Очередь (queue) — это структура данных, которая работает по принципу "первый вошел, первый вышел" (First-In, First-Out, FIFO). Элементы добавляются в конец очереди, а удаление происходит из ее начала.

Очереди широко используются в программировании, например, для реализации буферов обмена, планирования задач, обработки запросов и многих других задач.

Пример использования очереди может быть, например, в программе, которая обрабатывает запросы на сервере. Каждый новый запрос добавляется в очередь, а сервер обрабатывает их по порядку, сначала обрабатывая первый запрос в очереди, затем второй и т.д.

Вот пример кода на C#, который использует очередь для обработки запросов:

using System;

using System.Collections.Generic;

class Program

{

static void Main(string[] args)

{

// создаем очередь запросов

Queue<string> requests = new Queue<string>();

// добавляем запросы в очередь

requests.Enqueue("Запрос 1");

requests.Enqueue("Запрос 2");

requests.Enqueue("Запрос 3");

// обрабатываем запросы по порядку

while (requests.Count > 0)

{

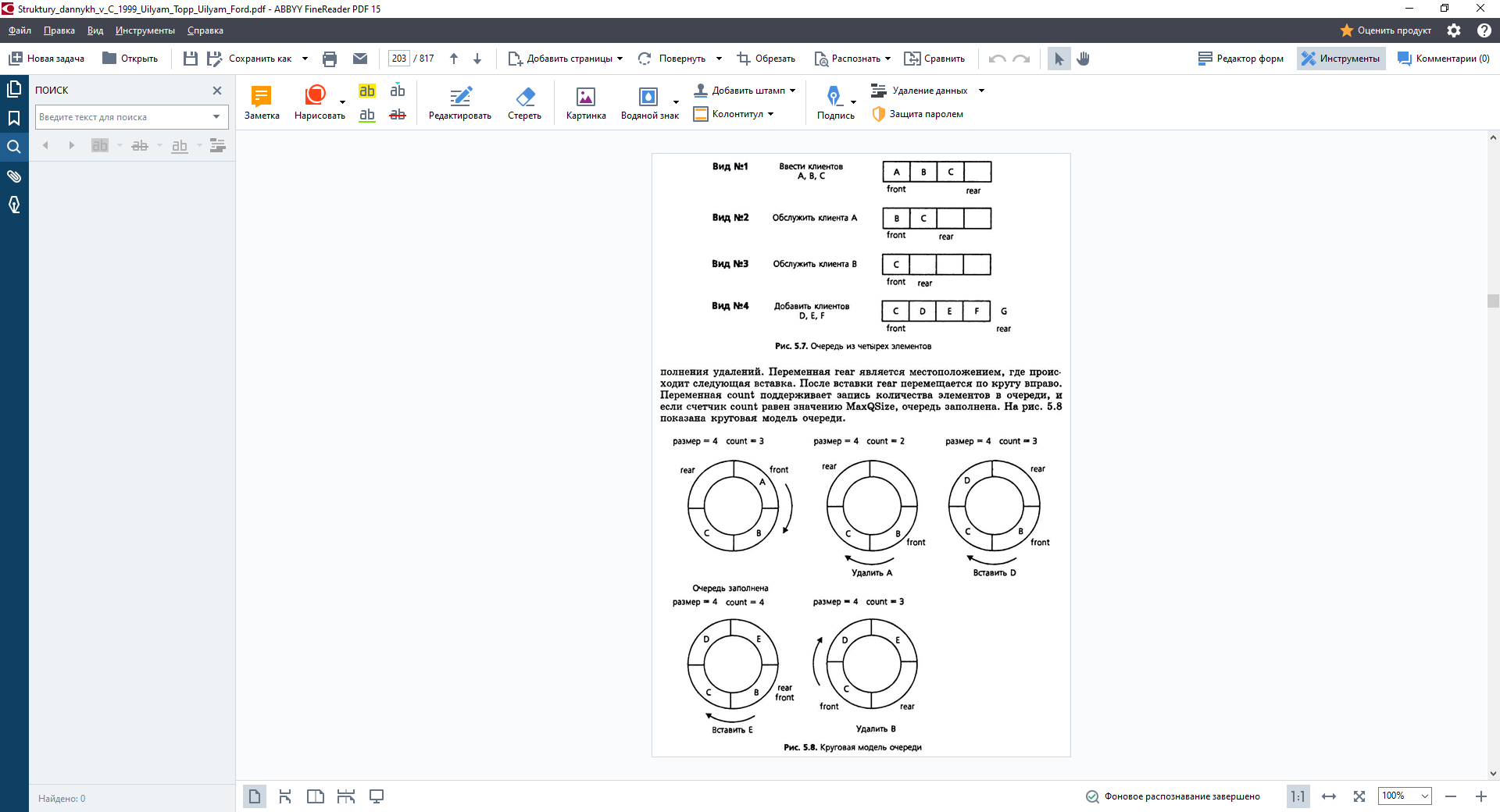
string request = requests.Dequeue();

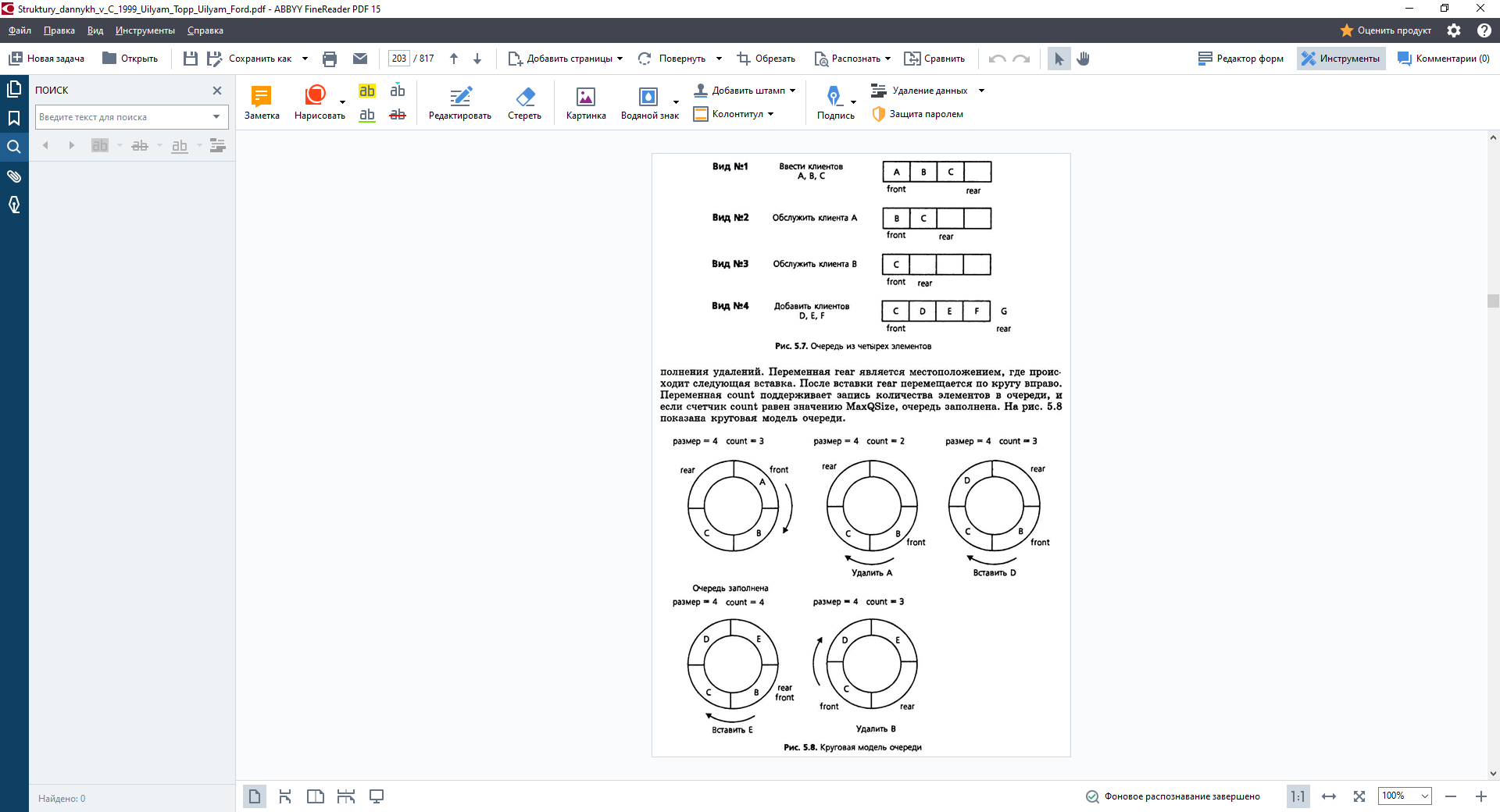
Console.WriteLine("Обработка запроса: " + request);

}

}

}







Мой класс Queue на C#

public class Queue<T>

{

private T[] elements; // массив элементов очереди

private int front; // индекс начала очереди

private int rear; // индекс конца очереди

private int count; // количество элементов в очереди

private const int defaultCapacity = 4; // стандартная емкость очереди

public Queue()

{

elements = new T[defaultCapacity]; // массив для хранения элементов очереди.

front = 0; // индекс первого элемента в очереди(с самым высоким приоритетом).

rear = -1; // индекс последнего элемента в очереди (с самым низким приоритетом).

count = 0;// количество элементов в очереди.

}

public Queue(int capacity)

{

elements = new T[capacity];

front = 0;

rear = -1;

count = 0;

}

public int Count // свойство для получения количества элементов в очереди

{

get { return count; }

}

public void Enqueue(T item) // метод для добавления элемента в конец очереди

{

if (count == elements.Length)// заполнена ли очередь

{

// Если очередь заполнена, увеличиваем ее емкость вдвое

T[] newElements = new T[2 \* elements.Length]; // в него копируются элементы из текущего массива elements

Array.Copy(elements, front, newElements, 0, elements.Length - front);

if (rear < front) // Если rear меньше front,то элементы до индекса rear копируются в начало newElements

{

Array.Copy(elements, 0, newElements, elements.Length - front, rear + 1);

}

//переменные elements, front и rear обновляются,

// указывая на новый массив и его новые границы.

elements = newElements;

front = 0;

rear = count - 1;

}

if (rear == elements.Length - 1) // находится ли rear в конце массива elements

{

rear = 0; // устанавливается в 0, чтобы элементы могли быть добавлены в начало массива

}

else

{

rear++;

}

elements[rear] = item; // новый элемент item добавляется в конец очереди

count++; // и количество элементов в очереди увеличивается на 1

}

public T Dequeue() // метод для удаления элемента из начала очереди

{

if (count == 0) // Если в очереди нет элементов, вызывается исключение

{

throw new InvalidOperationException("Queue is empty.");

}

T item = elements[front]; // Элемент в начале очереди , сохраняется в item.

elements[front] = default(T); // Элемент в начале очереди удаляется и заменяется значением по умолчанию для типа T

if (front == elements.Length - 1) //Если front указывает на последний элемент в elements

{

front = 0; // то front устанавливается на 0

}

else

{

front++; // иначе индекс увеличивается на 1.

}

count--; // Уменьшается счетчик элементов count.

return item; // Метод возвращает удаленный элемент item.

}

public T Peek() // метод для получения элемента из начала очереди без его удаления

{

if (count == 0)

{

throw new InvalidOperationException("Queue is empty.");

}

return elements[front];

}

public bool Contains(T item) // метод для проверки наличия элемента в очереди

{

int index = front; // проходит по всем элементам очереди, начиная с переднего

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

{

if (Equals(elements[index], item)) // совпадает ли каждый элемент с искомым элементом item.

{

return true; // Если находится совпадение, метод возвращает true

}

if (index == elements.Length - 1) // Если index указывает на последний элемент в elements,

// то index = на 0,чтобы продолжить поиск с начала массива.

{

index = 0;

}

else // Иначе индекс инкрементируется на 1, чтобы перейти к следующему элементу.

{

index++;

}

}

return false; // возвращает значение false, указывая на то, что элемент не найден в очереди.

}

public void Clear() // метод для очистки очереди

{

Array.Clear(elements, 0, elements.Length); // очищается весь массив , заполняя его значениями по умолчанию для типа T

front = 0; // Устанавливается начальный индекс front в 0, указывая на начало очереди.

rear = -1; // Устанавливается индекс rear в -1, указывая на отсутствие элементов в конце очереди.

count = 0; // Счетчик элементов count устанавливается в 0, что означает, что очередь теперь пуста.

}

}

АТД для класса Queue<T>:

Данные:

массив элементов очереди;

индекс начала очереди;

индекс конца очереди;

количество элементов в очереди;

стандартная емкость очереди.

Операции:

**конструктор** без параметров для создания пустой очереди;

**конструктор** с параметром количества элементов в очереди для создания пустой очереди заданной емкости;

**Количество элементов:**

вход: ничего

предусловия: отсутствуют;

процесс: ничего

выход: Количество элементов;

постусловия: нет

**Добавить в конец очереди:**

вход: элемент item типа T, добавляемый в конец очереди;

предусловия: отсутствуют;

процесс: добавление элемента item в конец очереди;

выход: отсутствует;

постусловия: количество элементов в очереди увеличивается на 1, элемент item добавляется в конец очереди.

**Удалить из начала очереди:**

вход: отсутствует;

предусловия: очередь не пуста;

процесс: удаление элемента из начала очереди;

выход: удаленный элемент типа T;

постусловия: количество элементов в очереди уменьшается на 1, элемент из начала очереди удаляется.

**Получить элемент из начала очереди**

вход: отсутствует;

предусловия: очередь не пуста;

процесс: получение элемента из начала очереди без его удаления;

выход: элемент типа T из начала очереди;

постусловия: количество элементов в очереди не изменяется.

**Наличие элемента**

вход: элемент item типа T, наличие которого необходимо проверить в очереди;

предусловия: отсутствуют;

процесс: проверка наличия элемента item в очереди;

выход: true, если элемент найден, иначе false;

постусловия: количество элементов в очереди не изменяется.

source: https://metanit.com/sharp/tutorial/4.7.php