Отчет по алгоритму построения латинских композиций в ориентированном графе

Теоретические сведения

Латинская композиция графа G = (V, E) представляет собой специфическую структуру, которая позволяет перечислять все возможные пути между заданными вершинами графа, удовлетворяющие определённым условиям.

Описание алгоритма

Алгоритм выполняет построение латинских композиций в ориентированном графе и перечисление всех возможных путей от начальной до конечной вершины с использованием этих композиций.

Этапы алгоритма:

1. **Инициализация:**
   - Создаются начальные матрицы M1 и M1′ где M1[v] содержит соседей вершины v, а M1′[v] содержит вершины, для которых v является соседом.
2. **Построение латинских композиций:**
   - Композиция матриц M1 и M1′ для получения новых матриц.
   - Этот процесс повторяется для построения всех латинских композиций.
3. **Перечисление путей:**
   - Используя построенные латинские композиции, выполняется поиск всех путей от начальной до конечной вершины с использованием рекурсивного поиска в глубину (DFS).

Оценка временной сложности алгоритма

Основные этапы алгоритма имеют следующие временные сложности:

- Построение начальных матриц M1 и M1′: O(n^2), где n — количество вершин в графе.
- Композиция матриц: O(n^3), так как необходимо проверять все возможные пары вершин.
- Перечисление всех путей: Зависит от количества путей и структуры графа, в худшем случае O(n!).

Таким образом, общая временная сложность алгоритма в худшем случае может достигать O(n!).

Логическая блок-схема

Программа

Программа написана на языке Python с использованием библиотек:

- Tkinter для графического интерфейса пользователя.
- NetworkX для работы с графами.
- Matplotlib для визуализации графов.

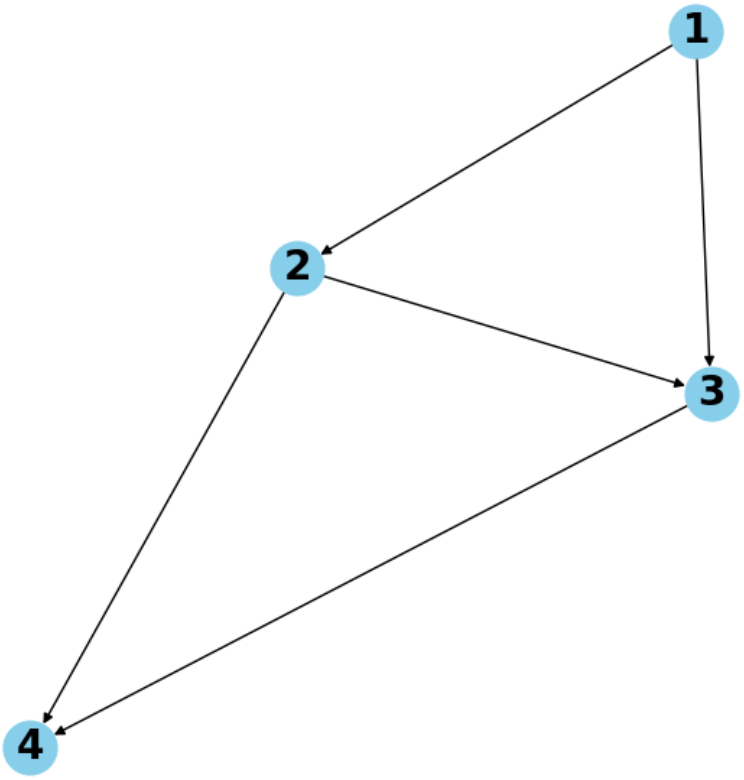
Оценка временной сложности алгоритма

- Построение начальных матриц M1 и M1′: O(n^2).
- Композиция матриц: O(n^3).
- Перечисление всех путей: O(n!).

Тестовые примеры. Скриншоты программы.

Пример 1.

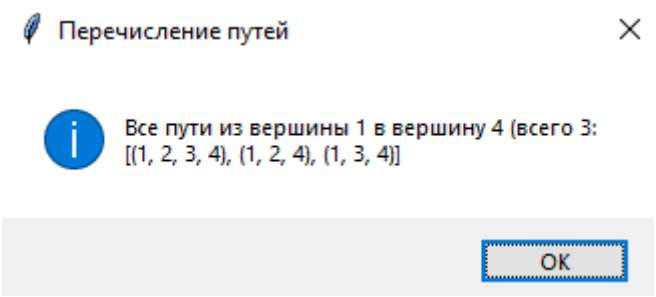
1. Введем ребра



2. Нажмем «Перечислить пути» и введем:
   начальную точку – 1; конечную точку – 4

3. Ожидаемый результат:
   3 – (1, 2, 3, 4), (1, 2, 4), (1, 3, 4)

4. Полученный результат:



Результат: верно.

Пример прикладной задачи

Модель управления маршрутизацией в сети:

- Создать программу, которая оптимизирует маршрутизацию в сети так, чтобы минимизировать количество пересылок данных между узлами. Каждому маршруту сопоставляется уникальный идентификатор, что минимизирует задержки и повышает эффективность сети.