Министерство образования и науки Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого

Институт прикладной математики и информатики

Лабораторная работа №2 по дисциплине

Дискретная математика

Тема: «Графы»

Вариант 3 – Алгоритм поиска наибольшего паросочетания

Выполнил студент гр.5030102/20202 Дрекалов Н. С.

Преподаватель Новиков Ф. А.

Санкт-Петербург

2024

Оглавление

[Формулировка задания 3](#_Toc182747069)

[Зачем нужно находить наибольшее паросочетание 3](#_Toc182747070)

[Используемый язык программирования и системы сборки 3](#_Toc182747071)

[Описание алгоритма поиска наибольшего паросочетания – Алгоритм Куна 4](#_Toc182747072)

[Построение массива классификации по долям (BFS) 4](#_Toc182747073)

[Поиск наибольшего паросочетания с помощью поиска в глубину (DFS) 4](#_Toc182747074)

[Демонстрация работы алгоритма 5](#_Toc182747075)

[Область применения реализованного алгоритма 7](#_Toc182747076)

[Формат входных и выходных данных 8](#_Toc182747077)

[Сложность алгоритма 8](#_Toc182747078)

[Сравнение работы алгоритма на различных допустимых входных данных 8](#_Toc182747079)

[О способе представления графов в программе 9](#_Toc182747080)

# Формулировка задания

В лабораторной работе требуется реализовать алгоритм поиска наибольшего паросочетания в двудольном графе

# Зачем нужно находить наибольшее паросочетание

Наибольшее паросочетание необходимо, например, в задачах распределения ресурсов, где есть ограничение на совместное использование ресурса.

Примеры:

* Есть набор учеников, каждый из которых хочет сидеть в паре с некоторыми другими учениками. Задача – рассадить учеников парами так, чтобы как можно большее их число сидело с тем, кем хотелось (это пример для недвудольного графа)
* Есть некоторые число работников и задач, причём каждый из работников способен выполнить только определённые задачи, и работник может выполнять только одну задачу одновременно. Необходимо распределить задачи между работниками так, чтобы как можно большее количество работников были заняты.

# Используемый язык программирования и системы сборки

* C++ 20
* CMake 3.28.3
* Ninja 1.11.1

# Описание алгоритма поиска наибольшего паросочетания – Алгоритм Куна

## Построение массива классификации по долям (BFS)

Сам массив классификации представляет из себя массив bool значений. True – если вершина в одной доле, False – если в другой

1. Обходим с помощью BFS вершины графа – для каждой вершины выполняем 2
2. Добавляем не посещенные смежные вершины в массив со значением, противоположным родителю и помещаем эти вершины в очередь обхода в ширину

## Поиск наибольшего паросочетания с помощью поиска в глубину (DFS)

1. Перебираем все вершины одной из долей и для каждой из вершин запускаем DFS (2-9)
2. Создаём стек, в котором будем хранить вершину и соответствующий ей список направленных рёбер (чтобы повторять логику рекурсии) и массив посещений.
3. Помещаем в стек начальную вершину
4. Если стек не пустой – выполняем пункты 5-9. Иначе возвращаемся к 1.
5. Достаём вершину из стека и пропускаем её (возвращаемся к пункту 4), если она уже была посещена. Если она не была посещена – помечаем её, как посещенную.
6. Перебираем все смежным к данной вершине (это будут вершины из правой доли) – для каждой выполняем 7-8
7. Если смежной вершины нет в паросочетании – добавляем её в паросочетание
8. Если смежная вершина есть в паросочетании – добавляем её вместе с увеличенной цепочкой в стек – чтобы проверить на следующих итерациях, можно ли из её текущей пары найти другую вершину (это позволит соединится со смежной вершиной, а предыдущую вершину, которая составляла ей пару, соединить со свободной)
9. Возвращаемся к пункту 4.

# Демонстрация работы алгоритма

Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Найдём наибольшее паросочетание с помощью вышеописанного алгоритма в таком графе

Далее будут расписаны шаги, после них – соответсвующие шагам рисунки.

1. Начальное состояние
2. Смотрим на вершину 1 – для неё сразу находится пара 5. Записываем ребро 5, 1 в ответ.
3. Смотрим на вершину 2 – для неё аналогично сразу нахожится пара 4. Записываем 4, 2 в ответ.
4. Смотрим на вершину 3 – для неё первая пара 5 уже есть в паросочетании.
5. Смотрим, можно ли заменить 1, которая парная к 5, на другую свободную вершину – получаем, что нет: у вершины 1 есть только вершина 5.
6. Значит, смотрим далее на смежные к 3 вершины – получаем, что есть свободная вершина 6. Записываем 6, 3 в ответ.

Таким образом, мы получаем, что наибольше паросочетание – [(5, 1), (4, 2), (6, 3)] – имеет длину 3.

Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски


Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски


Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски
Наброски


# Область применения реализованного алгоритма

Алгоритм можно применять для любого двудольного графа, который записан в файл в виде списка смежности

*Случаи некорректной работы программы:*

* Пользователь указал путь к пустому файлу.
* Пользователь указал путь к файлу, которого не существует или у программы недостаточно прав для чтения из указанного места.
* Пользователь указал путь к файлу, который был поврежден или не содержит валидного представления графа.
* Пользователь указал путь к файлу, в котором записан не двудольный граф.

В остальных случаях программа будет работать корректно.

# Формат входных и выходных данных

*Входные данные* – двудольный граф в файле, представленный как список смежности.

*Выходные данные* – наибольшее паросочетание, записанное в файл в виде списка рёбер.

# Сложность алгоритма

Пусть – число вершин в левой доле, – число рёбер.

Алгоритм выполняет следующие алгоритмы в процессе работы:

1. Отдельно с помощью BFS строит массив классификации по долям. Общая сложность BFS –
2. Перебирает все вершины из левой доли и для каждой вершины запускает поиск в глубину (DFS). Сложность перебора вершин из левой доли – ,
3. Инициализирует массив посещений вершин. Сложность – O(M).
4. DFS. DFS проходит по каждому ребру не более одного раза, поэтому сложность DFS – .

Таким образом, так как 1 выполняется отдельно, а 3 и 4 выполняются последовательно в цикле 2, общая сложность алгоритма

# Сравнение работы алгоритма на различных допустимых входных данных

Сложность алгоритма O (M \* N) (M – число вершин в левой доле, N – число рёбер), однако возможны ситуации, когда для каждой вершины достаточно будет пройти только по одному ребру (случай, когда вершина i связана со своим «зеркалом» (в данном случае «зеркалом» вершины из левой доли я называю вершину из правой доли, которая хранится по такому же номеру в правой доле, как и вершина в левой доле, если пронумеровать эти доли отдельно по порядку следования в массиве смежности) в правой доле и всеми нижестоящими, но не связана с верхними) – в таком случае сложность будет O(M), то есть *алгоритм будет работать лучше всего*. Аналогично, алгоритм будет *работать хуже всего*, если текущая вершина из левой доли связана только с «зеркалом» и вышестоящими вершинами – в таком случае DFS для каждой вершины будет проходить все связанный с ней ребра перед тем, как найти единственное подходящее паросочетание.

Алгоритм *не будет работать*, если граф не двудольный.

# О способе представления графов в программе

В программе граф представлен как массив смежности, где индексу i соответствует i-я вершина, и по индексу хранится массив смежных вершин. Также для определения, к какой доле относится граф, дополнительно создается массив вершин со значениями True/False, в зависимости от того, левой или правой доле принадлежит вершина.

Я выбрал этот способ из-за его в эффективности по памяти и быстроте нужных операций: граф занимает O ( памяти, где M – число вершин, а – число рёбер, при этом все нужные операции выполняются за хорошую сложность:

* Получение списка смежных вершин – O (1)
* Перебор всех смежных к вершине вершин – O (n), где n – число смежных к вершине вершин (то есть число рёбер, соединённых с вершиной)