Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра прикладной математики и кибернетики

Отчёт

по расчётно-графическому заданию

Вариант 2

Выполнил:

студент группы ИП-111

Кузьменок Д. В.

ФИО студента

Работу проверил: Перцев И. В.

ФИО преподавателя

Новосибирск 2025 г.

# **Содержание**

[**Содержание** 2](#_Toc191731502)

[**Задание** 3](#_Toc191731503)

[**Описание алгоритма работы программы** 4](#_Toc191731504)

[**Код программы** 5](#_Toc191731505)

[**Результаты работы** 11](#_Toc191731506)

# **Задание**

Написать программу-конвертор количества цветов в изображении.

Предлагаемый алгоритм. Для уменьшения количества цветов выбираются наиболее часто встречаемые цвета в исходном изображении. Причем эти цвета не должны быть слишком похожими друг на друга. Для сравнения цветов вычисляются разности между RGB составляющими.

Delta=(R1-R2)^2 + (G1-G2)^2+ (B1-B2)^2

После формирования новой палитры цвета в заменяются на наиболее похожие из записанных в палитру.

Можно использовать любой другой алгоритм преобразования цветов (например медианного сечения) главное требование – алгоритм должен быть реализован самостоятельно.

Программа должны выводить изображение на экран до и после конвертирования.

**Вариант задания:**

2. Преобразовать True Color BMP файл в 256-цветный BMP файл.

# **Описание алгоритма работы программы**

Вся программа выполняет несколько последовательных шагов для успешного выполнения задания.

Сначала из файла, имя которого указано в первом аргументе командной строки, прочитываются и сохраняются заголовки. Информация из них выводится на экран.

Далее рассчитывается количество бит, занимаемое одной строкой данных, и происходит чтение данных по строкам. После чего закрывается поток данных, связанных с изначальным файлом, так как больше он не будет использоваться.

Далее мы создаем не сортируемый словарь, с помощью которого мы сгруппируем цвета, чтобы исключить повторы. Основным ключом будет сумма всех трех параметров цвета, значением будет сам цвет. После создания словаря мы проходим по всем цветам пикселей и записываем данные в словарь.

Далее все собранные данные мы переписываем в вектор, сохраняя только параметры цвета. Записанные данные мы сортируем по количеству использования в порядке убывания и обрезаем вектор до размера 256, как и задано по варианту.

Далее мы создаем новый поток данных, для записи в выходной файл. Перед записью в файл, в прочитанном заголовке мы изменяем такие параметры как: количество бит на пиксель, количество используемых цветов и отступ до информации о пикселях. Два заголовка и таблицу цветов мы записываем в новый файл.

После этого мы рассчитываем количество бит, занимаемое одной строкой данных, и начинаем заполнять данные о пикселях картины. Для этого мы снова проходим по данным пикселей из входного файла и ищем максимально похожий цвет в нашей таблице цветов. Для этого мы используем формулу Delta=(R1-R2)^2 + (G1-G2)^2+ (B1-B2)^2. После использования формулы в функции мы получаем индекс цвета в таблице цветов, который мы и заносим в файл как значение пикселя. Во время данного цикла мы считали сколько бит уже было записано в файл, тем самым вычисляя разницу от необходимого нам количества бит на строку, мы заполняем оставшееся пространство нулями.

В итоге мы закрываем файл, и программа завершает свою работу.

# **Код программы**

**Program.cs**

**using** **System**;

**using** **System.IO**;

**using** **System.Collections.Generic**;

**using** **System.Linq**;

**using** **System.Runtime.InteropServices**;

**namespace** **RGR**;

**class** **Program**

{

**private** **const** **int** COLOR\_DIF = **100**;

[StructLayout(LayoutKind.Sequential, Pack = 1)]

**private** **struct** **FileHeader**

{

**public** **ushort** id;

**public** **uint** f\_size;

**public** **ushort** rez\_1, rez\_2;

**public** **uint** bm\_offset;

}

[StructLayout(LayoutKind.Sequential, Pack = 1)]

**private** **struct** **V3Header**

{

**public** **uint** h\_size;

**public** **uint** width;

**public** **uint** height;

**public** **ushort** planes, bit\_per\_pixel;

**public** **uint** compression;

**public** **uint** size\_image;

**public** **uint** h\_res;

**public** **uint** v\_res;

**public** **uint** clr\_used;

**public** **uint** clr\_imp;

}

[StructLayout(LayoutKind.Sequential, Pack = 1)]

**private** **struct** **ColorInfo**

{

**public** **byte** blue;

**public** **byte** green;

**public** **byte** red;

**public** **byte** temp;

**public** **ColorInfo**(**byte** b, **byte** g, **byte** r)

{

blue = b;

green = g;

red = r;

temp = **0**;

}

}

**private** **static** **int** **FindNead**(**byte** b, **byte** g, **byte** r, List<ColorInfo> arr)

{

**int** result = **0**;

**int** minVal = **int**.MaxValue;

**int** siz = arr.Count;

**int** temp;

**for** (**int** i = **0**; i < siz; ++i)

{

temp = (b - arr[i].blue) \* (b - arr[i].blue) +

(g - arr[i].green) \* (g - arr[i].green) +

(r - arr[i].red) \* (r - arr[i].red);

**if** (temp < minVal)

{

result = i;

minVal = temp;

}

}

**return** result;

}

**public** **static** **void** **ConvertBmp**(**string** inputFile, **string** outputFile)

{

**using** (FileStream fileIn = **new** FileStream(inputFile, FileMode.Open, FileAccess.Read))

**using** (BinaryReader reader = **new** BinaryReader(fileIn))

{

FileHeader header = ReadStruct<FileHeader>(reader);

V3Header headerV3 = ReadStruct<V3Header>(reader);

Console.WriteLine(" File header info:\n");

Console.WriteLine($"id={header.id}, f\_size={header.f\_size}, r1={header.rez\_1}, r2={header.rez\_2}, ofset={header.bm\_offset}\n \n");

Console.WriteLine(" v3 header info:\n");

Console.WriteLine($"size={headerV3.h\_size}, width={headerV3.width}, height={headerV3.height}\n");

Console.WriteLine($"planes={headerV3.planes}, bit\_per\_pix={headerV3.bit\_per\_pixel}, compression={headerV3.compression}\n");

Console.WriteLine($"size\_image={headerV3.size\_image}, h\_res={headerV3.h\_res}, v\_res={headerV3.v\_res}\n");

Console.WriteLine($"clr\_used={headerV3.clr\_used}, clr\_imp={headerV3.clr\_imp}\n \n");

fileIn.Seek(header.bm\_offset, SeekOrigin.Begin);

**int** bytesPerRow = (**int**)Math.Floor((headerV3.bit\_per\_pixel \* headerV3.width + **31**) / **32.0**) \* **4**;

List<**byte**[]> rows = **new** List<**byte**[]>();

**for** (**int** i = **0**; i < headerV3.height; ++i)

{

**byte**[] row = reader.ReadBytes(bytesPerRow);

rows.Add(row);

}

Console.WriteLine(" \* Start color selection\n");

List<ColorInfo> colors = **new** List<ColorInfo>();

**int** x = rows[**0**].Length;

**int** y = rows.Count;

**int** colorSize = **0**;

**bool** findEl;

**for** (**int** i = **0**; i < y; ++i)

{

**for** (**int** j = **0**; j < x; j += **3**)

{

findEl = **false**;

**for** (**int** ii = **0**; ii < colorSize; ++ii)

{

**int** diff = (rows[i][j] - colors[ii].blue) \* (rows[i][j] - colors[ii].blue) +

(rows[i][j + **1**] - colors[ii].green) \* (rows[i][j + **1**] - colors[ii].green) +

(rows[i][j + **2**] - colors[ii].red) \* (rows[i][j + **2**] - colors[ii].red);

**if** (diff <= COLOR\_DIF)

{

findEl = **true**;

ColorInfo tempColor = colors[ii];

tempColor.temp += **1**;

colors[ii] = tempColor;

**break**;

}

}

**if** (!findEl)

{

ColorInfo newColor = **new** ColorInfo(rows[i][j], rows[i][j + **1**], rows[i][j + **2**]);

newColor.temp = **1**;

colors.Add(newColor);

++colorSize;

}

}

}

colors = colors.OrderByDescending(c => c.temp).ToList();

**if** (colors.Count > **256**)

{

colors.RemoveRange(**256**, colors.Count - **256**);

}

**while** (colors.Count < **256**)

{

colors.Add(**new** ColorInfo(**0**,**0**,**0**));

}

Console.WriteLine(" \* End color selection\n");

**using** (FileStream fileOut = **new** FileStream(outputFile, FileMode.Create, FileAccess.Write))

**using** (BinaryWriter writer = **new** BinaryWriter(fileOut))

{

headerV3.bit\_per\_pixel = **8**;

headerV3.clr\_imp = **256**;

headerV3.clr\_used = **256**;

headerV3.size\_image = (**uint**)(Math.Floor((headerV3.bit\_per\_pixel \* headerV3.width + **31**) / **32.0**) \* **4** \* headerV3.height);

header.f\_size = (**uint**)(header.bm\_offset + headerV3.size\_image);

header.bm\_offset = (**uint**)(Marshal.SizeOf(**typeof**(FileHeader)) + Marshal.SizeOf(**typeof**(V3Header)) + **256** \* Marshal.SizeOf(**typeof**(ColorInfo)));

WriteStruct(writer, header);

WriteStruct(writer, headerV3);

**foreach** (**var** color **in** colors)

{

WriteStruct(writer, color);

}

Console.WriteLine(" \* Start write in file\n");

bytesPerRow = (**int**)Math.Floor((headerV3.bit\_per\_pixel \* headerV3.width + **31**) / **32.0**) \* **4**;

**for** (**int** i = **0**; i < y; ++i)

{

**int** curByte = **0**;

**for** (**int** j = **0**; j < x; j += **3**)

{

**byte** val = (**byte**)FindNead(rows[i][j], rows[i][j + **1**], rows[i][j + **2**], colors);

writer.Write(val);

++curByte;

}

**for** (**int** j = curByte; j < bytesPerRow; ++j)

{

writer.Write((**byte**)**0**);

}

}

Console.WriteLine(" \* End write in file\n");

}

}

}

**private** **static** T ReadStruct<T>(BinaryReader reader) **where** T : **struct**

{

**byte**[] bytes = reader.ReadBytes(Marshal.SizeOf(**typeof**(T)));

GCHandle handle = GCHandle.Alloc(bytes, GCHandleType.Pinned);

T theStruct = (T)Marshal.PtrToStructure(handle.AddrOfPinnedObject(), **typeof**(T));

handle.Free();

**return** theStruct;

}

**private** **static** **void** WriteStruct<T>(BinaryWriter writer, T obj) **where** T : **struct**

{

**int** size = Marshal.SizeOf(obj);

**byte**[] arr = **new** **byte**[size];

IntPtr ptr = Marshal.AllocHGlobal(size);

Marshal.StructureToPtr(obj, ptr, **true**);

Marshal.Copy(ptr, arr, **0**, size);

Marshal.FreeHGlobal(ptr);

writer.Write(arr);

}

**static** **void** **Main**(**string**[] args)

{

**if** (args.Length != **2**)

{

Console.WriteLine("Usage: BmpConverter <input.bmp> <output.bmp>");

**return**;

}

**string** inputFile = args[**0**];

**string** outputFile = args[**1**];

**try**

{

ConvertBmp(inputFile, outputFile);

Console.WriteLine("Conversion successful!");

**int** uniqueColorCountinput = CountUniqueColors(inputFile);

Console.WriteLine($"Unique colors in output file: {uniqueColorCountinput}");

**int** uniqueColorCountoutput = CountUniqueColors(outputFile);

Console.WriteLine($"Unique colors in output file: {uniqueColorCountoutput}");

}

**catch** (Exception ex)

{

Console.WriteLine($"Error: {ex.Message}");

}

}

**public** **static** **int** **CountUniqueColors**(**string** bmpFile)

{

HashSet<**int**> uniqueColors = **new** HashSet<**int**>();

**using** (FileStream fileIn = **new** FileStream(bmpFile, FileMode.Open, FileAccess.Read))

**using** (BinaryReader reader = **new** BinaryReader(fileIn))

{

FileHeader header = ReadStruct<FileHeader>(reader);

V3Header headerV3 = ReadStruct<V3Header>(reader);

fileIn.Seek(header.bm\_offset, SeekOrigin.Begin);

**int** bytesPerRow = (**int**)Math.Floor((headerV3.bit\_per\_pixel \* headerV3.width + **31**) / **32.0**) \* **4**;

**for** (**int** i = **0**; i < headerV3.height; ++i)

{

**byte**[] row = reader.ReadBytes(bytesPerRow);

**if** (headerV3.bit\_per\_pixel == **8**)

{

**for** (**int** j = **0**; j < row.Length; ++j)

{

uniqueColors.Add(row[j]);

}

}

**else** **if** (headerV3.bit\_per\_pixel == **24**)

{

**for** (**int** j = **0**; j < row.Length; j += **3**)

{

**int** color = (row[j] << **16**) | (row[j + **1**] << **8**) | row[j + **2**];

uniqueColors.Add(color);

}

}

}

}

**return** uniqueColors.Count;

}

}

# **Результаты работы**

Результат работы программы:

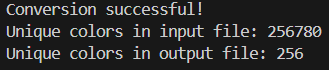
Изображение выглядит как рыба, Биология океана, Организм, подводный

Автоматически созданное описание

Исходное изображение



Преобразованное изображение



Подсчет количества цветов у изображений