# ЛАБОРАТОРНАЯ РАБОТА №10

**Тема:** Продвинутые алгоритмы сортировки. Динамическое программирование. Алгоритмы на графах.

**Цель:** ознакомиться с быстрыми алгоритмами сортировки, а также принципами динамического программирования; научиться программно работать с графами, производить обход графов в ширину и высоту.

## Ход работы

## Вариант 8

1. Реализовать в виде отдельных функций алгоритмы сортировки элементов массива (четные номера вариантов – по возрастанию, нечетные номера – по убыванию): слиянием, пирамидальная, быстрая. Каждую функцию вызвать 3 раза для разных входных данных: 1) массив из 100 000 элементов типа int, сгенерированный случайным образом; 2) тот же массив, отсортированный в порядке возрастания элементов; 3) тот же массив, отсортированный в порядке убывания элементов. Вывести на консоль и сравнить время работы всех алгоритмов в каждом случае («секунды : миллисекунды»). Вывести количество сравнений и перестановок элементов для каждого метода сортировки во всех трех случаях. Результаты сортировки программно записать в файл sorted.dat. Программно проверить, что данные были действительно отсортированы.

```
Ссылок: 0
static void Main()
{
    Random rand = new Random();
    int n = 100000;
    int[] arr1 = new int[n];
    for (int i = 0; i < n; i++)
    {
        arr1[i] = rand.Next(-100000, 100000);
    }


    int[] arr2 = (int[])arr1.Clone(); // массив по возрастанию
    Array.Sort(arr2);

    int[] arr3 = (int[])arr1.Clone(); // массив по убыванию
    Array.Sort(arr3);
    Array.Reverse(arr3);

    // Путь к файлу для записи результатов
    string filePath = "sorted.dat";

    SortAndMeasure(arr1, "Массив случайных чисел", "Сортировка слиянием", filePath);
    SortAndMeasure(arr2, "Массив по возрастанию", "Сортировка слиянием", filePath);
    SortAndMeasure(arr3, "Массив по убыванию", "Сортировка слиянием", filePath);

    SortAndMeasure(arr1, "Массив случайных чисел", "Пирамидальная сортировка", filePath);
    SortAndMeasure(arr2, "Массив по возрастанию", "Пирамидальная сортировка", filePath);
    SortAndMeasure(arr3, "Массив по убыванию", "Пирамидальная сортировка", filePath);

    SortAndMeasure(arr1, "Массив случайных чисел", "Быстрая сортировка", filePath);
    SortAndMeasure(arr2, "Массив по возрастанию", "Быстрая сортировка", filePath);
    SortAndMeasure(arr3, "Массив по убыванию", "Быстрая сортировка", filePath);
}
```

```
// Функция для сортировки с измерением времени и выводом результатов
Ссылок: 9
static void SortAndMeasure(int[] arr, string description, string sortType, string filePath)
{
    long comparisons = 0;
    long swaps = 0;
    var stopwatch = Stopwatch.StartNew();

    int[] arrCopy = (int[])arr.Clone();
    switch (sortType)
    {
        case "Сортировка слиянием":
            MergeSort(arrCopy, ref comparisons, ref swaps);
            break;
        case "Пирамидальная сортировка":
            HeapSort(arrCopy, ref comparisons, ref swaps);
            break;
        case "Быстрая сортировка":
            QuickSort(arrCopy, 0, arrCopy.Length - 1, ref comparisons, ref swaps);
            break;
    }

    stopwatch.Stop();
    TimeSpan elapsed = stopwatch.Elapsed;

    File.WriteAllText(filePath, string.Empty); // Очищаем файл

    // Записываем в файл только числа
    File.AppendAllText(filePath, string.Join(Environment.NewLine, arrCopy) + Environment.NewLine);

    // Вывод в консоль
    Console.WriteLine($"{description} ({sortType}): {elapsed.Seconds} секунд {elapsed.Milliseconds} миллисекунд");
    Console.WriteLine($"Количество сравнений: {comparisons}, перестановок: {swaps}");

    bool isSorted = IsSorted(filePath);
    Console.WriteLine($"Проверка сортировки для {description} ({sortType}):");
    Console.WriteLine($"Массив отсортирован в порядке возрастания: {isSorted}\n");
}
```

```
// Сортировка слиянием
Ссылок: 1
static void MergeSort(int[] arr, ref long comparisons, ref long swaps)
{
    int[] tempArr = new int[arr.Length];
    MergeSortRecursive(arr, tempArr, 0, arr.Length - 1, ref comparisons, ref swaps);
}

Ссылок: 3
static void MergeSortRecursive(int[] arr, int[] tempArr, int left, int right, ref long comparisons, ref long swaps)
{
    if (left < right)
    {
        int mid = (left + right) / 2;
        MergeSortRecursive(arr, tempArr, left, mid, ref comparisons, ref swaps);
        MergeSortRecursive(arr, tempArr, mid + 1, right, ref comparisons, ref swaps);
        Merge(arr, tempArr, left, mid, right, ref comparisons, ref swaps);
    }
}
```

```
static void Merge(int[] arr, int[] tempArr, int left, int mid, int right, ref long comparisons, ref long swaps)
{
    int leftEnd = mid;
    int rightEnd = right;
    int tempPos = left;
    int length = right - left + 1;

    int leftIndex = left;
    int rightIndex = mid + 1;

    while (leftIndex <= leftEnd && rightIndex <= rightEnd)
    {
        comparisons++;
        if (arr[leftIndex] <= arr[rightIndex])
        {
            tempArr[tempPos++] = arr[leftIndex++];
        }
        else
        {
            tempArr[tempPos++] = arr[rightIndex++];
            swaps++;
        }
    }

    while (leftIndex <= leftEnd)
    {
        tempArr[tempPos++] = arr[leftIndex++];
    }

    while (rightIndex <= rightEnd)
    {
        tempArr[tempPos++] = arr[rightIndex++];
    }

    for (int i = 0; i < length; i++)
    {
        arr[left + i] = tempArr[left + i];
    }
}
```

```
// Пирамидальная сортировка
Ссылок: 1
static void HeapSort(int[] arr, ref long comparisons, ref long swaps)
{
    int n = arr.Length;
    for (int i = n / 2 - 1; i >= 0; i--)
    {
        Heapify(arr, n, i, ref comparisons, ref swaps);
    }

    for (int i = n - 1; i > 0; i--)
    {
        Swap(arr, 0, i, ref swaps);
        Heapify(arr, i, 0, ref comparisons, ref swaps);
    }
}
```

```
Ссылок: 3
static void Heapify(int[] arr, int n, int i, ref long comparisons, ref long swaps)
{
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    comparisons++;

    if (left < n && arr[left] > arr[largest])
    {
        largest = left;
    }

    if (right < n && arr[right] > arr[largest])
    {
        largest = right;
    }

    if (largest != i)
    {
        Swap(arr, i, largest, ref swaps);
        Heapify(arr, n, largest, ref comparisons, ref swaps);
    }
}
```

```
// Функция обмена элементов
Ссылок: 5
static void Swap(int[] arr, int i, int j, ref long swaps)
{
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    swaps++;
}
```

```
// Быстрая сортировка
Ссылок: 1
static void QuickSort(int[] arr, int low, int high, ref long comparisons, ref long swaps)
{
    Stack<int> stack = new Stack<int>();
    stack.Push(low);
    stack.Push(high);

    while (stack.Count > 0)
    {
        high = stack.Pop();
        low = stack.Pop();

        if (low < high)
        {
            int pi = Partition(arr, low, high, ref comparisons, ref swaps);

            // Пушим в стек правую и левую части массива для дальнейшей сортировки
            if (pi - 1 > low)
            {
                stack.Push(low);
                stack.Push(pi - 1);
            }

            if (pi + 1 < high)
            {
                stack.Push(pi + 1);
                stack.Push(high);
            }
        }
    }
}
```

```
static int Partition(int[] arr, int low, int high, ref long comparisons, ref long swaps)
{
    // Выбор медианы из трех элементов
    int middle = low + (high - low) / 2;
    int pivotIndex = MedianOfThree(arr, low, middle, high);

    // Меняем опорный элемент с последним элементом для совместимости с Partition
    Swap(arr, pivotIndex, high, ref swaps);

    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++)
    {
        comparisons++;
        if (arr[j] < pivot)
        {
            i++;
            Swap(arr, i, j, ref swaps);
        }
    }

    Swap(arr, i + 1, high, ref swaps); // Помещаем опорный элемент в нужную позицию
    return (i + 1);
}
```

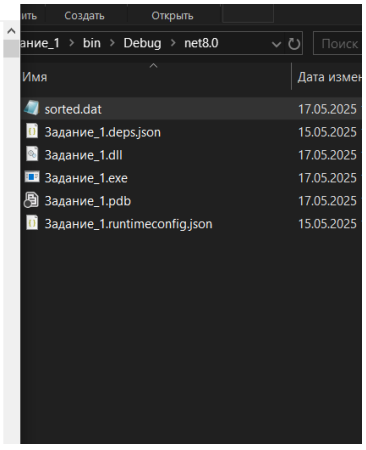
```
// Функция для выбора медианы из трех элементов
Ссылок: 1
static int MedianOfThree(int[] arr, int low, int middle, int high)
{
    int a = arr[low];
    int b = arr[middle];
    int c = arr[high];

    if ((a > b) == (a > c))
        return low;
    else if ((b > a) == (b > c))
        return middle;
    else
        return high;
}
```

```
// Функция для проверки, отсортирован ли массив
Ссылок: 1
static bool IsSorted(string filePath)
{
    try
    {
        string[] lines = File.ReadAllLines(filePath);

        int[] arr = lines.Select(line => int.Parse(line)).ToArray();

        // Проверка, отсортирован ли массив
        for (int i = 1; i < arr.Length; i++)
        {
            if (arr[i - 1] > arr[i])
            {
                return false;
            }
        }
        return true;
    }
    catch (Exception ex)
    {
        Console.WriteLine("Ошибка при чтении из файла: " + ex.Message);
        return false;
    }
}
```

```
Массив отсортирован в порядке возрастания: True

Массив по возрастанию (Сортировка слиянием): 0 секунд 9 миллисекунд
Количество сравнений: 853904, перестановок: 0
Проверка сортировки для Массив по возрастанию (Сортировка слиянием):
Массив отсортирован в порядке возрастания: True

Массив по убыванию (Сортировка слиянием): 0 секунд 11 миллисекунд
Количество сравнений: 830190, перестановок: 806491
Проверка сортировки для Массив по убыванию (Сортировка слиянием):
Массив отсортирован в порядке возрастания: True

Массив случайных чисел (Пирамидальная сортировка): 0 секунд 25 миллисекунд
Количество сравнений: 1625527, перестановок: 1575527
Проверка сортировки для Массив случайных чисел (Пирамидальная сортировка):
Массив отсортирован в порядке возрастания: True

Массив по возрастанию (Пирамидальная сортировка): 0 секунд 21 миллисекунд
Количество сравнений: 1698891, перестановок: 1648891
Проверка сортировки для Массив по возрастанию (Пирамидальная сортировка):
Массив отсортирован в порядке возрастания: True

Массив по убыванию (Пирамидальная сортировка): 0 секунд 19 миллисекунд
Количество сравнений: 1547569, перестановок: 1497569
Проверка сортировки для Массив по убыванию (Пирамидальная сортировка):
Массив отсортирован в порядке возрастания: True

Массив случайных чисел (Быстрая сортировка): 0 секунд 18 миллисекунд
Количество сравнений: 2159863, перестановок: 1233379
Проверка сортировки для Массив случайных чисел (Быстрая сортировка):
Массив отсортирован в порядке возрастания: True

Массив по возрастанию (Быстрая сортировка): 8 секунд 776 миллисекунд
Количество сравнений: 4999950000, перестановок: 199998
Проверка сортировки для Массив по возрастанию (Быстрая сортировка):
Массив отсортирован в порядке возрастания: True

Массив по убыванию (Быстрая сортировка): 17 секунд 38 миллисекунд
Количество сравнений: 4401292100, перестановок: 2200840606
Проверка сортировки для Массив по убыванию (Быстрая сортировка):
Массив отсортирован в порядке возрастания: True


C:\Users\Mikhail\Documents\repos\University-Homework\Base_Programming\ЛР10\Задание_1\Задание_1\b
1.exe (процесс 19120) завершил работу с кодом 0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Парам
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

Файл Правка Формат Вид Справка

```
-99995
-99994
-99990
-99986
-99978
-99977
-99977
-99976
-99974
-99972
-99972
-99970
-99968
-99965
-99961
-99956
-99955
-99955
```



2. Сгенерировать массив из 100 купюр произвольным образом (купюры могут быть номиналом 1, 2, 5, 10, 20, 50, 100 единиц). Отсортировать массив алгоритмом сортировки подсчетами и вывести на экран.

```
static void Main()
{
    Random rand = new Random();
    int n = 100;
    int[] bills = new int[n];

    // Генерация массива с купюрами номиналом 1, 2, 5, 10, 20, 50, 100
    int[] options = { 1, 2, 5, 10, 20, 50, 100 };

    for (int i = 0; i < n; i++)
    {
        bills[i] = options[rand.Next(options.Length)];
    }

    // Вывод сгенерированного массива
    Console.WriteLine("Сгенерированный массив купюр:");
    foreach (var bill in bills)
    {
        Console.Write(bill + " ");
    }
    Console.WriteLine();

    // Сортировка подсчетами
    CountingSort(bills);

    Console.WriteLine("\nОтсортированный массив купюр:");
    foreach (var bill in bills)
    {
        Console.Write(bill + " ");
    }
}
```

```
static void CountingSort(int[] arr)
{
    // Определяем диапазон возможных номиналов
    int[] options = { 1, 2, 5, 10, 20, 50, 100 };
    int maxValue = options[options.Length - 1];

    // Массив для подсчета количества каждого номинала
    int[] count = new int[maxValue + 1];

    // Подсчитываем количество каждой купюры
    foreach (int bill in arr)
    {
        count[bill]++;
    }

    // Заполняем массив отсортированными элементами
    int index = 0;
    for (int i = 0; i < count.Length; i++)
    {
        while (count[i] > 0)
        {
            arr[index++] = i;
            count[i]--;
        }
    }
}
```

```
Консоль отладки Microsoft Visual Studio
Сгенерированный массив купюр:
50 5 50 50 10 10 100 100 20 5 20 20 2 10 20 50 2 20 5 2 2 5 50 10 5 2 2 10 1 5
5 100 1 10 10 20 10 20 10 10 2 1 2 100 20 100 1 20 2 50 5 2 1 1 50 100 5 100 1
1 5 100 100 1 2 50 2 50 10 5 5 50 50 20 50 100 10 50 5 5 50 2 50 20 100 2 10 20
 20 100 20 5 1 50 100 5 5 1 50 5

Отсортированный массив купюр:
1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 10 10 10 10 10 10 10 10 10 10 10 10 10 20 20 20 20 20 20 20 20 20 20 20
20 20 20 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 100 100 100 100 100
 100 100 100 100 100 100 100 100
C:\Users\Mikhail\Documents\repos\University-Homework\Base_Programming\ЛР10\Зада
ние_2\Задание_2\bin\Debug\net8.0\Задание_2.exe (процесс 4348) завершил работу с
 кодом 0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр
"Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остано
вке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

3. Задан граф (рис.1). Представить его в виде матрицы инцидентности и в виде связных списков. Программа должна позволять вводить номера вершин X и Y, после чего вывести путь от X к Y, найденный алгоритмом DFS, и все пути от X к Y, найденные алгоритмом BFS. Вывод программы для вершин 1 и 8:

```
DFS:
1 -> 2 -> 7 -> 8

BFS:
1 -> 3 -> 8
```

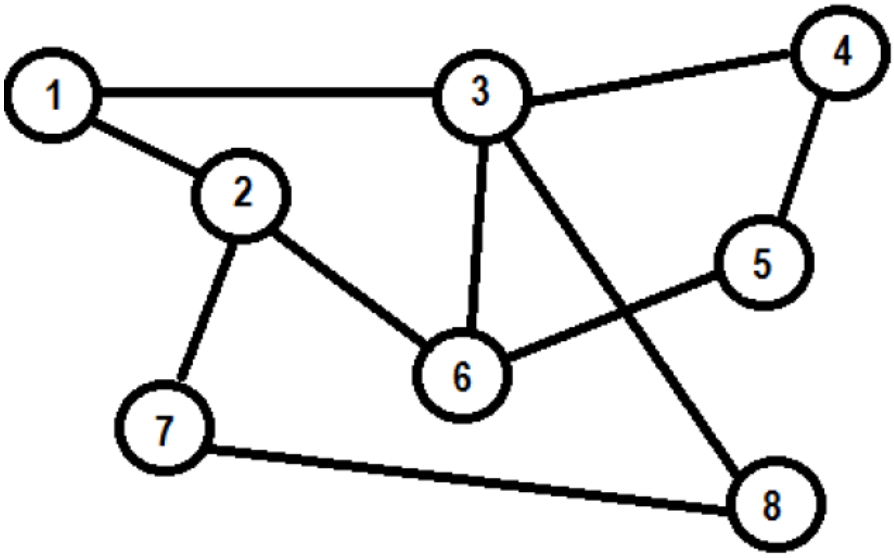



Рисунок 1 – Пример графа для задания 3

Матрица смежности:

0, 1, 1, 0, 0, 0, 0, 0

1, 0, 0, 0, 0, 1, 1, 0

1, 0, 0, 1, 0, 1, 0, 1

0, 0, 1, 0, 1, 0, 0, 0

0, 0, 0, 1, 0, 1, 0, 0

0, 1, 1, 0, 1, 0, 0, 0

0, 1, 0, 0, 0, 0, 0, 1

0, 0, 1, 0, 0, 0, 1, 0

```
Ссылок: 3
class Graph
{
    private int[,] connectionsMatrix;
    private int numberOfVertices;
    private List<int>[] connectionsList;

    Ссылок: 1
    public Graph(int[,] matrix)
    {
        numberOfVertices = matrix.GetLength(0);
        connectionsMatrix = matrix;
        connectionsList = new List<int>[numberOfVertices];

        for (int i = 0; i < numberOfVertices; i++)
        {
            connectionsList[i] = new List<int>();
            for (int j = 0; j < numberOfVertices; j++)
            {
                if (connectionsMatrix[i, j] == 1)
                {
                    connectionsList[i].Add(j);
                }
            }
        }
    }
```

```
Ссылок: 1
public void DFS(int start, int end)
{
    bool[] visited = new bool[numberOfVertices];
    List<int> path = new List<int>();
    DFSRecursive(start, end, visited, path);
}

Ссылок: 2
private void DFSRecursive(int current, int end, bool[] visited, List<int> path)
{
    visited[current] = true;
    path.Add(current + 1);

    if (current == end)
    {
        Console.WriteLine("Алгоритм DFS:");
        Console.WriteLine(string.Join(" -> ", path));
    }
    else
    {
        foreach (int neighbor in connectionsList[current])
        {
            if (!visited[neighbor])
            {
                DFSRecursive(neighbor, end, visited, path);
                path.RemoveAt(path.Count - 1);
            }
        }
    }
}
```

```
public void BFS(int start, int end)
{
    bool[] visited = new bool[numberOfVertices];
    Queue<List<int>> queue = new Queue<List<int>>();
    visited[start] = true;
    queue.Enqueue(new List<int> { start });

    List<List<int>> allPaths = new List<List<int>>();

    while (queue.Count > 0)
    {
        List<int> path = queue.Dequeue();
        int node = path[path.Count - 1];

        if (node == end)
        {
            allPaths.Add(new List<int>(path));
        }

        foreach (int neighbor in connectionsList[node])
        {
            if (!visited[neighbor])
            {
                visited[neighbor] = true;
                List<int> newPath = new List<int>(path) { neighbor };
                queue.Enqueue(newPath);
            }
        }
    }

    Console.WriteLine("Алгоритм BFS:");
    foreach (var p in allPaths)
    {
        Console.WriteLine(string.Join(" -> ", p.ConvertAll(n => n + 1)));
    }
}
```

```
public void PrintConnectionsMatrix()
{
    int edgesCount = 0;

    for (int i = 0; i < numberOfVertices; i++)
    {
        for (int j = i + 1; j < numberOfVertices; j++)
        {
            if (connectionsMatrix[i, j] == 1)
            {
                edgesCount++;
            }
        }
    }
```

```
    int[,] connectionsMatrixOutput = new int[numberOfVertices, edgesCount];
    int edgeIndex = 0;

    for (int i = 0; i < numberOfVertices; i++)
    {
        for (int j = i + 1; j < numberOfVertices; j++)
        {
            if (connectionsMatrix[i, j] == 1)
            {
                connectionsMatrixOutput[i, edgeIndex] = 1;
                connectionsMatrixOutput[j, edgeIndex] = 1;
                edgeIndex++;
            }
        }
    }

    Console.WriteLine("Матрица связей:");
    for (int i = 0; i < numberOfVertices; i++)
    {
        for (int j = 0; j < edgesCount; j++)
        {
            Console.Write(connectionsMatrixOutput[i, j] + " ");
        }
        Console.WriteLine();
    }
}
```

```
  public void PrintConnectionsList()
  {
      Console.WriteLine("Связанные вершины:");
      for (int i = 0; i < numberOfVertices; i++)
      {
          Console.Write((i + 1) + ": ");
          foreach (int neighbor in connectionsList[i])
          {
              Console.Write((neighbor + 1) + " ");
          }
          Console.WriteLine();
      }
  }
```

```
static void Main()
{
    int[,] connectionsMatrix = {
        { 0, 1, 1, 0, 0, 0, 0, 0 },
        { 1, 0, 0, 0, 0, 1, 1, 0 },
        { 1, 0, 0, 1, 0, 1, 0, 1 },
        { 0, 0, 1, 0, 1, 0, 0, 0 },
        { 0, 0, 0, 1, 0, 1, 0, 0 },
        { 0, 1, 1, 0, 1, 0, 0, 0 },
        { 0, 1, 0, 0, 0, 0, 0, 1 },
        { 0, 0, 1, 0, 0, 0, 1, 0 }
    };

    Graph graph = new Graph(connectionsMatrix);

    graph.PrintConnectionsMatrix();
    graph.PrintConnectionsList();

    int start = GetVertexInput("Введите вершину X (от 1 до 8):");

    int end = GetVertexInput("Введите вершину Y (от 1 до 8):");

    graph.DFS(start, end);
    graph.BFS(start, end);
}
```

```csharp
static int GetVertexInput(string prompt)
{
    int vertex;
    bool validInput = false;

    while (!validInput)
    {
        Console.WriteLine(prompt);
        string input = Console.ReadLine();

        if (int.TryParse(input, out vertex) && vertex >= 1 && vertex <= 8)
        {
            validInput = true;
            return vertex - 1;  // Возвращаем индекс вершины от 0 до 7
        }
        else
        {
            Console.WriteLine("Ошибка: введите число от 1 до 8.");
        }
    }

    return -1;
}
```

```
Матрица связей:
1 1 0 0 0 0 0 0 0 0
1 0 1 1 0 0 0 0 0 0
0 1 0 0 1 1 1 0 0 0
0 0 0 0 1 0 0 1 0 0
0 0 0 0 0 0 0 1 1 0
0 0 1 0 0 1 0 0 1 0
0 0 0 1 0 0 0 0 0 1
0 0 0 0 0 0 1 0 0 1
Связанные вершины:
1: 2 3
2: 1 6 7
3: 1 4 6 8
4: 3 5
5: 4 6
6: 2 3 5
7: 2 8
8: 3 7
Введите вершину X (от 1 до 8):
9
Ошибка: введите число от 1 до 8.
Введите вершину X (от 1 до 8):
a
Ошибка: введите число от 1 до 8.
Введите вершину X (от 1 до 8):
1
Введите вершину Y (от 1 до 8):
7
Алгоритм DFS:
1 -> 2 -> 6 -> 3 -> 8 -> 7
Алгоритм BFS:
1 -> 2 -> 7

C:\Users\Mikhail\Documents\repos\University-Homework\Base_Programming
3.exe (процесс 27384) завершил работу с кодом 0 (0x0).
```

4. Заданы города и двустороняя система дорог между ними в виде матрицы A, где a[i,j] = L (длина пути из города i в город j) или a[i,j]=–1, если из города i в город j прямого пути нет. Найти все города, в которые из заданного города можно добраться по суммарному пути не длиннее 200 км.

```
class Program
{
    Ссылок: 0
    static void Main()
    {
        Random rand = new Random();

        Console.WriteLine("Введите количество городов:");
        int n = int.Parse(Console.ReadLine());

        // Генерация матрицы смежности
        int[,] A = GenerateRandomMatrix(n, rand);

        Console.WriteLine("\nСгенерированная матрица смежности:");
        PrintMatrix(A);

        Console.WriteLine($"Сгенерирована матрица для {n} городов.");
        Console.WriteLine("Введите номер исходного города (от 1 до " + n + "):");
        int startCity = int.Parse(Console.ReadLine()) - 1;  // Индекс в массиве начинается с 0

        int maxDistance = 200;

        // Массив для отслеживания посещенных городов
        bool[] visited = new bool[n];

        Console.WriteLine("Города, в которые можно добраться по пути не длиннее 200 км:");

        DFS(A, startCity, 0, maxDistance, visited);
    }
```

```
Ссылок: 1
static int[,] GenerateRandomMatrix(int n, Random rand)
{
    int[,] matrix = new int[n, n];

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (i == j)
            {
                matrix[i, j] = 0;
            }
            else
            {
                // С вероятностью 75% создаем путь, иначе -1
                if (rand.NextDouble() <= 0.75)
                {
                    matrix[i, j] = rand.Next(1, 210);
                }
                else
                {
                    matrix[i, j] = -1;
                }
            }
        }
    }

    return matrix;
}
```

```
static void PrintMatrix(int[,] matrix)
{
    int n = matrix.GetLength(0);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (matrix[i, j] == -1)
                Console.Write(" -1 ");
            else
                Console.Write($"{matrix[i, j],3} ");
        }
        Console.WriteLine();
    }
}
```

```
// Функция для поиска в глубину
Ссылок: 2
static void DFS(int[,] A, int currentCity, int currentDistance, int maxDistance, bool[] visited)
{
    if (visited[currentCity]) return;

    visited[currentCity] = true;

    if (currentDistance <= maxDistance)
    {
        Console.WriteLine($"Город {currentCity + 1} (расстояние: {currentDistance} км)");
    }

    // Рекурсивно проверяем все соседние города
    for (int i = 0; i < A.GetLength(0); i++)
    {
        if (A[currentCity, i] != -1 && !visited[i])
        {
            int newDistance = currentDistance + A[currentCity, i];

            // Если сумма расстояний не превышает максимального значения, продолжаем искать
            if (newDistance <= maxDistance)
            {
                DFS(A, i, newDistance, maxDistance, visited);
            }
        }
    }
}
```

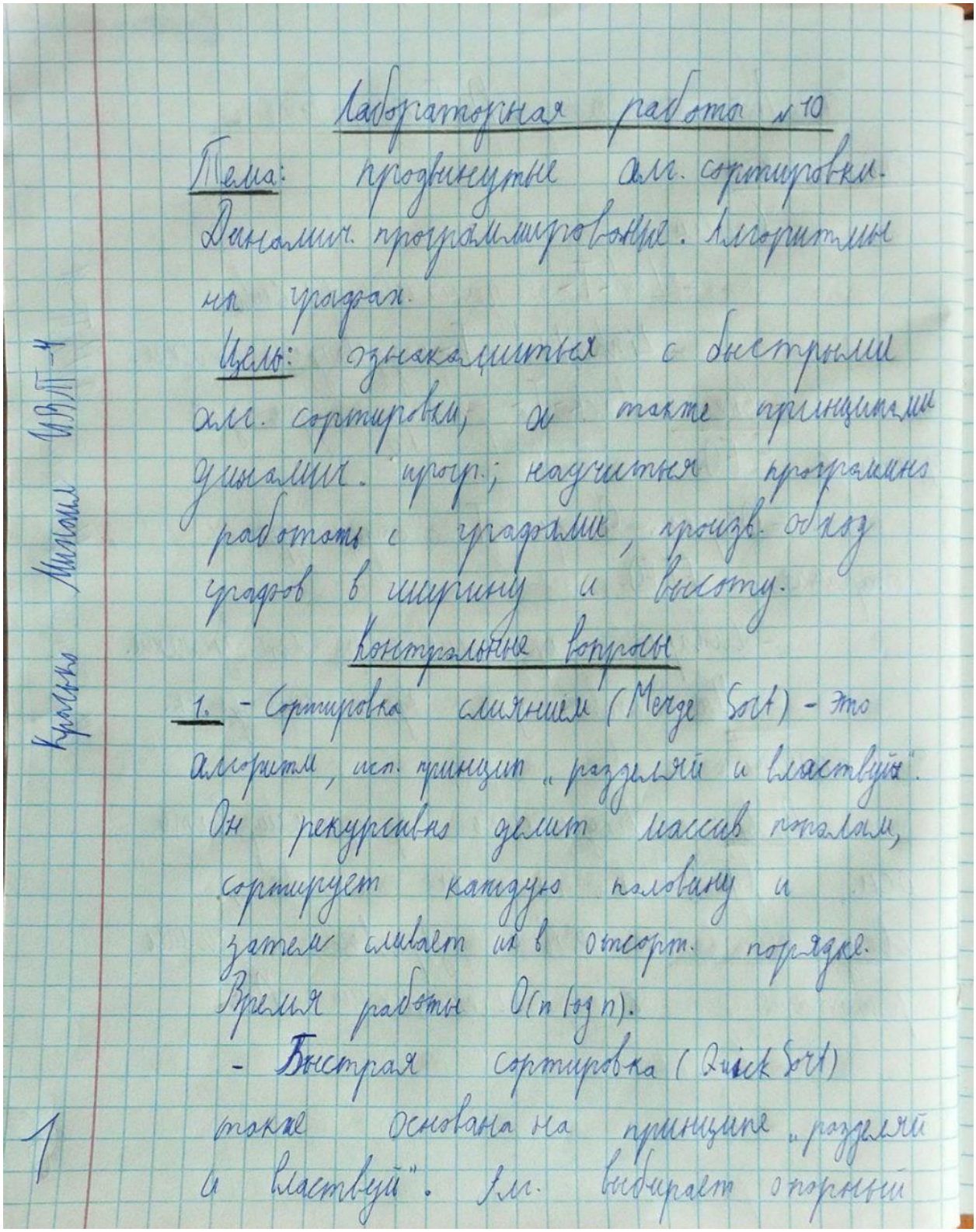
```
Введите количество городов:
6

Сгенерированная матрица смежности:
  0  -1  -1 126 155  -1
121   0 134  -1 185 183
 67   7   0 178 208  -1
115  -1  -1   0  -1  -1
 -1 205  88  69   0  -1
 97 185 119  11  21   0
Сгенерирована матрица для 6 городов.
Введите номер исходного города (от 1 до 6):
5
Города, в которые можно добраться по пути не длиннее 200 км:
Город 5 (расстояние: 0 км)
Город 3 (расстояние: 88 км)
Город 1 (расстояние: 155 км)
Город 2 (расстояние: 95 км)
Город 4 (расстояние: 69 км)
```

```
Введите количество городов:
15

Сгенерированная матрица смежности:
  0  -1 171  63  84  40 147  -1  93 190  65  72 100  -1 148
 63   0  -1 188 112  33 128  -1  29 180  -1  38  -1  -1  -1
 31 131   0 147  78  -1  13  53  -1  54  -1  46  -1 178  75
186  -1 120   0  10  49  -1  95  98  23  -1  38 173  -1 179
196  61  -1  -1   0  47 189 148 181  30  81 179  19  26  67
189  90  29 196 177   0  89 166  -1 156  -1  -1  57  28  -1
180 200  60  -1 197  -1   0 105 108 207  36  -1  -1 147  -1
 25  55 151  94 171  -1  -1   0  36 205  30  65 118  -1  -1
 38 101  74  28  58 190 180 115   0  -1  -1 179 200  13 191
204  50 160  55  24 182 141 150  15   0  -1  85 152 146 155
133 128  35   6  34  30  -1 118  91  -1   0  68   4  87 201
105  38  -1  17 172 150  -1 101  93 182  42   0  -1  -1  47
112 199 184 209  78 102  11 111 164  33  -1 196   0 127  -1
 -1  -1  63  -1  79  -1 171  85 127  -1  -1  57 109   0 149
165  -1  75  40 123 106  98  -1  19  19  81  25  -1  -1   0
Сгенерирована матрица для 15 городов.
Введите номер исходного города (от 1 до 15):
4
Города, в которые можно добраться по пути не длиннее 200 км:
Город 4 (расстояние: 0 км)
Город 1 (расстояние: 186 км)
Город 3 (расстояние: 120 км)
Город 5 (расстояние: 198 км)
Город 7 (расстояние: 133 км)
Город 11 (расстояние: 169 км)
Город 6 (расстояние: 199 км)
Город 13 (расстояние: 173 км)
Город 8 (расстояние: 173 км)
Город 10 (расстояние: 174 км)
Город 9 (расстояние: 189 км)
Город 12 (расстояние: 166 км)
Город 15 (расстояние: 195 км)
```

```
Введите количество городов:
12

Сгенерированная матрица смежности:
  0 133 187  17  89  17  62  -1 128   4  68 206
 -1   0  53  -1 111  -1  29  62 156 115 204  -1
 -1 165   0 130  67 204  97  18 168  17  45 174
 -1  -1  -1   0  -1 197   1  -1  19 122  50 172
 71 138  42 143   0  42  64  -1 178  -1  28  37
208  35  37  63   7   0  -1  -1 147  63  95 108
 -1  47  23  60  92  -1   0  -1  60 190 138 132
146  -1 153  -1 130  44 198   0 147  49 162  25
207 166  -1 204 149 153 202 113   0  27  18 127
 10  11  -1  -1 203  67  90  -1  16   0 185  -1
 72  -1  -1 179 171  -1 115  -1  66  -1   0  53
137  93  55  21  78 174  16 101  83 127  67   0
Сгенерирована матрица для 12 городов.
Введите номер исходного города (от 1 до 12):
8
Города, в которые можно добраться по пути не длиннее 200 км:
Город 8 (расстояние: 0 км)
Город 1 (расстояние: 146 км)
Город 4 (расстояние: 163 км)
Город 7 (расстояние: 164 км)
Город 3 (расстояние: 187 км)
Город 9 (расстояние: 182 км)
Город 11 (расстояние: 200 км)
Город 6 (расстояние: 163 км)
Город 2 (расстояние: 198 км)
Город 5 (расстояние: 170 км)
Город 10 (расстояние: 150 км)
Город 12 (расстояние: 25 км)

C:\Users\Mikhail\Documents\repos\University-Homework\Base_Programming\ЛР10\3
```

Контрольные вопросы на следующей странице

Контрольные вопросы



Лабораторная работа №10

Тема: продвинутые алг. сортировки. Динамич. программирование. Алгоритмы на графах.

Цель: ознакомиться с быстрыми алг. сортировки, а также принципами динамич. прогр.; научиться программно работать с графами, произв. обход графов в ширину и высоту.

Контрольные вопросы

1. - Сортировка слиянием (Merge Sort) - это алгоритм, исп. принцип „разделяй и властвуй". Он рекурсивно делит массив пополам, сортирует каждую половину и затем сливает их в отсорт. порядке. Время работы O(n log n).

- Быстрая сортировка (Quick Sort) также основана на принципе „разделяй и властвуй". Эл. выбирает опорный

Кравцов Михаил ИВТ-4

1

элемент и делит массив на 2 подмассива: элементы, меньшие опорного, и элементы, большие опорного. Средние подмассивы, Далее рекурсивно создаются время работы $O(n \log n)$.

2. Алгоритм ~~пр~~ пирамидальной сортировки (Heap Sort) - это алг. сортировки, к-й строит структуру данных под названием „кучу". Куча предст. собой полное бинарное дерево, где каждый родительский элемент не меньше своих дочерних элементов. Для сортировки строится куча из элементов массива, затем на каждом шаге извлекается макс. (или мин.) элемент из кучи и восст. св-ва кучи. Время работы алг. - $O(n \log n)$.

3 Алг. сортирует эл. путём подсчёта их частоты. Создаётся массив подсчёта

для каждого возможного значения, затем накапливаются результаты и сортируются в итоговый массив. Это эффективно для целых чисел с огр. диапазоном значений. Время работы: $O(n+k)$, где $n$ - кол-во эл., $k$ - диапазон значений ~~элементов~~.

4. Динамич. програм. исп. для решения задач, к-е можно разбить на подзадачи. Результаты подзадач сохраняются, чтобы избежать их повторного вычисл. Алгоритм включает разбиение задачи на подзадачи, решение и сохранение промежуточных результатов для построения решения исходной задачи.

5. - Матрица смежности - подходит для плотных графов, где хранится инф. о наличии ребер между вершинами

Краснов Михаил ИВТ-4

3

- Список смежности - эффективен для разреженных графов, хранит для каждой вершины список её соседей.

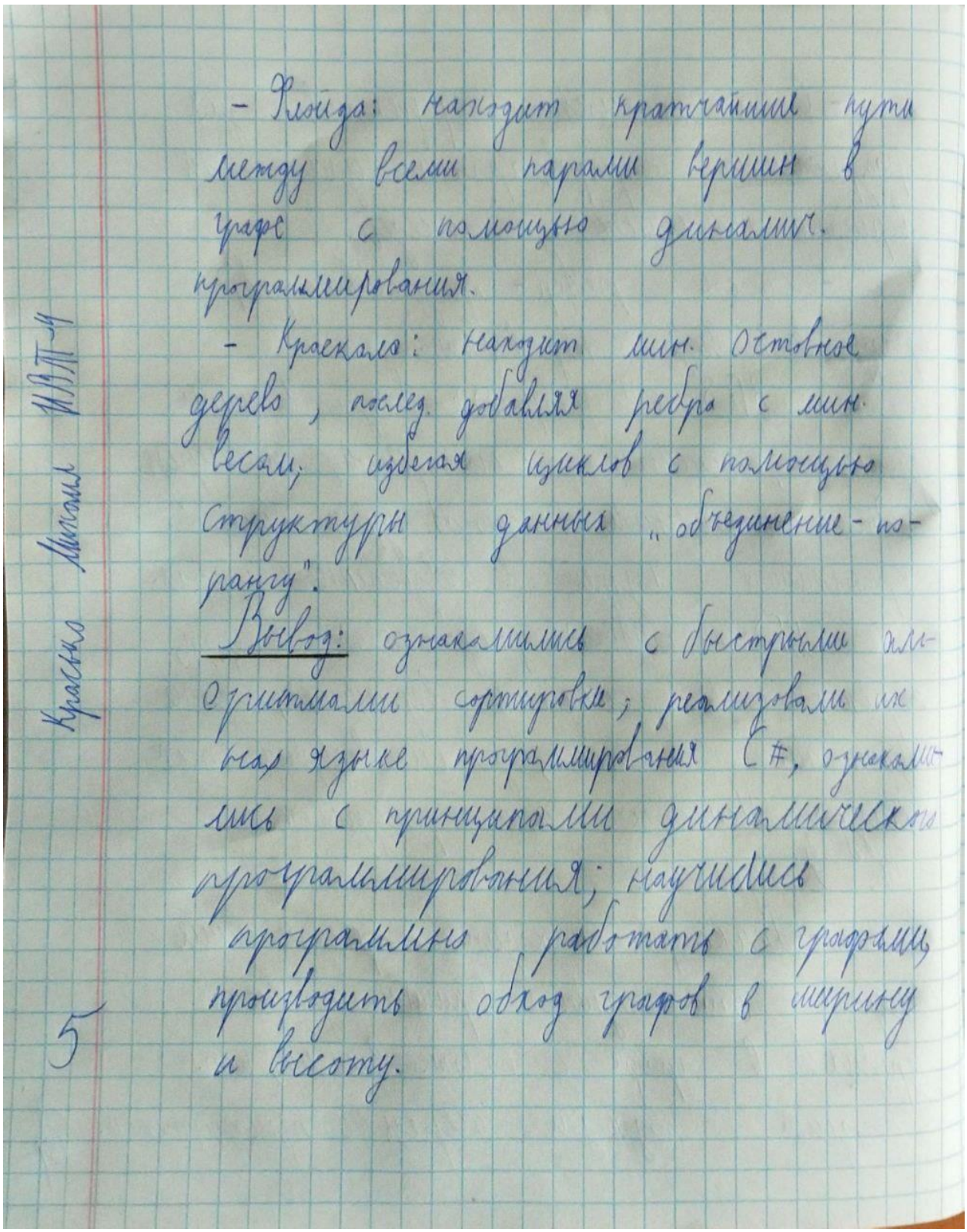
- список ребер - предст. все рёбра графа в виде пар вершин.

6. BFS (по ширине): обходит вершины уровнями, начиная с начальной, исп. очередь. Это идеальный способ для поиска кратчайших путей в невзвешенных графах.

DFS (по глубине): углубляется в граф, переходя к соседним вершинам, пока не достигнет конца, затем возвращается назад. Исп. стек или рекурсию.

7. Алгоритмы:

- Дейкстры: находит кратчайшие пути от одной вершины до всех других в графе с полож. весами, исп. жадный подход и очередь с приоритетом.




**Вывод:** ознакомились с быстрыми алгоритмами сортировки, реализовали их на языке программирования C#, изучили базовые принципы динамического программирования; научились программно работать с графами, производить обход графов в ширину и высоту.