1. Постановка задачи

Изучить и реализовать алгоритмы на графах в соответствии с вариантом:

1) Метод дейкстры;
2) Ядро графа (орграф);
3) Нахождение ярусно-параллельной формы графа.

2. Метод Дейкстры

2.1. Описание алгоритма

Прежде чем описывать алгоритм, следует условиться о том, что собой представляет маршрут.

Маршрут в графе — это чередующаяся последовательность вершин и рёбер $v_0, e_1, v_1, e_2, v_2, \dots, e_k, v_k$, в которой любые два соседних элемента инцидентны. Если $v_0 = v_k$, то маршрут *замкнут*, иначе *открыт*.

Алгоритм дейкстры используется для нахождения наименьших путей из одной вершины во все остальные. Он применяется как для неориентированного графа, так и для орграфа, где:

1) Неориентированный граф – граф, в котором ребра не имеют направлений, и по ним можно двигаться в обе стороны.
2) Орграф, соответственно – граф, в которой ребра имеют направления.

Алгоритм имеет 3 стадии выполнения. Давайте рассмотрим их на примере данного графа (рис. 2.1.1):

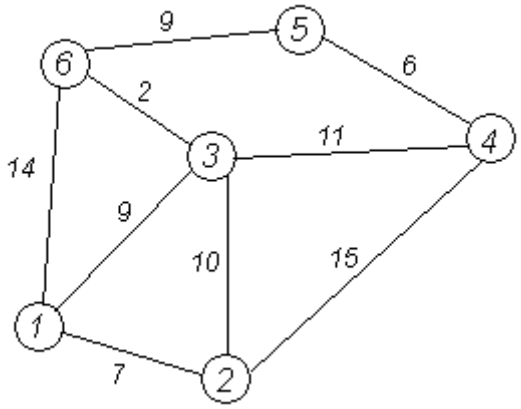

Рисунок 2.1.1 – пример графа

Итак, перая стадия – **инициализация**.

Выбирается одна из вершин (условимся взять первую), ее метка ставится равной 0, метки всех остальных вершин – бесконечность. Все вершины графа помечаются как непосещенные.

Метка – кратчайший путь от первой вершины до данной. Логично, что таковая первой вершины равна 0, так как, чтобы прийти из себя в себя же, нужно пройти расстояние равное 0. Метки остальных вершин – бесконечность, потому чтоь пока мы не знаем точных расстояний.

На нашем графе стадия инициализации будет выглядеть так (рис 2.1.2)

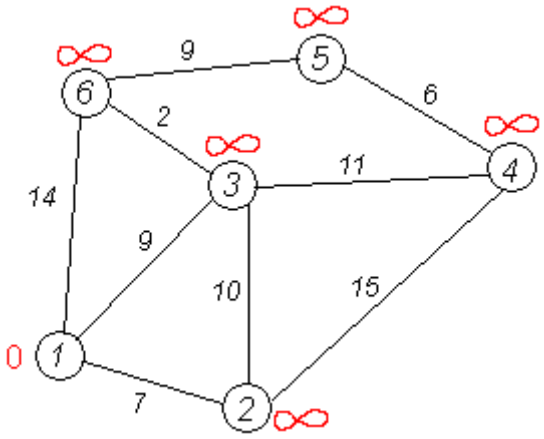

Рисунок 2.1.2 – граф после инициализации

Вторая стадия – **шаг алгоритма**.

Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещённых вершин выбирается вершина ***u***, имеющая минимальную метку. Мы рассматриваем всевозможные маршруты, в которых ***u*** является предпоследним пунктом. Вершины, в которые ведут рёбра из ***u***, назовём *соседями* этой вершины. Для каждого соседа вершины ***u***, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки ***u*** и длины ребра, соединяющего ***u*** с этим соседом. Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину ***u*** как посещённую и повторим шаг алгоритма.

Проведем шаг алгоритма для нашего графа. При поиске вершины с минимальной меткой наш выбор упадет на начальную вершину. Она помечена 0, а все остальные бесконечностью (рис 2.1.3).

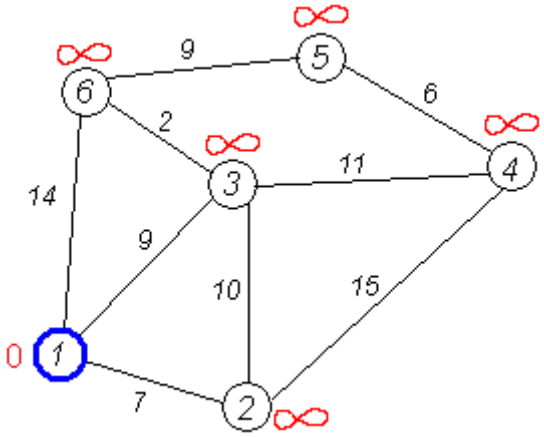

Рисунок 2.1.3 – выбор вершины с минимальной меткой

Первый по очереди сосед вершины 1 — вершина 2, потому что длина пути до неё минимальна. Длина пути в неё через вершину 1 равна сумме значения метки вершины 1 и длины ребра, идущего из 1-й в 2-ю, то есть 0 + 7 = 7. Это меньше текущей метки вершины 2, бесконечности, поэтому новая метка 2-й вершины равна 7 (рис 2.1.4).

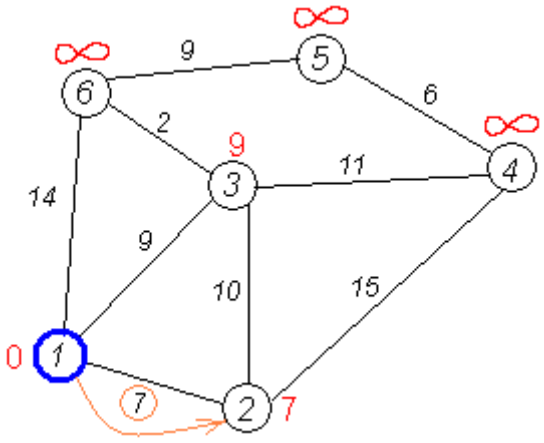

Рисунок 2.1.4

Аналогичную операцию проделываем с двумя другими соседями 1-й вершины — 3-й и 6-й (Рисунок 2.1.5).

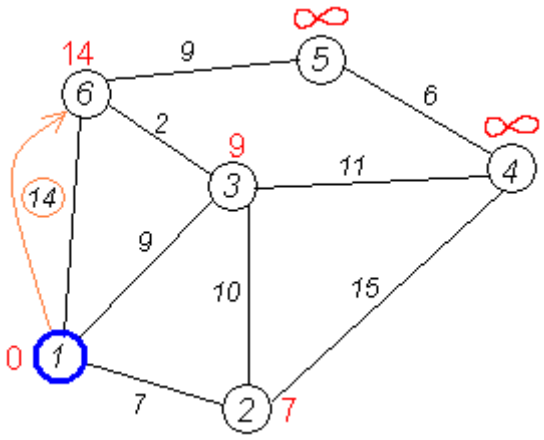

Рисунок 2.1.5

Все соседи вершины 1 проверены. Текущее минимальное расстояние до вершины 1 считается окончательным и пересмотру не подлежит. Вычеркнем её из графа, чтобы отметить, что эта вершина посещена (рис. 2.1.6).

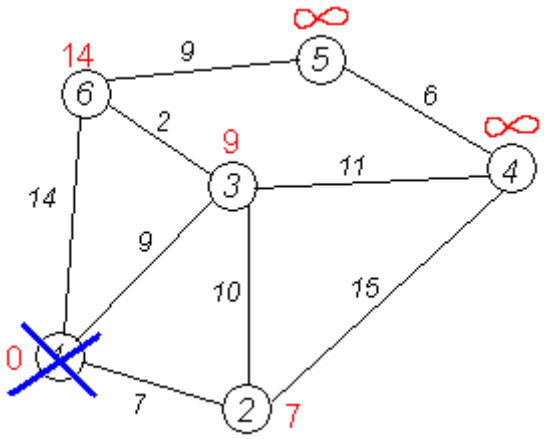

Рисунок 2.1.6

На данном моменте шаг алгоритма закончен. Далее данная последовательность действий повторяется, пока не буду посещены все вершины.

Наш граф к концу преобразований будет выглядеть так:

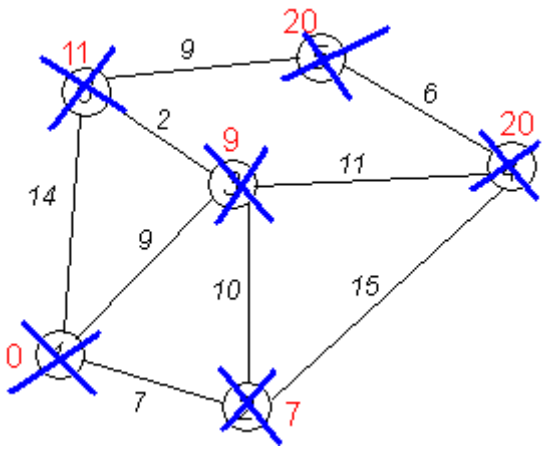

Рисунок 2.1.7 – граф после применения к нему алгоритма Дейкстры

И, наконец, 3 стадия – **обратный ход.**

На данной стадии мы следуем от конечной вершины до начальной в поисках минимального пути от последней вершины до первой. Делается это посредством вычитания из метки вершины весов ребер инцидентных данной. Частное будет равно метке одной из смежных вершин. Именно из нее и был осоущствлен преход.

Для нашего графа алгоритм обратного хода будет таковым:

Конечная вершина (№ 5) имеет метку, равную 20. Имеется два инцидентных ей ребра: 5 – 6 с весом 9 и 5 – 4 с весом 6.

20 – 6 = 14 != 20 (не подходит)

20 – 9 = 11 (подходит)

Аналогично получаем последовательность 5-6-3-1. Алгоритм закончил работу.

2.2. Описание практического использования алгоритма

Задачи поиска кратчайшего пути – огромный пласт задач, к решению которых сводятся тысячи разных реальных задач.

Нахождения минимального пути в GPS навигаторе, максимально короткий и быстрый путь из точки А в точку Б в компьютерных играх, составление максимально бюджетного пути перелетов и т.д.

Так как алгоритм Дийкстры не применим к графам с отрицательными весами ребер, с помощью него обычно решаются навигационные и тому-подобные задачи.

Яркий примеры:

Вариант 1. Дана сеть автомобильных дорог, соединяющих города Московской области. Некоторые дороги односторонние. Найти кратчайшие пути от города Москвы до каждого города области (если двигаться можно только по дорогам).

Вариант 2. Имеется некоторое количество авиарейсов между городами мира, для каждого известна стоимость. Стоимость перелёта из A в B может быть не равна стоимости перелёта из B в A. Найти маршрут минимальной стоимости (возможно, с пересадками) от Копенгагена до Барнаула.

3. Поиск ядер орграфа

3.1 Описание алгоритма

Приведём несколько важных определений.

Внутренне устойчивым множеством V вершин графа G называется такое множество его вершин, что никакие две вершины v_i и v_j, входящие в это множество, не являются смежными.

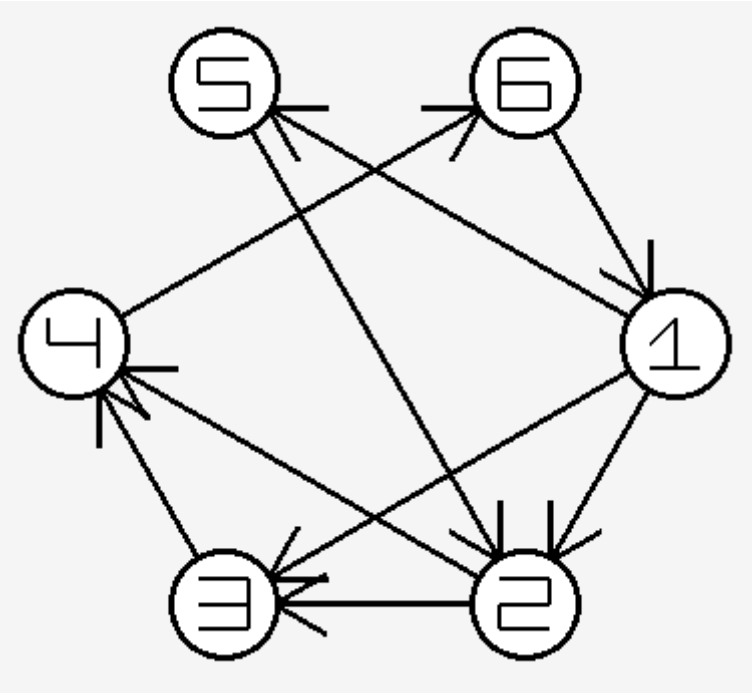

Рисунок 3.1.1 – пример графа

На рисунке 3.1.1 представлен граф, для которого существует четыре подмножества внутренне устойчивых вершин:

- {4, 5};
- {3, 5, 6};
- {2, 6};
- {1, 4}.

Внешне устойчивым множеством V вершин графа G называется такое множество его вершин, что для каждой вершины v_i, не входящей в множество V, существует дуга, исходящая из некоторой вершины множества V и заходящая в вершину v_i.

На рисунке 3.1.1 представлен граф, для которого существует шесть подмножеств внешне устойчивых вершин:

- {1, 2, 4};
- {1, 4, 5};
- {2, 3, 6};
- {2, 4, 6};
- {3, 5, 6};
- {4, 5, 6}.

Ядро – это множество одновременно и внутренне, и внешне устойчивых вершин. В графе на рисунке 3.1.1 ядром является множество вершин {3, 5, 6}.

В принципе граф может обладать несколькими ядрами или вообще не иметь ядра.

Чтобы найти ядра орграфа, нужно, соответственно, найти вершины внутренней и внешней устойчивости. Для этого воспользуемся матрицей смежности, относящейся к графу на рисунке 3.1.1.

	1	2	3	4	5	6
1	0	1	1	0	1	0
2	0	0	1	1	0	0
3	0	0	0	1	0	0
4	0	0	0	0	0	1

5	0	1	0	0	0	0
6	1	0	0	0	0	0

Рисунок 3.1.2 – матрица смежности для графа на рисунке 3.1.1

Для поиска вершин внутренней устойчивости составим логическое выражение, которое получается таким образом: из каждой строки выписывается дизъюнкт вершины v_i, соответствующей данной строке, и конъюнкции вершин, в которые входит дуга, исходящая из вершины v_i. Например, для первой строки дизъюнкт будет таким: 1 ∨ 2&3&5. Все дизъюнкты входят в конъюнкцию.

В итоге получим следующее выражение:

(1 ∨ 2&3&5) & (2 ∨ 3&4) &(3 ∨ 4) &(4 ∨ 6) &(5 ∨ 2)&(6 ∨ 1).

Используя законы алгебры логики, оно будет преобразовано к такому виду:

1&2&4 ∨ 1&3&4 &5 ∨ 1&2&3 &6 ∨ 2&3&5 &6.

В соответствии с каждым конъюнктом выписываем все вершины, которые в него не вошли. Так, в конъюнкте 1&2&4 не достаёт вершин 3, 5 и 6. Тогда они войдут в одно из подмножеств вершин внутренней устойчивости.

В конце концов, получится четыре подмножества, сколько и должно быть:

- {4, 5};
- {3, 5, 6};
- {2, 6};
- {1, 4}.

Для того чтобы найти вершины внешней устойчивости в главную диагональ матрицы смежности помещаются единицы (рисунок 3.1.3):

	1	2	3	4	5	6
1	1	1	1	0	1	0
2	0	1	1	1	0	0
3	0	0	1	1	0	0
4	0	0	0	1	0	1

5	0	1	0	0	1	0
6	1	0	0	0	0	1

Рисунок 3.1.3 – дополненная матрица смежности для матрицы смежности на рисунке 3.1.2

Логическое выражение будет составляться немного иначе. Всё также проходимся построчно, но в дизъюнкт теперь входят все вершины, для которых в соответствующей ячейке стоит 1. Например, дизъюнкт для первой строчки будет выглядеть так: 1∨2∨3∨5.

Путём перемножения дизъюнктов формируется такое выражение:

(1∨2∨3∨5)&(2∨3∨4)&(3 ∨ 4)&(4 ∨ 6)&(2 ∨ 5)&(1∨6).

Затем, применив законы алгебры логики, оно преобразуется в:

1&2&4 ∨ 1&4 &5 ∨ 2&4 &6 ∨ 4&5&6 ∨ 2&3&6 ∨ 3&5&6.

В отличие от предыдущего поиска выписывать не достающие вершины не нужно, все остаётся как есть.

Ответ – 6 подмножеств вершин внешней устойчивости:

- {1, 2, 4};
- {1, 4, 5};
- {2, 3, 6};
- {2, 4, 6};
- {3, 5, 6};
- {4, 5, 6}.

Ядро – это общее в первом и втором списках, а именно - {3, 5, 6}.

3.2. Описание практического использования алгоритма

Алгоритм нахождения ядра может применться в архитектурных задачах.

Одним из методов, имеющих большой потенциал для применения в архитектурном проектировании, является метод анализа архитектурной композиции путем сопоставления с ядром графа, который предварительно создается по предполагаемым связям внутри объекта. В случае несовпадения графа с его ядром производится корректировка связей. Метод был разработан

Н. М. Зубовым на кафедре ТСАП и ВМ Свердловского архитектурного института.

При анализе памятников архитектуры с помощью графов, поставленных в соответствие, Н.М. Зубов пришел к выводу, что наиболее четкая композиционная структура соответствует графу, содержащему ядро и совпадающему с ним. Возможность применения данного метода двояка: если мы анализируем уже существующий объект, сопоставление с ядром графа помогает создать наиболее правильную модель его восприятия. Если же мы проектируем объект, то данный метод может выступать как средство нахождения более гармоничного и цельного проектного решения. Предлагаемая модельная задача анализа функциональных связей внутри объекта относится ко второму случаю.

4. Нахождение ярусно-параллельной формы ориентированного ациклического графа

4.1. Описание алгоритма

Ярусно-параллельная форма графа (ЯПФ) — деление вершин ориентированного ациклического графа на перенумерованные подмножества V_i такие, что, если дуга e идет от вершины $v_1 \in V_j$ к вершине $v_2 \in V_k$, то обязательно j<k.

Каждое из множеств V_i называется *ярусом* ЯПФ, i — его *номером*, количество вершин $|V_i|$ в ярусе — его *шириной*. Количество ярусов в ЯПФ называется её *высотой*, а максимальная ширина её ярусов — *шириной* ЯПФ.

Вершины первого яруса не имеют предшествующих вершин, вершины последнего – не имеют последующих. Вершины любого яруса, кроме последнего, не имеют предшествующих вершин в следующем ярусе. Должна быть хотя бы одна заходящая дуга из предыдущего яруса. Вершины одного и того же яруса дугами не соединяются.

ЯПФ – иначе говоря, один из способов представления графа, но только ориентированного ациклического. В противном случае такой граф представить невозможно, так как найдутся такие вершины, которые нельзя однозначно определить в определенные ярусы.

На рисунке 4.1.1 представлен пример графа, который обладает своей ЯПФ, показанной на рисунке 4.1.2.

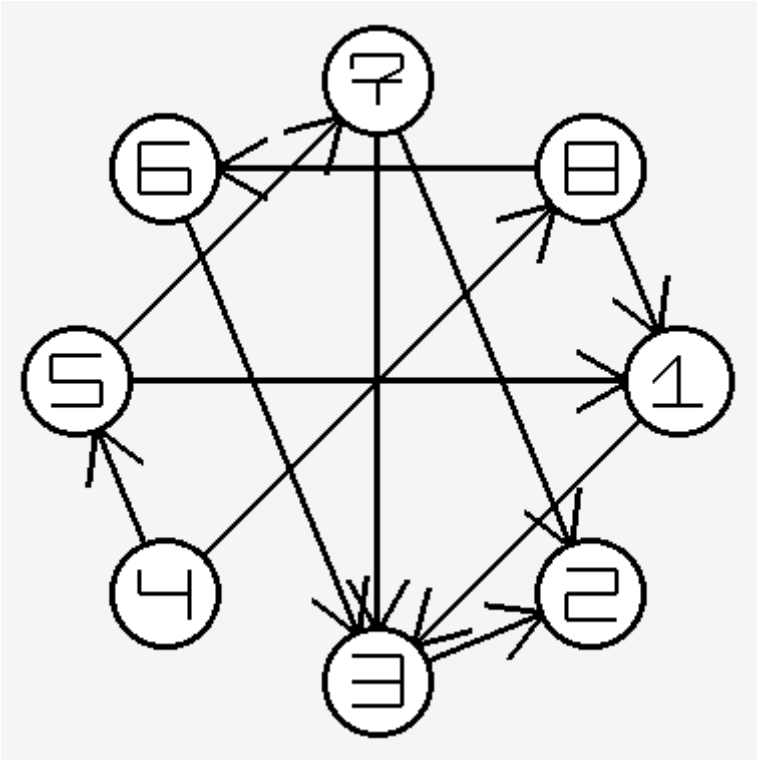

Рисунок 4.1.1 – пример графа

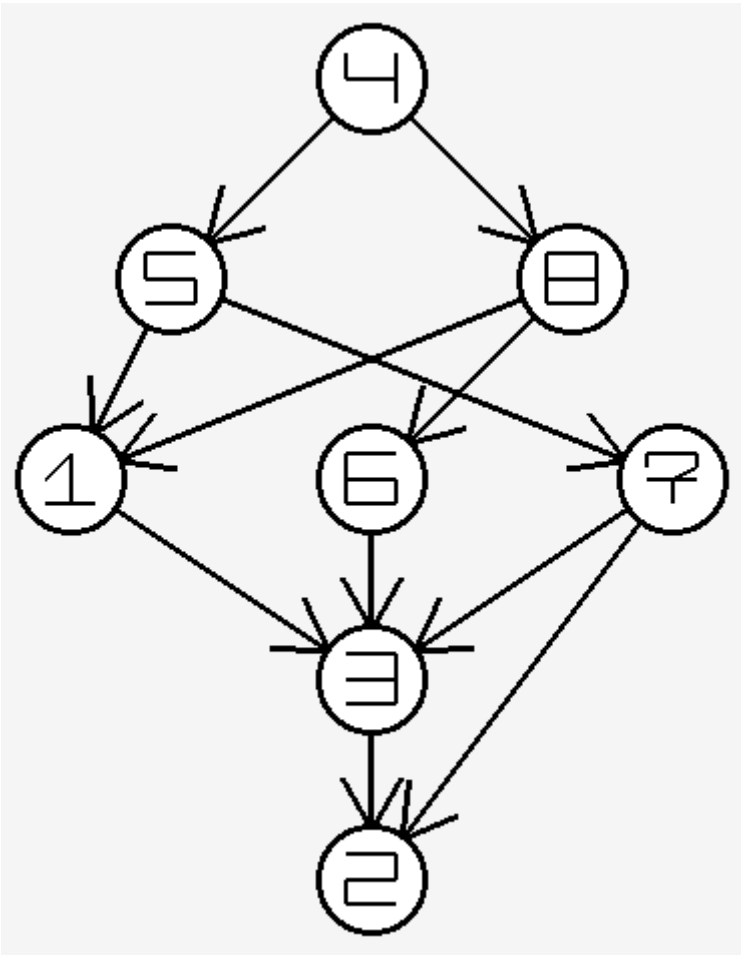

Рисунок 4.1.2 – ЯПФ графа на рисунке 4.1.1

Для нахождения ЯПФ работаем с матрицей смежности исходного графа. В данном случае возьмем для примера граф на рисунке 4.1.1. Внимание на рисунок 4.1.3:

	1	2	3	4	5	6	7	8
1	0	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0
4	0	0	0	0	1	0	0	1
5	1	0	0	0	0	0	1	0
6	0	0	1	0	0	0	0	0
7	0	1	1	0	0	0	0	0
8	1	0	0	0	0	1	0	0

Рисунок 4.1.3 – матрица смежности графа на рисунке 4.1.1

Алгоритм нахождения ЯПФ следующий:

1) ищем столбцы, заполненные только нулями;
2) помещаем вершины найденных столбцов в новый ярус;
3) обнуляем строки, соответствующие найденным столбцам (если столбец относится к вершине v_i, обнуляется строка, относящаяся к вершине v_i);
4) возвращаемся к пункту 1, пока не будут посещены все столбцы матрицы.

Поэтапно обнуление матрицы происходит следующим образом:

1) нулевой столбец только у вершины 4, помещаем вершину 4 в первый ярус, обнуляем соответствующую строку (рисунок 4.1.4):

	1	2	3	4	5	6	7	8
1	0	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	1	0
6	0	0	1	0	0	0	0	0
7	0	1	1	0	0	0	0	0
8	1	0	0	0	0	1	0	0

Рисунок 4.1.4 – шаг 1

2) появились новые нулевые столбцы – у вершин 5 и 8, помещаем их во второй ярус, обнуляем соответствующие строки (рисунок 4.1.5):

	1	2	3	4	5	6	7	8
1	0	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	1	0	0	0	0	0
7	0	1	1	0	0	0	0	0
8	0	0	0	0	0	0	0	0

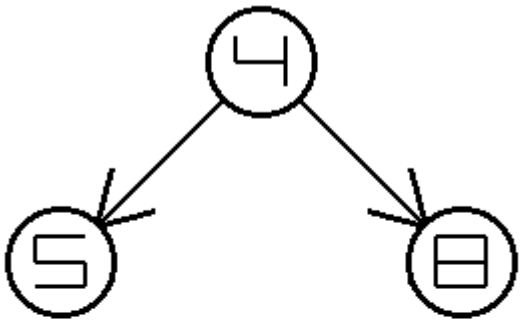

Рисунок 4.1.5 – шаг 2

3) появились новые нулевые столбцы – у вершин 1, 6 и 7, помещаем их в третий ярус, обнуляем соответствующие строки (рисунок 4.1.6):

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0

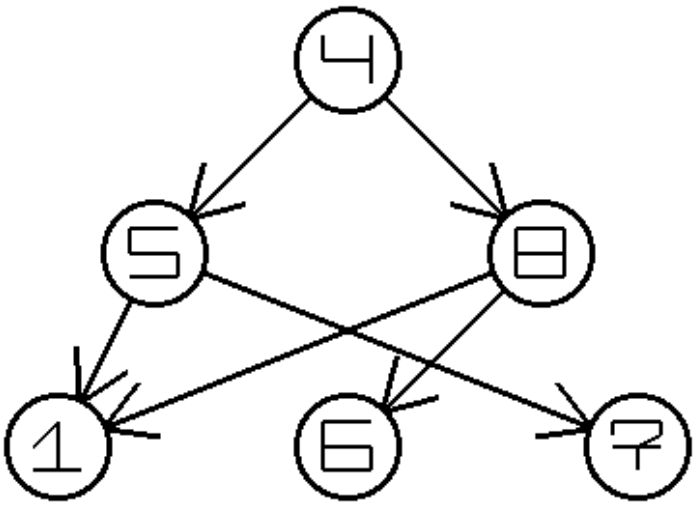

Рисунок 4.1.6 – шаг 3

4) появился новый нулевой столбец – у вершины 3, помещаем ее в четвёртый ярус, обнуляем соответствующую строку (рисунок 4.1.7):

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0

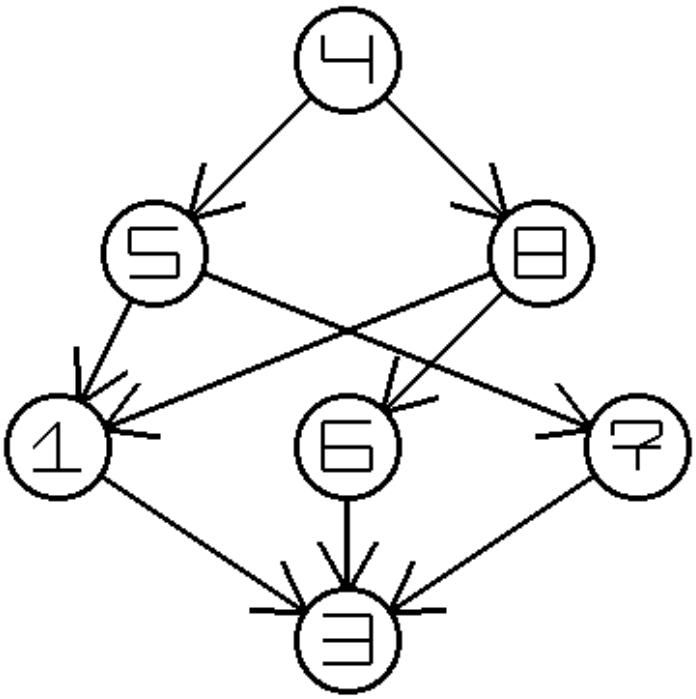

Рисунок 4.1.7 – шаг 4

5) появился новый нулевой столбец – у вершины 2, помещаем ее в последний пятый ярус, соответствующая строка уже заполнена только нулями (рисунок 4.1.8):

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0

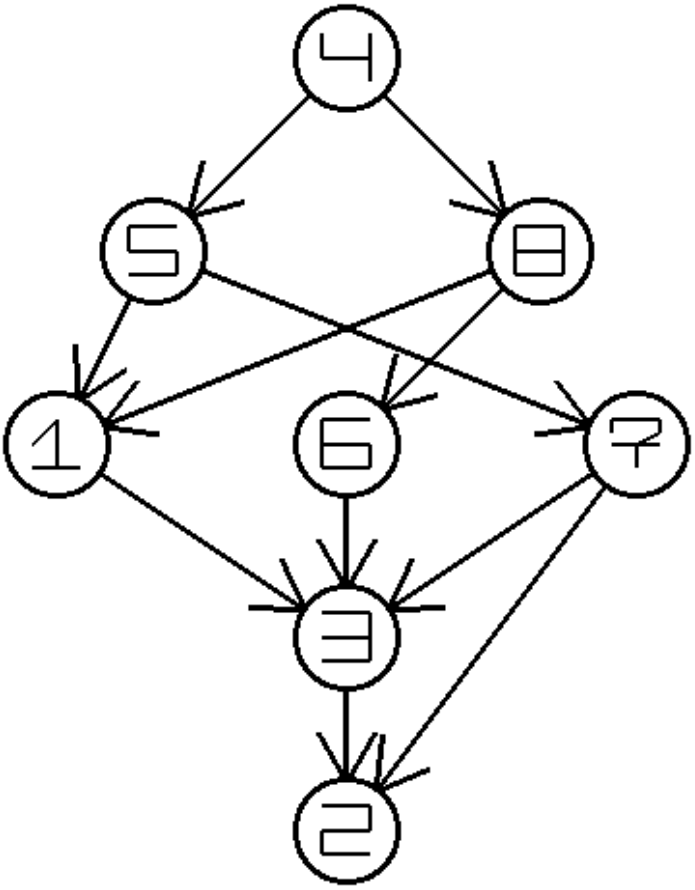

Рисунок 4.1.8 – шаг 5

Ярусно-параллельная форма найдена.

4.2. Описание практического использования алгоритма

Предположим, имеется неорганизованный набор действий с известными первыми и конечными действиями, а также промежуточными, которые обязаны истекать из каких-либо предыдущих, то есть им нужны данные, которые обрабатываются другими действиями до них. ЯПФ помогает организовать этот набор для упрощения понимания, условившись, что действия – это вершины графа, а переход из одного действия в другое – это дуга.

Кроме того, Задача приведения к ярусно-параллельной форме графа так же находит своё применение при построении системы управления в организации (кто кем управляет).

Генеалогическое древо человека – также ЯПФ графа.

5. Обоснование способов представления графа, данных, результата

5.1 Представление графа и результата

Программа реализована, как консольное приложение.

В программе все графы представлены в виде матриц. Их тип будет варьироваться в зависимости от задачи. Для алгоритма Дейкстры применяется матрица весов, для ядра и ярусно-параллельной формы – матрица смежности. Также, в зависимости от типа графа (ориентированный/неориентированный) алгоритм заполнения матриц будет упрощаться, минимизируя неудобства ввода через консоль.

5.1 Представление графа и результата

Так как все 3 алгоритма решают совершенно разные задачи, я посчитал целесообразным реализовать их, как 3 независимых проекта.

Принцип построения программ схож. Сначала задается число вершин и всем им задаются имена. Это было сделано для того, чтобы не привязывать вершины к индексам матрицы. То есть пользователь сам может задать название каждой вершины.

Затем строится матрица и по ней реализовывается алгоритм.

Вывод включает в себя ответ поставленной задачи, а также исходную матрицу.

6. Блок-схемы

6.1. Алгоритм Дейкстры

Основными функциями реализации алгоритма Дейкстры являются, непосредственно, шаг алгоритма (рис 6.1.1) и обратный проход (рис 6.1.2).

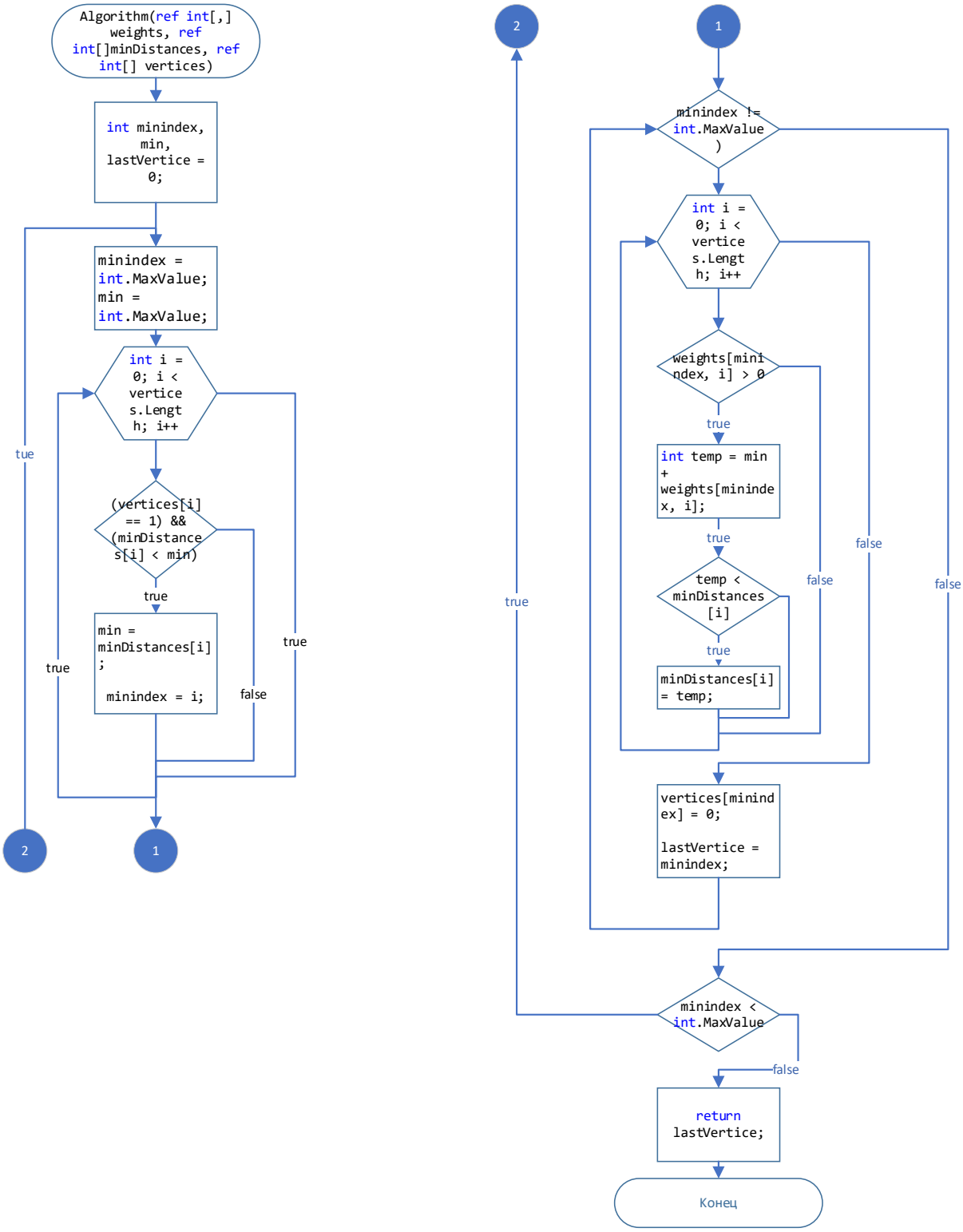

Рисунок 6.1.1 – функция реализующая шаг алгоритма Дейкстры

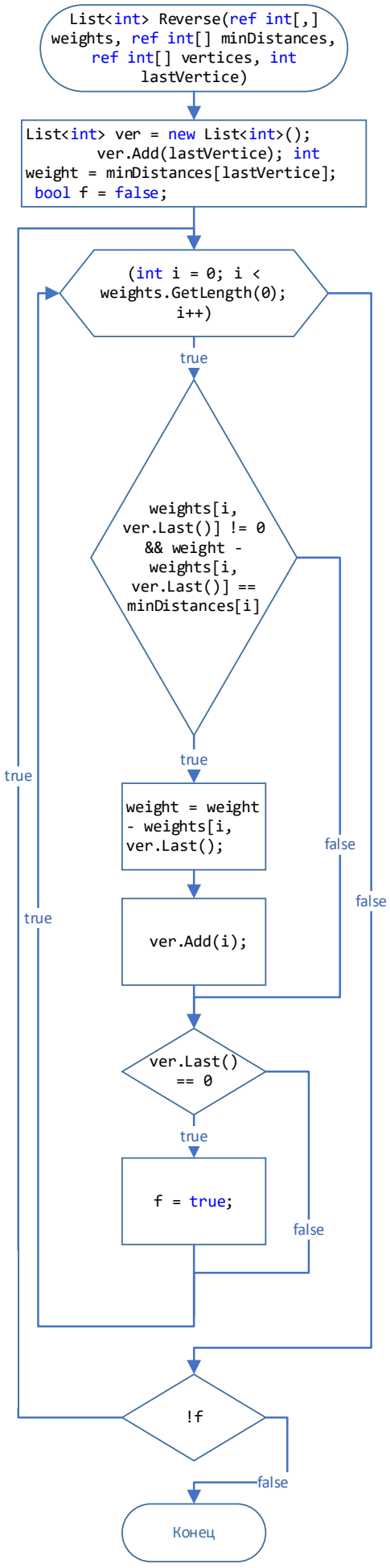

Рисунок 6.1.2 – обратный проход алгоритма Дейкстры

6.2. Нахождение ядер орграфа

Основой реализации алгоритма нахождения ядра графа является, нахождение ДНФ (рис 6.2.1)

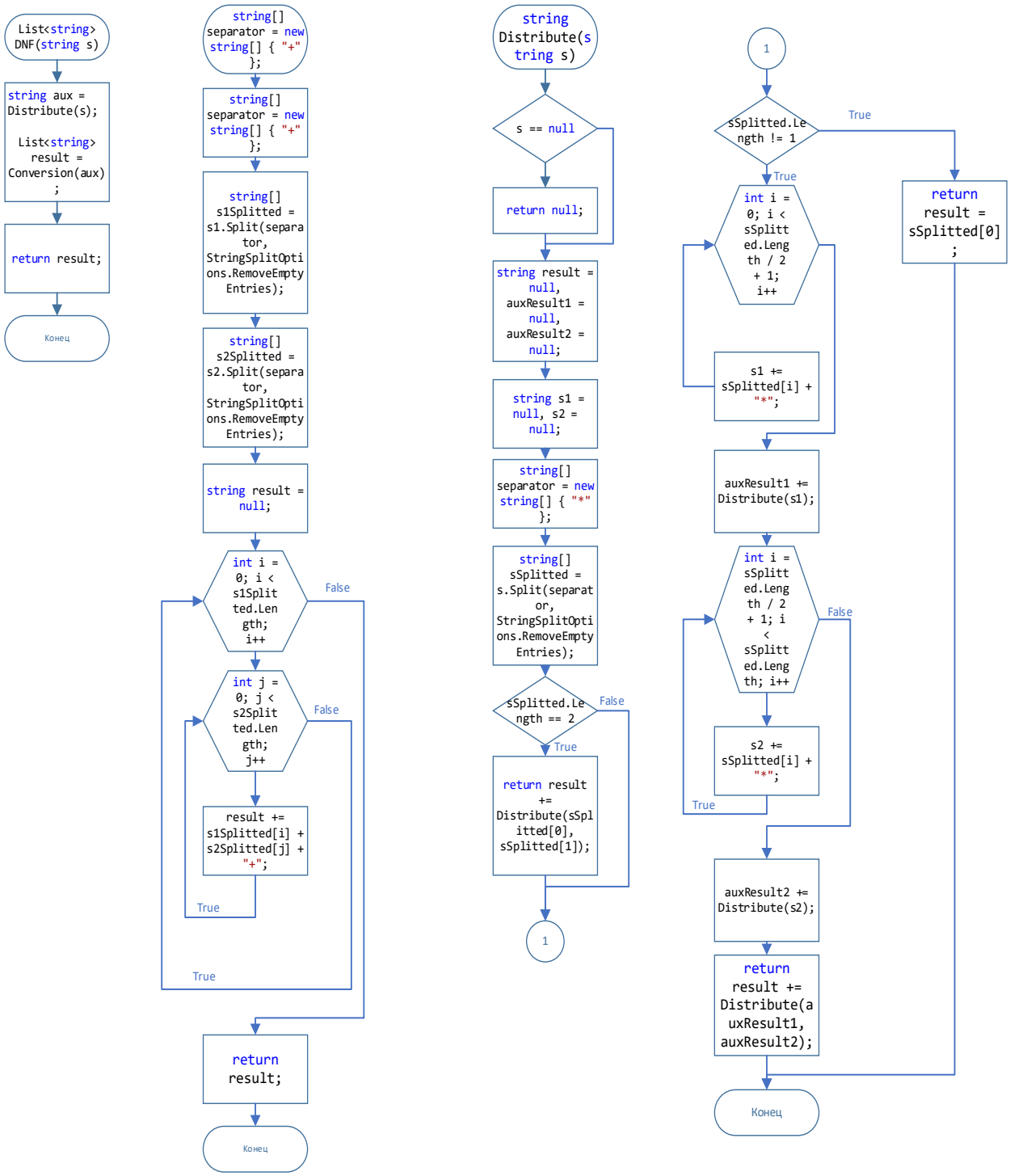

Рисунок 6.2.1 – функции нахождения ДНФ

6.3. Нахождение ЯПФ

Основой реализации алгоритма нахождения ЯПФ является, нахождение непосредственно функция нахождения ЯПФ (рис 6.3.1)

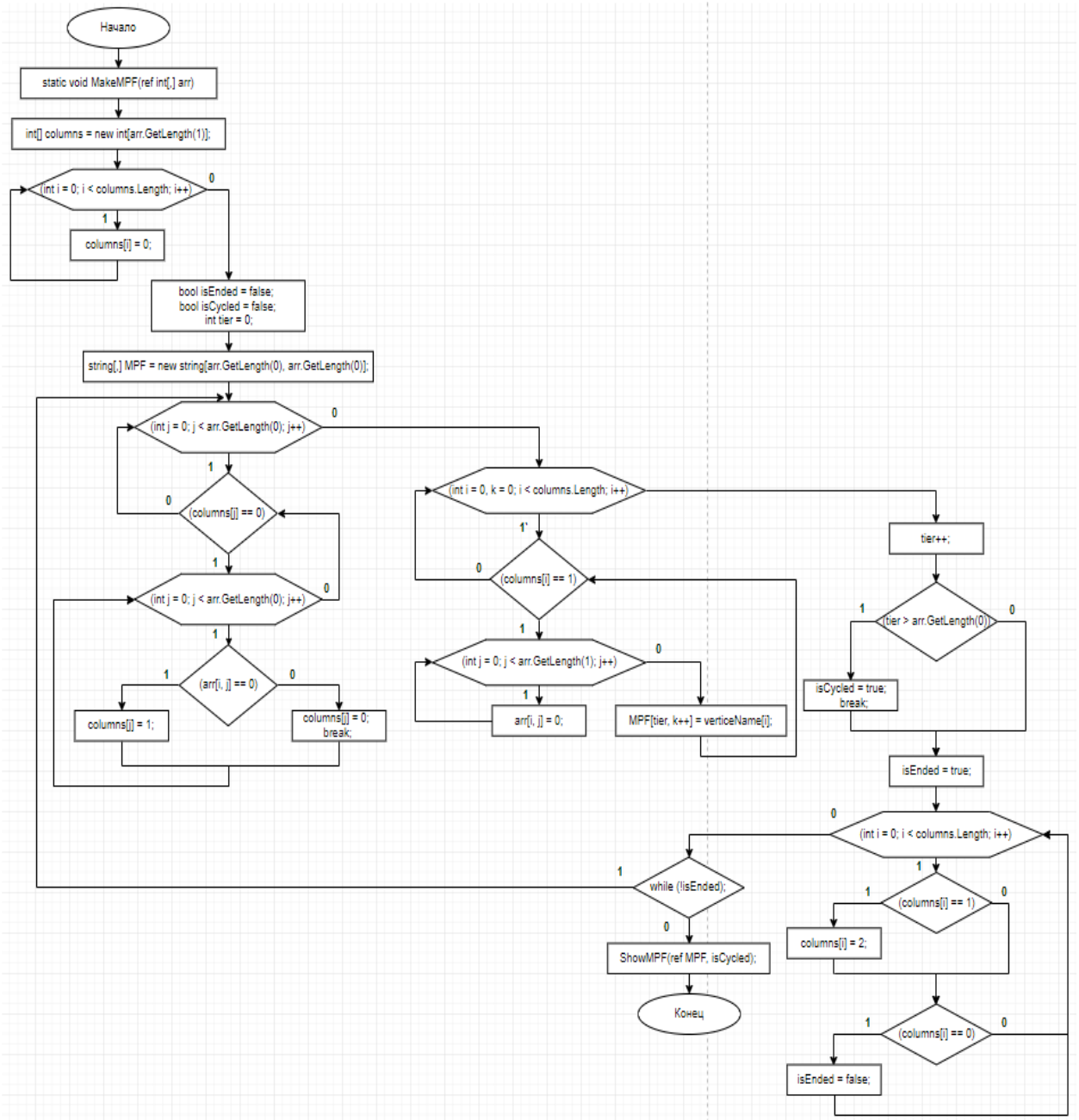

Рисунок 6.3.1 – функция нахождения ЯПФ

Функция нахождения ядер орграфа представлена на рисунке 6.3.1.

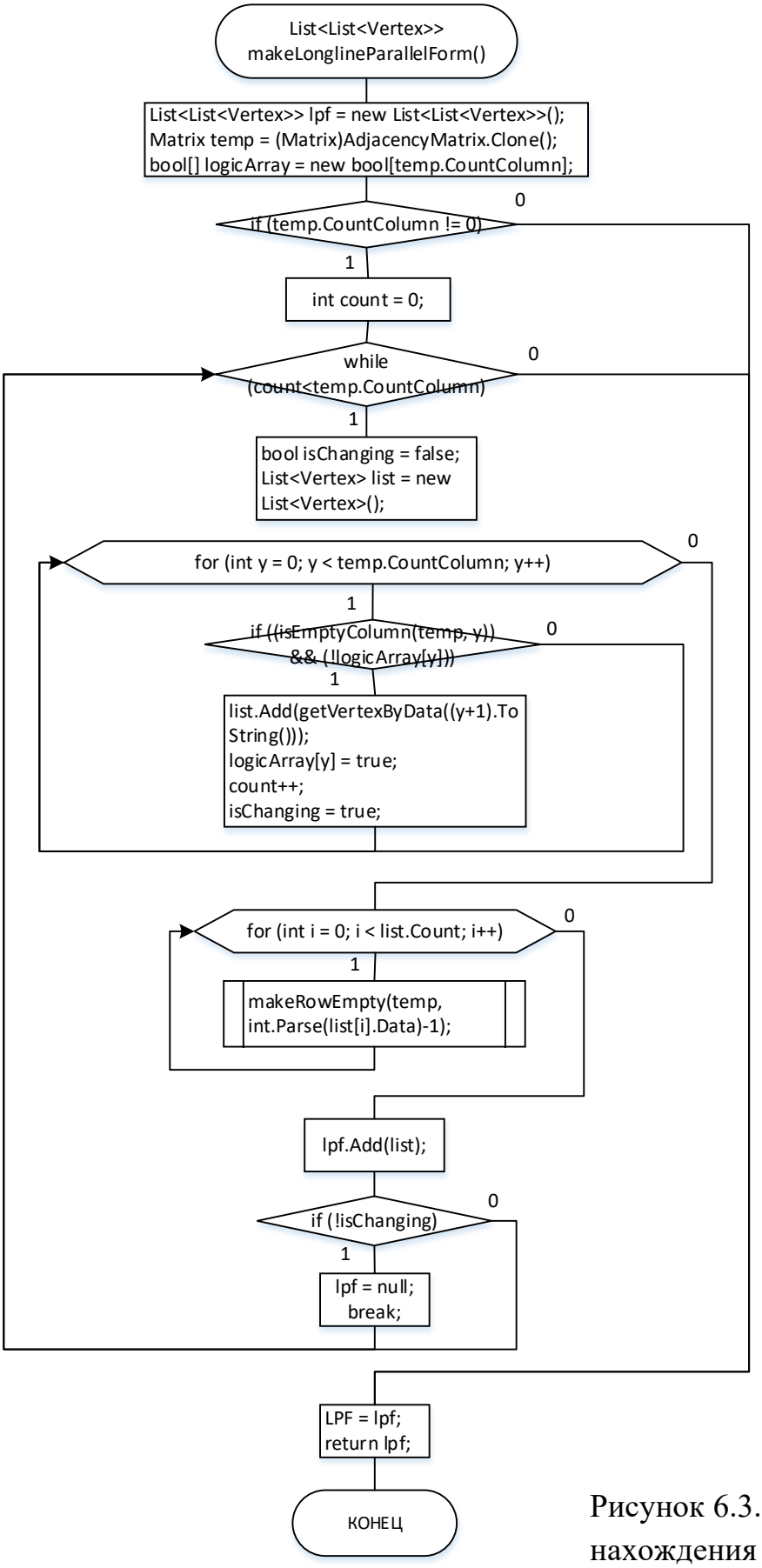

Рисунок 6.3.1 – функция нахождения ЯПФ

7. Описание классов и методов, используемых в программе

7.1 Метод Дийкстры

В программе реализованы методы:

- void SetVerticeNames(ref string[] arr) – инициализирует строковый массив, который содержит в себе пользовательские названия вершин графа
- void CorrectInputOfMatrixMember(out int var) – метод проверка на правельный ввод элемента матрицы весов (допускаемые значения >0)
- void CorrectInputofNumber(out int var) - метод-проверка на правельный ввод числа вершин (допускается количество от 0 до 9)
- void ShowWeightsMatrix(ref int[,] arr) - вывод матрицы весов на экран
- void ShowMinPathes(ref int[] minDistances) - вывод минимальных путей до вершин
- void ShowPath(ref List<int> ver) – вывод пути от конечной вершины до начальной
- int[,] PathesMatrix(int n) – инициализация матрицы путей для неориентированного графа
- int[,] PathesMatrixOriented(int n) - инициализация матрицы путей для ориентированного графа
- void Preparetions(ref int[] minDistances, ref int[] vertices) – приготовления к алгоритму (всем вершинам кроме первой метки intMax, все вершины не посещены)
- int Algorithm(ref int[,] weights, ref int[]minDistances, ref int[] vertices) – сам алгоритм
- List<int> Reverse(ref int[,] weights, ref int[] minDistances, ref int[] vertices, int lastVertice) – обратный проход

7.2 Ядро графа

В программе реализованы методы:

- void SetVerticeNames(ref string[] arr) – инициализирует строковый массив, который содержит в себе пользовательские названия вершин графа

- void CorrectInputofNumber (out int var) – метод-проверка на правельный ввод числа вершин (допускается количество от 0 до 9)
- void CorrectInputOfMatrixMember (out int var) – метод проверка на правельный ввод элемента матрицы смежности (допускаемые значения 0 и 1)
- void ShowAdjacencyMatrix(ref int[,] arr) – вывод в консоль матрицы смежности
- string Distribute(string s1, string s2) – дистрибутивный закон для двух операндов
- string Distribute(string s) – метод рекурсивно раскладывающий входную строку на пары операндов для метода Distribute, описанного выше
- List<string> Conversion(string s) – преобразование строки множества внутренней и внешней устойчивостей в список
- List<string> DNF(string s) – нахождение ДНФ
- int[,] AdjacencyMatrix(int n) – инициализация матрицы смежности
- List<string> MakeInternalStabilitySet(int[,] arr, int n) – нахождение множества внутренней устойчивости
- List<string> MakeExternalStabilitySet (int[,] arr) – нахождение множества внешней устойчивости
- List<string> FindCore(int[,] weights, int n) – нахождение ядра графа

7.2 Ярустно-параллельная форма представления графа

- void SetVerticeNames(ref string[] arr) - инициализирует строковый массив, который содержит в себе пользовательские названия вершин графа
- void CorrectInputOfMatrixMember(out int var) - метод проверка на правельный ввод элемента матрицы смежности (допускаемые значения 0 и 1)
- void CorrectInputofNumber(out int var) - метод-проверка на правельный ввод числа вершин (допускается количество от 0 до 9)
- int[,] AdjacencyMatrix(int n) – инициализация матрицы смежности

- void ShowAdjacencyMatrix(ref int[,] arr) вывод в консоль матрицы смежности
- void ShowMPF(ref string[,] arr, bool f) – вывести на экран ярусы вершин
- void MakeMPF(ref int[,] arr) – привести граф к ярустно-параллельной форме

8. Тестирование

8.1 Алгоритм Дейкстры

8.1.1 Ввод количества вершин

№ теста	Входные данные	Ожидаемый результат	Реальный результат	Примечание
1	-1	Сообщение о выходе за границы диапозона допустимых значений	Сообщение о выходе за границы диапозона допустимых значений	Ввод значения количества
2	0	Сообщение о выходе за границы диапозона допустимых значений	Сообщение о выходе за границы диапозона допустимых значений	
3	1	1	1	
4	8	8	8	
5	9	9	9	
6	10	Сообщение о том, что слишком большое значение	Сообщение о том, что слишком большое значение	
7	abc	Сообщение о необходимости ввода целого числа	Сообщение о необходимости ввода целого числа	
8	1.5	Сообщение о необходимости ввода целого числа	Сообщение о необходимости ввода целого числа	

8.1.2 Ввоб элементов матрицы весов

№ теста	Входные данные	Ожидаемый результат	Реальный результат	Примечание
1	-1	Сообщение о выходе за границы диапозона допустимых значений	Сообщение о выходе за границы диапозона допустимых значений	Ввод значения веса ребра в диапазоне >=0
2	0	0	0	
3	1	1	1	
4	8	8	8	

5	abc	Сообщение о необходимости ввода целого числа	Сообщение о необходимости ввода целого числа	
6	1000000000000000	Сообщение о необходимости ввода целого числа	Сообщение о необходимости ввода целого числа	

8.1.3 Нахождение наименьшего пути алгоритмом Дейкстры

Входные данные	Ожидаемый результат	Реальный результат	Примечание
граф на рисунке 8.3.1	A-E-F	A-E-F	
граф на рисунке 8.3.2	A-E-F	A-E-F	
граф на рисунке 8.3.3	A-C-F-E	A-C-F-E	
граф на рисунке 8.4.3	A-E-D	A-E-D	

Таблица 8.3.1 – пример графа №1

	a	b	c	d	e	f
a		11		14	15	
b			13			
c						13
d		7	11		9	
e		11	10			14
f						

Рисунок 8.3.2 – пример графа №2

	a	b	c	d	e	f
a		5	8	7	18	
b			11			
c						17
d		10	12		6	
e		7	8			11
f						

Рисунок 8.3.3 – пример графа №3

	a	b	c	d	e	f
a	0	7	9	0	0	14
b	7	0	10	15	0	0
c	9	10	0	11	0	2
d	0	15	11	0	6	0
e	0	0	0	6	0	9
f	14	0	2	0	9	0

Рисунок 8.3.4 – пример графа №4

	a	b	c	d	e
a		10	30	50	10
b					
c					10
d		40	20		
e	10		10	30	

8.2 Ядро графа

8.2.1 Ввод количества вершин

№ теста	Входные данные	Ожидаемый результат	Реальный результат	Примечание
1	-1	Сообщение о выходе за границы диапозона допустимых значений	Сообщение о выходе за границы диапозона допустимых значений	Ввод значения количества
2	0	Сообщение о выходе за границы диапозона допустимых значений	Сообщение о выходе за границы диапозона допустимых значений	
3	1	1	1	
4	8	8	8	
5	9	9	9	
6	10	Сообщение о том, что слишком большое значение	Сообщение о том, что слишком большое значение	
7	abc	Сообщение о необходимости ввода целого числа	Сообщение о необходимости ввода целого числа	
8	1.5	Сообщение о необходимости ввода целого числа	Сообщение о необходимости ввода целого числа	

8.2.2 Ввоб элементов матрицы смежности

№ теста	Входные данные	Ожидаемый результат	Реальный результат	Примечание
1	-1	Сообщение о выходе за границы диапозона	Сообщение о выходе за границы диапозона допустимых значений	Ввод значения веса ребра в

		допустимых значений		диапазоне (1 либо 0)
2	0	0	0	
3	1	1	1	
4	8	Сообщение о выходе за границы диапозона	Сообщение о выходе за границы диапозона	
5	abc	Сообщение о необходимости ввода целого числа	Сообщение о необходимости ввода целого числа	
6	1000000000000000	Сообщение о необходимости ввода целого числа	Сообщение о необходимости ввода целого числа	

8.2.3 Нахождение ядер графа

Входные данные	Ожидаемый результат	Реальный результат	Примечание
граф на рисунке 8.2.1	Ядра нет	Ядра нет	
граф на рисунке 8.2.2	Ядра нет	Ядра нет	
граф на рисунке 8.2.3	{3,5}	{3,5}	
граф на рисунке 8.2.3	{2,5}	{2,5}	

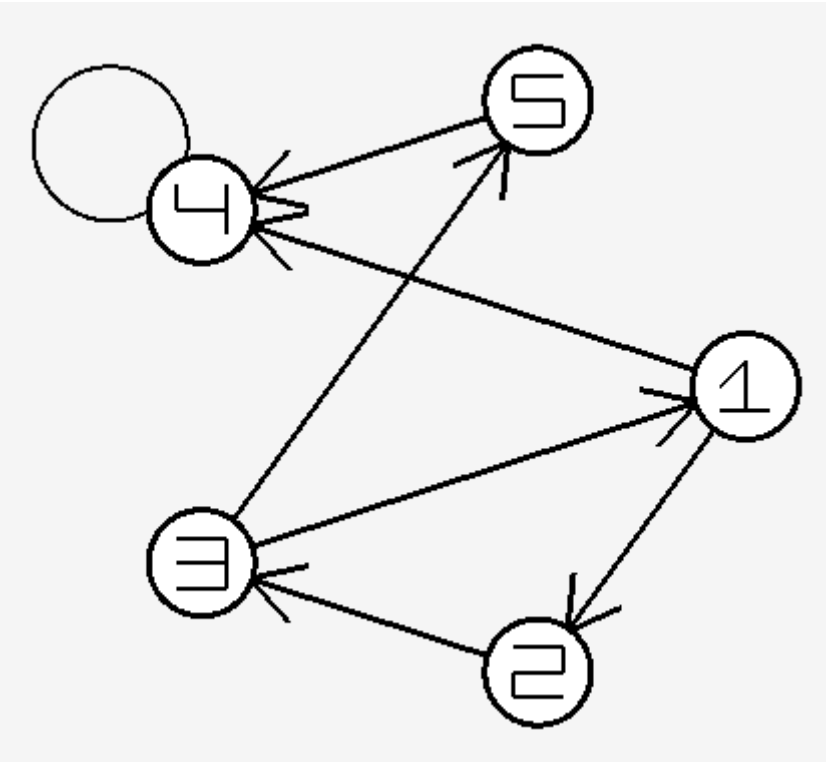

Рисунок 8.2.1 – пример графа

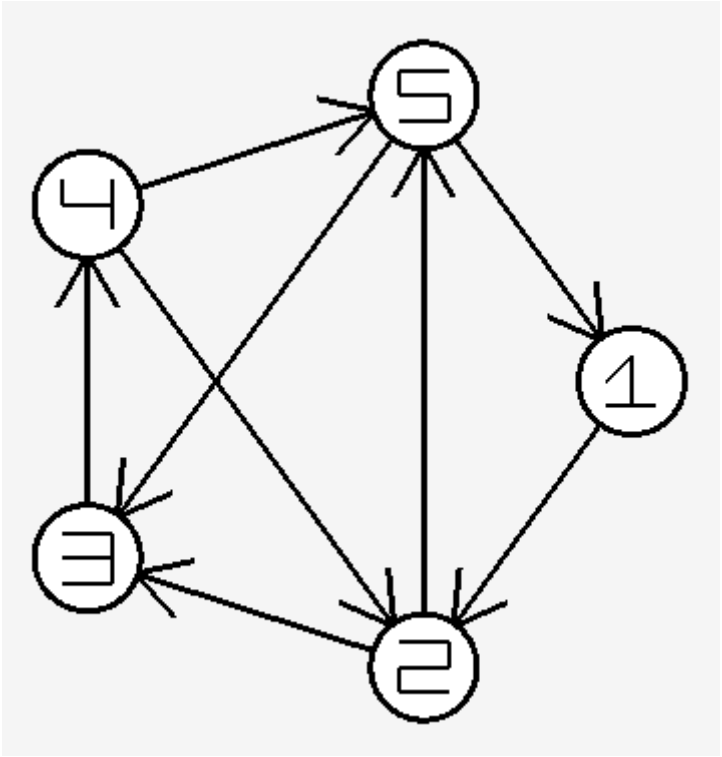

Рисунок 8.2.2 – пример графа №1

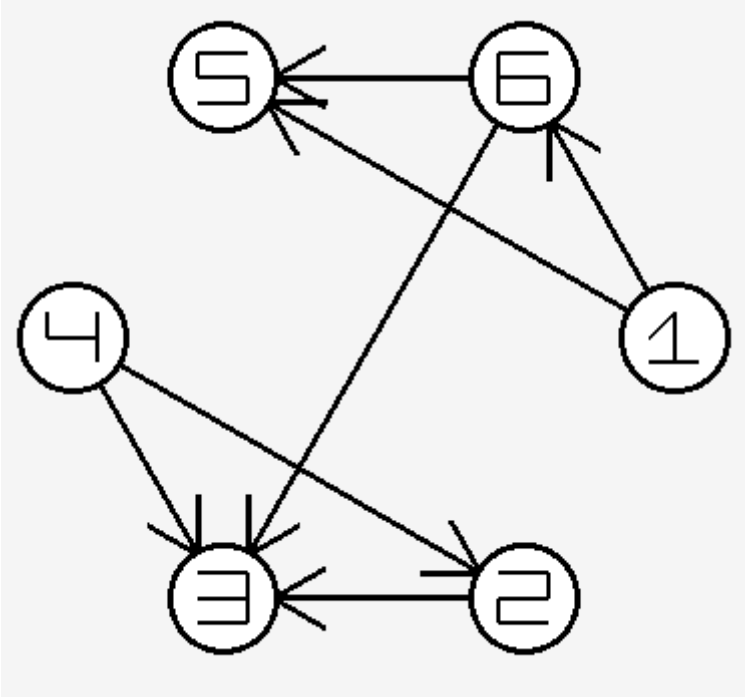

Рисунок 8.2.3 – пример графа №2

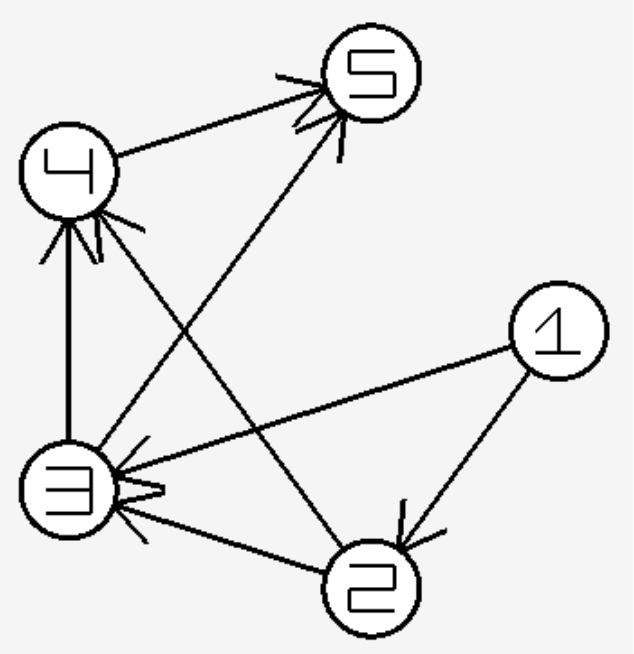

Рисунок 8.2.4 – пример графа №3

8.3 Ярусно-параллельная форма

8.3.1 Ввод количества вершин

№ теста	Входные данные	Ожидаемый результат	Реальный результат	Примечание
1	-1	Сообщение о выходе за границы диапозона допустимых значений	Сообщение о выходе за границы диапозона допустимых значений	Ввод значения количества
2	0	Сообщение о выходе за границы диапозона допустимых значений	Сообщение о выходе за границы диапозона допустимых значений	
3	1	1	1	
4	8	8	8	
5	9	9	9	
6	10	Сообщение о том, что слишком большое значение	Сообщение о том, что слишком большое значение	

7	abc	Сообщение о необходимости ввода целого числа	Сообщение о необходимости ввода целого числа	
8	1.5	Сообщение о необходимости ввода целого числа	Сообщение о необходимости ввода целого числа	

8.3.2 Ввоб элементов матрицы смежности

№ теста	Входные данные	Ожидаемый результат	Реальный результат	Примечание
1	-1	Сообщение о выходе за границы диапозона допустимых значений	Сообщение о выходе за границы диапозона допустимых значений	Ввод значения веса ребра в диапазоне (1 либо 0)
2	0	0	0	
3	1	1	1	
4	8	Сообщение о выходе за границы диапозона	Сообщение о выходе за границы диапозона	
5	abc	Сообщение о необходимости ввода целого числа	Сообщение о необходимости ввода целого числа	
6	1000000000000000	Сообщение о необходимости ввода целого числа	Сообщение о необходимости ввода целого числа	

8.3.3 Нахождение ЯПФ

Входные данные	Ожидаемый результат	Реальный результат	Примечание
граф на рисунке 8.3.1	Сообщение о том, что ЯПФ не существует	Ошибка ввода, т.к. я заведомо не допускаю пути из вершины в саму себя	
граф на рисунке 8.3.2	1 2 3 5 4	1 2 3 5 4	
граф на рисунке 8.3.3	1	1	
граф на рисунке 8.3.4	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	

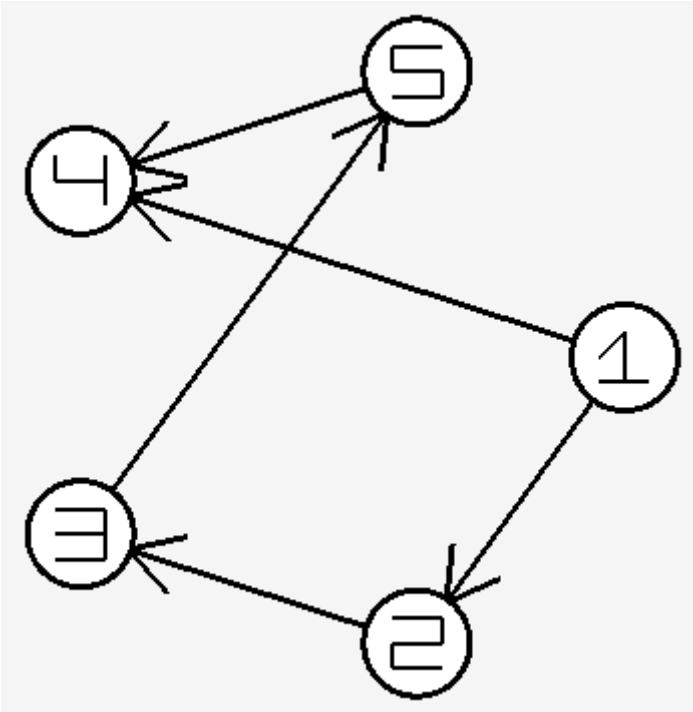

Рисунок 8.3.1 – пример графа №1

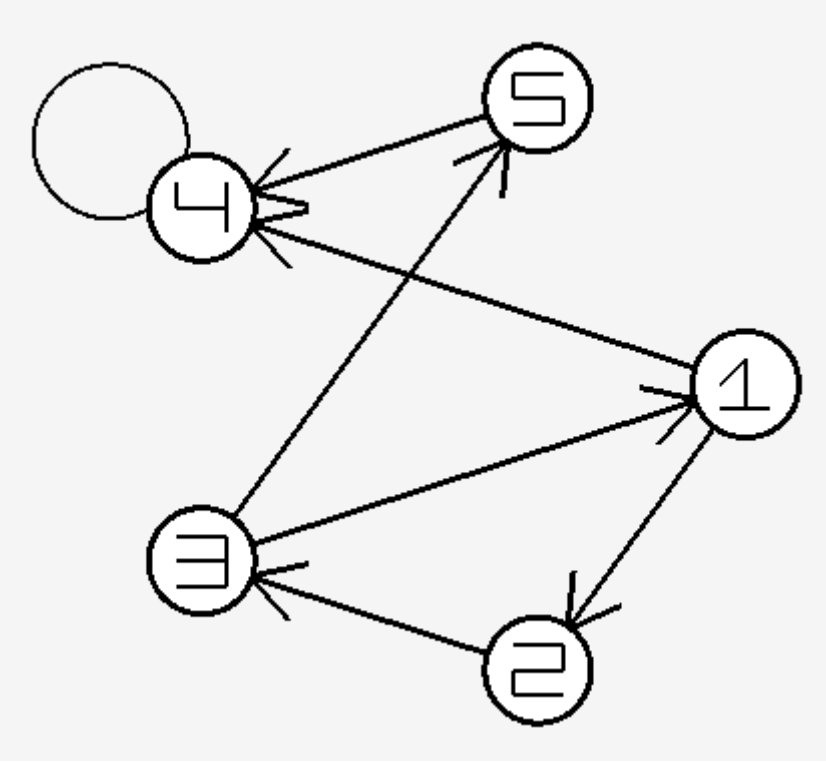

Рисунок 8.3.2 – пример графа №2

Рисунок 8.3.3 – пример графа №3

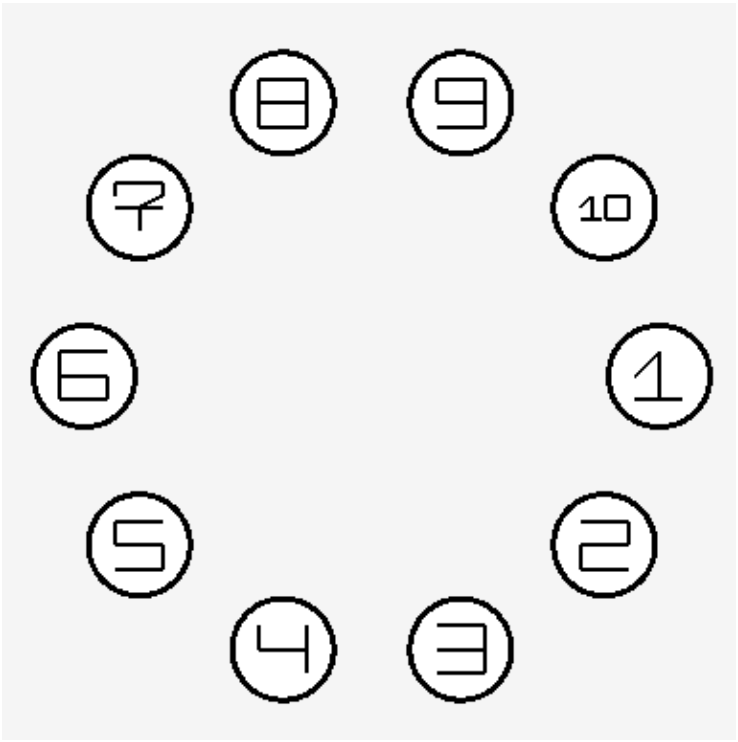

Рисунок 8.3.4 – пример графа №4

9. Код программы

9.1. Алгоритм Дейкстры

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Graphs
{
    class Program
    {
        static string[] verticeName;
        static void SetVerticeNames(ref string[] arr)
        {
            Console.WriteLine();
            for (int i = 0; i < arr.Length; i++)
            {
                Console.Write($"Введите название вершины {i + 1}: ");
                arr[i] = Console.ReadLine();
            }
            Console.WriteLine();
        } // задание навзаний вершин
        static void CorrectInputOfMatrixMember(out int var)   // Проверка на ввод числа в
матрице смежностиа
        {
            do
            {
                while (!int.TryParse(Console.ReadLine(), out var))
                {
                    Console.WriteLine("Ошибка ввода. Введите число ");
                }
                if (var < 0) Console.WriteLine("Ошибка ввода. Введите число не меньше
0");
            } while (var < 0);
        }
        static void CorrectInputofNumber(out int var)   // Проверка на ввод числа вершин
графа
        {
            do
            {
                while (!int.TryParse(Console.ReadLine(), out var))
                {
                    Console.WriteLine("Ошибка ввода. Введите целое число ");
                }
                if (var < 1 || var > 9) Console.WriteLine("Ошибка ввода. Введите число
больше 0 и меньше 10");
            } while (var < 1 || var > 9);
        }

        static void ShowWeightsMatrix(ref int[,] arr)
        {
            Console.WriteLine("\nМатрица весов:");
            for (int i = 0; i < arr.GetLength(0); i++)
            {
                for(int j = 0; j < arr.GetLength(1); j++)
                {
                    Console.Write(arr[i, j] + "    ");
                }
                Console.WriteLine("\n");
            }
```

```
        } // матрица весов
        static void ShowMinPathes(ref int[] minDistances)
        {
            Console.WriteLine("\nМинимальные пути до вершин:");
            for (int i = 0; i < minDistances.Length; i++)
                Console.WriteLine($"Вершина {verticeName[i] }: { minDistances[i] }");
            Console.WriteLine();
        } // минимальные пути до вершин
        static void ShowPath(ref List<int> ver) // путь
        {
            Console.WriteLine("\nСамый короткий путь из первой вершины в последнюю:");
            ver.Reverse();
            foreach(int element in ver)
                Console.Write(verticeName[element] + " ");
            Console.WriteLine("\n");
        }

        static int[,] PathesMatrix(int n) // Формирование матрицы путей
        {
            int[,] weights = new int[n, n];
            for (int i = 0; i < n; i++)
            {
                weights[i, i] = 0;
                for (int j = i + 1; j < n; j++)
                {
                    Console.Write($"Введите расстояние от вершины { i+1 } до вершины {
j+1 }: ");
                    CorrectInputOfMatrixMember(out weights[i, j]);
                    weights[j, i] = weights[i, j];
                }
            }
            ShowWeightsMatrix(ref weights);
            return weights;
        }
        static int[,] PathesMatrixOriented(int n) // Формирование матрицы путей
        {
            int[,] weights = new int[n, n];
            for (int i = 0; i < n; i++)
            {
                weights[i, i] = 0;
                for (int j = 0; j < n; j++)
                {
                    if (j != i)
                    {
                        Console.Write($"Введите расстояние от вершины { i + 1 } до
вершины { j + 1 }: ");
                        CorrectInputOfMatrixMember(out weights[i, j]);
                    }
                }
            }
            ShowWeightsMatrix(ref weights);
            return weights;
        }

        static void Preparetions(ref int[] minDistances, ref int[] vertices) //
приготовления
        {
            for (int i = 0; i < vertices.Length; i++)   // все вершины кроме первой не
посещены, а расстояния intmax
            {
                minDistances[i] = int.MaxValue;
                vertices[i] = 1;
            }
            minDistances[0] = 0;
        }
```

```
        static int Algorithm(ref int[,] weights, ref int[]minDistances, ref int[]
vertices)
        {
            int minindex, min, lastVertice = 0;
            do
            {
                minindex = int.MaxValue; // вершина с минимальной меткой
                min = int.MaxValue; // длина пути
                for (int i = 0; i < vertices.Length; i++) // выбор вершины с минимальной
меткой
                {
                    if ((vertices[i] == 1) && (minDistances[i] < min)) // Если вершину
ещё не обошли и метка меньше min
                    {
                        min = minDistances[i];
                        minindex = i;
                    }
                }
                /* Добавляем найденный минимальный вес
                к текущему весу вершины
                и сравниваем с текущим минимальным весом вершины */
                if (minindex != int.MaxValue)
                {
                    for (int i = 0; i < vertices.Length; i++)
                    {
                        if (weights[minindex, i] > 0)
                        {
                            int temp = min + weights[minindex, i];
                            if (temp < minDistances[i])
                            {
                                minDistances[i] = temp;
                            }
                        }
                    }
                    vertices[minindex] = 0;
                    lastVertice = minindex;
                }
            } while (minindex < int.MaxValue);
            return lastVertice;
        } // алгоритм

        static List<int> Reverse(ref int[,] weights, ref int[] minDistances, ref int[]
vertices, int lastVertice) // восстановление пути
        {
            List<int> ver = new List<int>(); // массив посещенных вершин
            ver.Add(lastVertice); // номер конечной вершины становится номером начальной
вершины
            int weight = minDistances[lastVertice];
            bool f = false; // достигли ли начальной вершины?
            do
            {
                for (int i = 0; i < weights.GetLength(0); i++)
                {
                    if (weights[i, ver.Last()] != 0 && weight - weights[i, ver.Last()] ==
minDistances[i])
                    {
                        weight = weight - weights[i, ver.Last()];
                        ver.Add(i);
                    }
                    if (ver.Last() == 0)
                        f = true;
                }
            } while (!f);
            return ver;
```

```
        }

        static void Main(string[] args)
        {
            int[,] weights = null; // матрица весов
            Console.Write("Введите число вершин: ");
            CorrectInputofNumber(out int n);
            verticeName = new string[n];
            SetVerticeNames(ref verticeName);
            int userChoice = 0;
            do
            {
                Console.WriteLine("Выберите тип графа:\n1 - Неориентированный\n2 -
Ориентированный");
                CorrectInputofNumber(out userChoice);
                switch (userChoice)
                {
                    case 1:
                        {
                            weights = PathesMatrix(n);
                            break;
                        }
                    case 2:
                        {
                            weights = PathesMatrixOriented(n);
                            break;
                        }
                    default:
                        {
                            Console.WriteLine("Ошибка! Введено неправильное число,
попрокуйте еще раз");
                            break;
                        }
                }
            } while (userChoice != 1 && userChoice != 2);

            int[] minDistances = new int[n]; // минимальная дистанция до каждой вершины
            int[] vertices = new int [n]; // посещенные вершины - 0, не посещенные 1
            Preparetions(ref minDistances, ref vertices);
            int lastVertice = Algorithm(ref weights, ref minDistances, ref vertices);
            ShowMinPathes(ref minDistances);
            List<int> ver = Reverse(ref weights, ref minDistances, ref vertices,
lastVertice);
            ShowPath(ref ver);

            Console.ReadKey();
        }
    }
}
```

9.2 Ядро графа

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CoreOfGraph
{
    class Program
    {
        static string[] verticeName;
        static void SetVerticeNames(ref string[] arr)
        {
            Console.WriteLine();
            for (int i = 0; i < arr.Length; i++)
            {
                Console.Write($"Введите название вершины {i + 1}: ");
                arr[i] = Console.ReadLine();
            }
            Console.WriteLine();
        } // задание навзаний вершин
        static void CorrectInputOfMatrixMember(out int var)   // Проверка на ввод числа в
матрице смежности
        {
            do
            {
                while (!int.TryParse(Console.ReadLine(), out var))
                {
                    Console.WriteLine("Ошибка ввода. Введите число ");
                }
                if (var < 0 || var > 1) Console.WriteLine("Ошибка ввода. Введите 1, если
есть путь, либо 0, если пути нет");
            } while (var < 0 && var > 1);
        }
        static void CorrectInputofNumber(out int var)   // Проверка на ввод числа вершин
графа
        {
            do
            {
                while (!int.TryParse(Console.ReadLine(), out var))
                {
                    Console.WriteLine("Ошибка ввода. Введите число ");
                }
                if (var < 1 || var > 9) Console.WriteLine("Ошибка ввода. Введите число
больше 0 и меньше 10");
            } while (var < 1 || var > 9);
        }
        static void ShowAdjacencyMatrix(ref int[,] arr)
        {
            Console.WriteLine("\nМатрица смежности:");
            for (int i = 0; i < arr.GetLength(0); i++)
            {
                for (int j = 0; j < arr.GetLength(1); j++)
                {
                    Console.Write(arr[i, j] + "    ");
                }
                Console.WriteLine("\n");
            }
        } // Матрица смежности

        static string Distribute(string s1, string s2)
        {
```

```
            string[] separator = new string[] { "+" };
            string[] s1Splitted = s1.Split(separator,
StringSplitOptions.RemoveEmptyEntries);
            string[] s2Splitted = s2.Split(separator,
StringSplitOptions.RemoveEmptyEntries);
            string result = null;
            for(int i = 0; i < s1Splitted.Length; i++)
            {
                for(int j = 0; j < s2Splitted.Length; j++)
                {
                    result += s1Splitted[i] + s2Splitted[j] + "+";
                }
            }
            return result;
        }
        static string Distribute(string s)
        {
            if (s == null) return null;
            string result = null, auxResult1 = null, auxResult2 = null;
            string s1 = null, s2 = null;
            string[] separator = new string[] { "*" };
            string[] sSplitted = s.Split(separator,
StringSplitOptions.RemoveEmptyEntries);
            if (sSplitted.Length == 2)
            {
                return result += Distribute(sSplitted[0], sSplitted[1]);
            }
            else if(sSplitted.Length != 1)
            {
                for (int i = 0; i < sSplitted.Length / 2 + 1; i++)
                    s1 += sSplitted[i] + "*";
                auxResult1 += Distribute(s1);
                for (int i = sSplitted.Length / 2 + 1; i < sSplitted.Length; i++)
                    s2 += sSplitted[i] + "*";
                auxResult2 += Distribute(s2);
                return result += Distribute(auxResult1, auxResult2);
            }
            else
            {
                return result = sSplitted[0];
            }
        }
        //static bool Absorb(string s1, string s2)
        //{
        //    for (int i = 0; i < s2.Length; i++)
        //    {
        //        if (!s1.Contains(s2[i]))
        //            return false;
        //    }
        //    return true;
        //}
        static List<string> Conversion(string s)
        {
            if (s == null) return null;
            string[] separator = new string[] { "+" };
            string[] sSplitted = s.Split(separator,
StringSplitOptions.RemoveEmptyEntries);
            char[] aux;
            for (int i = 0; i < sSplitted.Length; i++)
            {
                aux = sSplitted[i].Distinct().ToArray();
                sSplitted[i] = string.Concat(aux);
            }
            List<string> reformed = sSplitted.ToList();
            for(int i = 0; i < reformed.Count; i++)
```

```
    {
        char[] charArray = reformed[i].ToCharArray();
        Array.Sort(charArray);
        reformed[i] = new String(charArray);
    }
    //for (int i = 1; i < reformed.Count; i++)
    //{
    //    for (int j = 0; j < i; j++)
    //    {
    //        if (reformed[i].Contains(reformed[j]))
    //        {
    //            reformed.RemoveAt(i);
    //            i--;
    //            break;
    //        }
    //        else if (reformed[j].Contains(reformed[i]))
    //        {
    //            reformed.RemoveAt(j);
    //            break;
    //        }
    //    }
    //}
    return reformed;
}
static List<string> DNF(string s)
{
    string aux = Distribute(s);
    List<string> result = Conversion(aux);
    return result;
}

static int[,] AdjacencyMatrix(int n) // Формирование матрицы смежности
{
    int[,] weights = new int[n, n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
           Console.Write($"{ i + 1 } -> { j + 1 }: ");
           CorrectInputOfMatrixMember(out weights[i, j]);
        }
    }
    ShowAdjacencyMatrix(ref weights);
    return weights;
}
static List<string> MakeInternalStabilitySet(int[,] arr, int n)
{
    string aux = null;
    for (int i = 0; i < arr.GetLength(0); i++)
    {
        for (int j = 0; j < arr.GetLength(1); j++)
        {
            if (arr[i, j] == 1)
                aux += i + "+" + j + "*";
        }
    }
    List<string> result = DNF(aux);
    if(result != null )
    {
        string all = null;
        for (int i = 0; i < n; i++)
            all += i;
        char[] symbols;
        for (int i = 0; i < result.Count; i++)
        {
```

```
                    symbols = all.Except(result[i]).ToArray();
                    result[i] = string.Concat(symbols);
                }
            }
            return result;
        }
        static List<string> MakeExternalStabilitySet(int[,] arr)
        {
            for (int i = 0; i < arr.GetLength(0); i++)
                arr[i,i] = 1;
            string aux = null;
            for (int i = 0; i < arr.GetLength(0); i++)
            {
                for (int j = 0; j < arr.GetLength(1); j++)
                {
                    if (arr[i, j] == 1)
                        aux += j + "+";
                }
                aux += "*";
            }
            List<string> result = DNF(aux);
            return result;
        }
        static List<string> FindCore(int[,] weights, int n)
        {
            List<string> internalSet = MakeInternalStabilitySet(weights, n);
            List<string> externalSet = MakeExternalStabilitySet(weights);
            List<string> core = new List<string>();
            if(internalSet != null && externalSet != null)
            {
                for (int i = 0; i < internalSet.Count; i++)
                {
                    for (int j = 0; j < externalSet.Count; j++)
                    {
                        if (internalSet[i] == externalSet[j] &&
!core.Contains(internalSet[i]))
                            core.Add(internalSet[i]);
                    }
                }
            }
            return core;
        }

        static void Main(string[] args)
        {
            Console.Write("Введите число вершин: ");
            CorrectInputofNumber(out int n);
            verticeName = new string[n];
            SetVerticeNames(ref verticeName);
            int[,] adjacencyMatrix = AdjacencyMatrix(n);
            List<string> result = FindCore(adjacencyMatrix, n);
            if(result.Count == 0) Console.WriteLine("\nУ данного графа нет ядра!");
            else
            {
                Console.WriteLine("\nЯдро графа:");
                foreach (string element in result)
                {
                    for(int i = 0; i < element.Length; i++)
                    {

Console.Write(verticeName[(int)Char.GetNumericValue(element[i])]);
                    }
                }
                    Console.Write(" ");
            }
```

```
            Console.ReadKey();
        }
    }
}
```

9.3 Ярусно-паралелльная форма

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Multi_Parallel_Form
{
    class Program
    {
        static string[] verticeName;
        static void SetVerticeNames(ref string[] arr)
        {
            Console.WriteLine();
            for (int i = 0; i < arr.Length; i++)
            {
                Console.Write($"Введите название вершины {i + 1}: ");
                arr[i] = Console.ReadLine();
            }
            Console.WriteLine();
        } // задание навзаний вершин
        static void CorrectInputOfMatrixMember(out int var)   // Проверка на ввод числа в
матрице смежностиа
        {
            do
            {
                while (!int.TryParse(Console.ReadLine(), out var))
                {
                    Console.WriteLine("Ошибка ввода. Введите число ");
                }
                if (var < 0 || var > 1) Console.WriteLine("Ошибка ввода. Введите 1, если
есть путь, либо 0, если пути нет");
            } while (var < 0 || var > 1);
        }
        static void CorrectInputofNumber(out int var)   // Проверка на ввод числа вершин
графа
        {
            do
            {
                while (!int.TryParse(Console.ReadLine(), out var))
                {
                    Console.WriteLine("Ошибка ввода. Введите число ");
                }
                if (var < 1 || var > 9) Console.WriteLine("Ошибка ввода. Введите число
больше 0 и меньше 10");
            } while (var < 1 && var > 9);
        }

        static int[,] AdjacencyMatrix(int n) // Формирование матрицы смежности
        {
            int[,] arr = new int[n, n];
            for (int i = 0; i < n; i++)
            {
                arr[i, i] = 0;
                for (int j = 0; j < n; j++)
                {
```

```
                    if (j != i)
                    {
                        Console.Write($"{ i + 1 } -> { j + 1 }: ");
                        CorrectInputOfMatrixMember(out arr[i, j]);
                    }
                }
            }
            ShowAdjacencyMatrix(ref arr);
            return arr;
        }
        static void ShowAdjacencyMatrix(ref int[,] arr)
        {
            Console.WriteLine("\nМатрица смежности:");
            for (int i = 0; i < arr.GetLength(0); i++)
            {
                for (int j = 0; j < arr.GetLength(1); j++)
                {
                    Console.Write(arr[i, j] + "    ");
                }
                Console.WriteLine("\n");
            }
        } // Матрица смежности
        static void ShowMPF(ref string[,] arr, bool f)
        {
            Console.WriteLine("\nЯрусно-параллельная форма:");
            if (!f)
            {
                for (int i = 0; i < arr.GetLength(0); i++)
                {
                    for (int j = 0; j < arr.GetLength(1) && arr[i, j] != null; j++)
                    {
                        Console.Write(arr[i, j] + " ");
                    }
                    Console.WriteLine();
                }
            }
            else
            {
                Console.WriteLine("Привести данный граф к ярусно-параллельной форме
невозможно, так как он имеет цикл\n");
            }
        }

        static void MakeMPF(ref int[,] arr)
        {
            /* массив столбцов, имеющий 3 состояния:
                0 - столбец еще не зачеркнут
                1 - столбец зачеркнут
                2 - столбец был зачеркнут в прошлых итерациях */
            int[] columns = new int[arr.GetLength(1)];
            for (int i = 0; i < columns.Length; i++)
                columns[i] = 0;
            bool isEnded = false; // конец цикла - все столбцы зачеркнуты
            bool isCycled = false; // есть ли в графе цикл
            int tier = 0; // номер очередного яруса
            string[,] MPF = new string[arr.GetLength(0), arr.GetLength(0)]; // массив
вершин в ярусно-параллельной форме
            do
            {
                for (int j = 0; j < arr.GetLength(0); j++)
                {
                    if(columns[j] == 0) // если столбец не зачеркнут
                    {
                        for (int i = 0; i < arr.GetLength(1); i++)
                        {
```

```
                            if (arr[i, j] == 0) columns[j] = 1; // зачеркиваем столбец
                            else
                            {
                                columns[j] = 0;
                                break; // не зачеркиваем столбец
                            }
                        }
                    }
                }
                for (int i = 0, k = 0; i < columns.Length; i++)
                {
                    if (columns[i] == 1) // если на этой итерации был зачекнут столбец
                    {
                        for (int j = 0; j < arr.GetLength(1); j++) // зачеркиваем
строку(обнуляем ее)
                            arr[i, j] = 0;
                        MPF[tier, k++] = verticeName[i]; // добавляем номер в массив
вывода
                    }
                }
                tier++;
                if(tier > arr.GetLength(0))
                {
                    isCycled = true;
                    break;
                }
                isEnded = true;
                for(int i = 0; i < columns.Length; i++)
                {
                    if (columns[i] == 1)
                        columns[i] = 2; // переводим столбец состояние "был зачернут
ранее"
                    if(columns[i] == 0)
                        isEnded = false;
                }
            } while (!isEnded);
            ShowMPF(ref MPF, isCycled);
        }

        static void Main(string[] args)
        {
            Console.Write("Введите число вершин: ");
            CorrectInputofNumber(out int n);
            verticeName = new string[n];
            SetVerticeNames(ref verticeName);
            int[,] adjacencyMatrix = AdjacencyMatrix(n);
            MakeMPF(ref adjacencyMatrix);
            Console.ReadKey();
        }
    }
}
```

10. Список используемой литературы

1. Алгоритм Дейкстры. Поиск оптимальных маршрутов на графе

[Электронный ресурс] URL: https://habr.com/post/111361/ (Дата обращения: 20.12.18)
2. Ядро графа [Электронный ресурс] URL: http://sci.sernam.ru/book_comb.php?id=32 (Дата обращения: 20.12.18)