

Описание Алгоритма

Принятые допущения:

1. Алгоритм загружает данные из файла «input.txt» который находится в том же каталоге.
2. В файле слои разделены пустыми строками, а ряды разделены знаком конца строки.

Решение задачи

Для решение поставленной задачи, нам необходимо где-то хранить полученные данные. Задача предлагает хранить все в одномерном массиве, но это будет очень не удобно для реализации и читабельности кода, по этому хранение ячеек будет реализовано в трех-мерном массиве.

```
public static Slot[, ,] DataSlot = new Slot[Nmax, Nmax, Nmax];
```

Из условия задачи известно, что данные могут быть только «0» или «1», по этому для экономии памяти процессора значение каждой ячейки будем хранить в перечисляемом типе **Slot**:

```
enum Slot: byte
{
    check = 200,
    zero = 0,
    value = 1
}
```

значение **check** нам понадобится когда будем обходить массив данных.

Ввод данных реализуем из файла "**input.txt**" с помощью стандартного класса **StreamReader** . Описывать работу ввода вывода не буду, просто приложу листинг:

```
// ===== Ввод =====
string path = Path.Combine (Environment.CurrentDirectory, "input.txt");
StreamReader input = new StreamReader (path);
Nx = int.Parse (input.ReadLine());
Ny = int.Parse (input.ReadLine());
Nz = int.Parse (input.ReadLine());

Console.WriteLine ( "Nx = " + Nx + "\t Ny = " + Ny + "\t Nz = " + Nz);
Console.WriteLine ("Загружаю файл: " + path);
int m;
for (int k = 0; k < Nz; k++) {
    for (int j = 0; j < Ny; j++) {
        for (int i = 0; i < Nx; i++) {
            m = input.Read ();
            DataSlot [i, j, k] = (Slot)(m - 48);
        }
    }
}
```

```

        input.ReadLine ();
    }
    input.ReadLine ();
    Console.Write ($"Загружаю слой #{k} \r"); // Чтобы пользователю не было
    скучно смотреть в пустой экран!
}
input.Close();
Console.WriteLine ("Файл загружен. Время " + watch.ElapsedMilliseconds + "мс.");

```

Теперь когда в памяти программы есть все необходимое, приступаем к обходу массива данных. для этого будем использовать три вложенных цикла **for**:

```

for (int k = 0; k < Nz; k++) {
    for (int j = 0; j < Ny; j++) {
        for (int i = 0; i < Nx; i++) {
            // Операторы выполняемые при обходе этой ячейки
        }
    }
}

```

Выделяем только те ячейки которые имеют значение **Slot.value**

```

if (DataSlot [i, j, k] == Slot.value) {
    // Приступаем к исследованию этой области
} else
    DataSlot [i, j, k] = Slot.check;

```

Изначально эту задачу я решил рекурсией:

```

public static void AddSlot(int i, int j, int k)
{
    if (DataSlot [i, j, k] == Slot.value) {
        DataSlot [i, j, k] = Slot.check;
        Area[NowArea]= i+ Nx*(j-1)+Ny*Nx*(k-1);
        NowArea++;
        //Console.Write ($"Нашла {NowArea} ячеек! \r");
        if (i < Nx - 1)
            AddSlot (i + 1, j, k);
        if (i > 0)
            AddSlot (i - 1, j, k);
        if (j < Ny - 1)
            AddSlot (i, j + 1, k);
        if (j > 0)
            AddSlot (i, j - 1, k);
        if (k < Nz - 1)
            AddSlot (i, j, k + 1);
        if (k > 0)
            AddSlot (i, j, k - 1);
    }
}

```

И программа прекрасно работала до размеров куба 200*100*100, далее произошло переполнение стека вызовы функции, и программу пришлось переделать под **обход графа в ширину**¹ с использованием стека:

1 Известная задача программирования

Стек реализован на хранение координат, по этому:

```
struct Point
{
    public int x, y, z;
    public Point(int newX, int newY, int newZ)
    {
        this.x = newX;
        this.y = newY;
        this.z = newZ;
        MainClass.DataSlot [x , y, z] = Slot.check2;
    }
    public void Take(int newX, int newY, int newZ)
    {
        this.x = newX;
        this.y = newY;
        this.z = newZ;
        MainClass.DataSlot [x , y, z] = Slot.check;
    }
}
```

Структура **Point** подходит для реализации как нельзя лучше.

Стек из координат будем реализовывать в виде очереди:

```
Queue<Point> Area = new Queue<Point>();
Point m = new Point (x, y, z); // оперативная единица для добавления в очередь
Area.Enqueue (m); // Первая ячейка в стеке
```

Будем выполнять цикл пока не опустеет очередь, это будет означать что данный граф обойден, и область записана:

```
do {
    //Найти всех соседей этой ячейки и добавить их в очередь
} while (Area.Count > 0); // Выполняем цикл пока очередь не опустеет
```

Складывать найденные ячейки будем именно сюда

```
public static List<int> OutArea = new List<int>();
```

Но поскольку программа требует номер ячейки в одномерном массиве, то придётся перевести координаты **x, y и z** в номер ячейки в одномерном массиве.

```
Index = Nx*Ny*(m.z)+Nx*(m.y)+m.x+1;
```

После чего добавляем номер ячейки в лист **OutArea**:

```
OutArea.Add(Nx*Ny*(m.z)+Nx*(m.y)+m.x+1); //Добавляем ячейку для вывода
```

Соседей у ячейки в трех-мерном пространстве 6, их и будем проверять.

Сосед справа:

```
if ((x < Nx - 1) && (DataSlot [x + 1, y, z] == Slot.value))
// Добавляем соседа только если он не выходит за пределы области (x < Nx - 1)
// И если у него значение 1 (DataSlot [x + 1, y, z] == Slot.value)
{
    m.Take(x + 1, y, z); // засовываем координаты в структуру
    Area.Enqueue (m); // Добавляем этого соседа в очередь
}
```

Аналогично со всеми остальными соседями:

```
// Сосед справа
if ((x < Nx - 1) && (DataSlot [x + 1, y, z] == Slot.value))
{m.Take(x + 1, y, z);Area.Enqueue (m);}
//Сосед слева
if ((x-1 > 0) && (DataSlot [x - 1, y, z] == Slot.value))
{m.Take(x - 1, y, z); Area.Enqueue (m);}
//Сосед снизу
if ((y < Ny - 1) && (DataSlot [x, y + 1, z] == Slot.value))
{m.Take(x, y + 1, z); Area.Enqueue (m);}
//Сосед сверху
if ((y-1 > 0) && (DataSlot [x, y - 1, z] == Slot.value))
```

```

        {m.Take(x, y - 1, z); Area.Enqueue (m);}
//Сосед перед ним(3 измерение)
if ((z < Nz - 1) && (DataSlot [x, y, z + 1] == Slot.value))
    {m.Take(x, y, z + 1);Area.Enqueue (m);}
//Сосед за ним(3 измерение)
if ((z-1 > 0) && (DataSlot [x, y, z - 1] == Slot.value))
    {m.Take(x, y, z - 1);Area.Enqueue (m);}

```

В итоге имеем метод **AddSlot**

```

public static void AddSlot(int i, int j, int k)
{
    int x = i; int y = j; int z = k;
    // Для первой ячейки выставляем значение проверено
    DataSlot[x, y, z] = Slot.check;
    Queue<Point> Area = new Queue<Point>(); //Создаём очередь
    Point m = new Point (x, y, z); // оперативная единица
    Area.Enqueue (m); // Первая ячейка в стеке
    do { //Найти всех соседей этой ячейки и добавить их в очередь
        m = Area.Dequeue ();
        x = m.x; y = m.y; z = m.z;
        OutArea.Add(Nx*Ny*(m.z)+Nx*(m.y)+m.x+1); //Добавляем ячейку для вывода
        if ((x < Nx - 1) && (DataSlot [x + 1, y, z] == Slot.value))
            {m.Take(x + 1, y, z);Area.Enqueue (m);}
        if ((x-1 > 0) && (DataSlot [x - 1, y, z] == Slot.value))
            {m.Take(x - 1, y, z); Area.Enqueue (m);}
        if ((y < Ny - 1) && (DataSlot [x, y + 1, z] == Slot.value))
            {m.Take(x, y + 1, z); Area.Enqueue (m);}
        if ((y-1 > 0) && (DataSlot [x, y - 1, z] == Slot.value))
            {m.Take(x, y - 1, z); Area.Enqueue (m);}
        if ((z < Nz - 1) && (DataSlot [x, y, z + 1] == Slot.value))
            {m.Take(x, y, z + 1); Area.Enqueue (m); }
        if ((z-1 > 0) && (DataSlot [x, y, z - 1] == Slot.value))
            { m.Take(x, y, z - 1); Area.Enqueue (m); }
    } while (Area.Count > 0); // Выполняем цикл пока очередь не опустеет
}

```

Вызываем этот метод каждый раз как встречаем значение **Slot.value**

```

for (int k = 0; k < Nz; k++) {
    for (int j = 0; j < Ny; j++) {
        for (int i = 0; i < Nx; i++) {
            if (DataSlot [i, j, k] == Slot.value) {
                AddSlot (i, j, k);
                // Вывод в файл output
                output.WriteLine ($"Область №{CountArea} ({OutArea.Count})");
                int count = 0;
                foreach (int element in OutArea)
                {
                    count++;
                    output.WriteLine($"Ячейка #{count}: {element}");
                }
                OutArea.Clear ();
            } else
                DataSlot [i, j, k] = Slot.check;
        }
    }
}

```

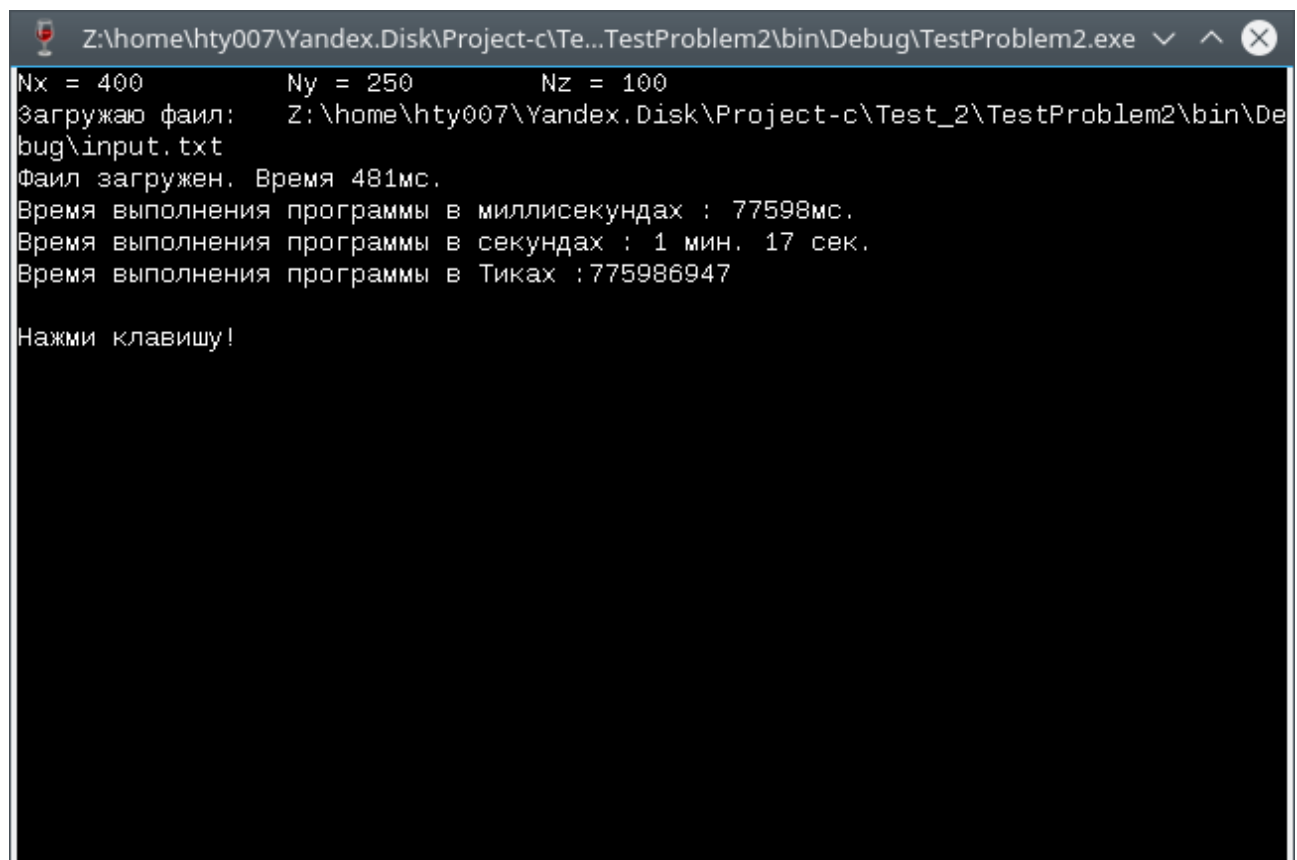
В результате выполнения получаем список ячеек **OutArea**, который выводи в файл **output.txt**.

Время работы программы

Программу я тестировал в ОС Linux Mint 18.2. По этому время в Реальной Windows будет сильно отличаться!

Объем	Частота «1» (1-5)	Время выполнения	
		Windows(эмуляция)	Linux
50*50*50	5	0,6 с	0,5 с
100*100*100	2	7 с	1 с
100*100*100	4	13 с	1 с
200*200*200	2	48 с	10 с
250*250*250	2	120 с	20 с
300*300*300	3	-	34 с
400*250*100	2	(1:17)77 с / Экран 1	12 с /Экран 2
400*250*100	3	более 10 минут	12 с
400*250*100	4	-	13с
400*250*100	5	-	13 с
500*500*500	2	-	((2:38) 158 с

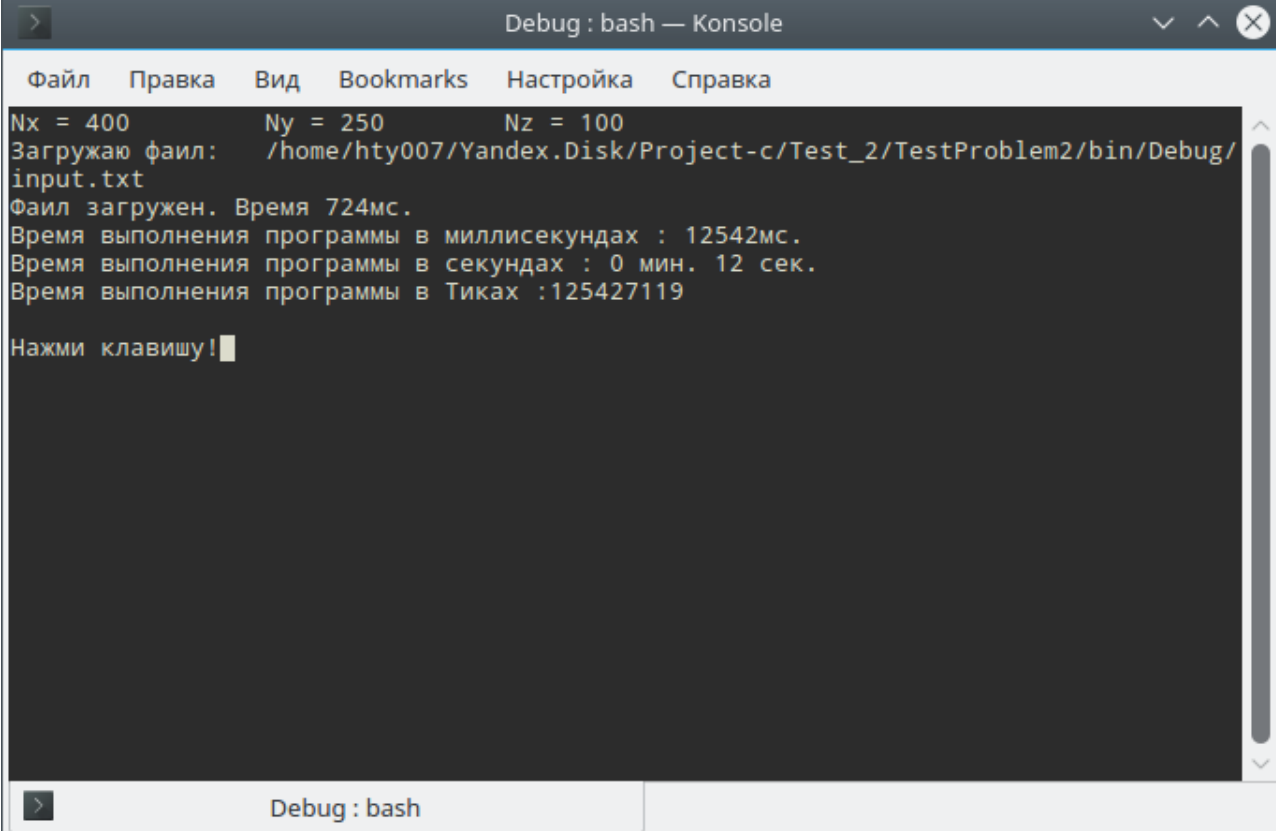
Экран 1



```
Z:\home\hty007\Yandex.Disk\Project-c\Te...TestProblem2\bin\Debug\TestProblem2.exe
Nx = 400      Ny = 250      Nz = 100
Загружаю файл:  Z:\home\hty007\Yandex.Disk\Project-c\Test_2\TestProblem2\bin\De
bug\input.txt
Файл загружен. Время 481мс.
Время выполнения программы в миллисекундах : 77598мс.
Время выполнения программы в секундах : 1 мин. 17 сек.
Время выполнения программы в Тиках :775986947

Нажми клавишу!
```

Экран 2



```
Debug : bash — Konsole
Файл  Правка  Вид  Bookmarks  Настройка  Справка
Nx = 400      Ny = 250      Nz = 100
Загружаю файл: /home/hty007/Yandex.Disk/Project-c/Test_2/TestProblem2/bin/Debug/
input.txt
Файл загружен. Время 724мс.
Время выполнения программы в миллисекундах : 12542мс.
Время выполнения программы в секундах : 0 мин. 12 сек.
Время выполнения программы в Тиках :125427119

Нажми клавишу!
```