int l) |  |  |
| { |  |  |
| if (x < 0 || x >= size || y < 0 || y >= size) | C1 | 1 |
| return; |  |  |
| Stack<(int, int)> stack = new Stack<(int, int)>(); | C2 | 1 |
| stack.Push((x, y)); | C3 | 1 |
| while (stack.Count > 0) | C4 | stack.Count - 1 |
| { |  |  |
| (int cx, int cy) = stack.Pop(); | C5 | 1 |
| if (cx < 0 || cx >= size || cy < 0 || cy >= size) | C6 | 1 |
| continue; |  |  |
| int index = (cy \* l) + (cx / 8); | C7 | 1 |
| int bitIndex = cx % 8; | C8 | 1 |
| if (((t[index] >> (7 - bitIndex)) & 1) != targetColor) | C9 | 1 |
| continue; |  |  |
| t[index] &= (byte)~(0x80 >> bitIndex); | C10 | 1 |
| t[index] |= (byte)((replacementColor & 1) << (7 - bitIndex)); | C11 | 1 |
| stack.Push((cx + 1, cy)); | C12 | 1 |
| stack.Push((cx - 1, cy)); | C12 | 1 |
| if (cy > 0) | C13 | 1 |
| stack.Push((cx, cy - 1)); |  |  |
| if (cy < size - 1) | C14 | 1 |
| stack.Push((cx, cy + 1)); |  |  |
| } |  |  |
| } |  |  |

**Išvados**

Parašytas programos kodas, kuris rekursiškai piešia rombus, .bmp formato nuotraukoje. Apskaičiuotas rekursijos veiksmų kiekis bei laikas keičiant rekursijos gyli ar paveikslėlio dydį. Be to, nupiešti grafikai parodo programos kodo veikimo laiko ir atliktų veiksmų kiekio priklausomybę nuo rekursijos gylio ir paveikslėlio dydžio.