МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА 25

КУРСОВАЯ РАБОТА (ПРОЕКТ)

ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

доцент, канд. тех. наук		Е. М. Линский
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ

МАКСИМАЛЬНЫЙ ПОТОК В ТРАНСПОРТНОЙ СЕТИ

по дисциплине: ОСНОВЫ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №	2352		Галинов Л.П
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

Содержание

ПОСТАНОВКА ЗАДАЧИ

Задачей данной курсовой работы является разработка программы, которая реализует алгоритм нахождения максимального потока в транспортной сети. Входные данные содержат описание транспортной сети, представленной графом, где вершины — это узлы, а ребра — это каналы между узлами с ограниченной пропускной способностью. Необходимо вычислить максимальный поток, который можно провести из источника в сток, используя алгоритм поиска максимального потока.

Максимальный поток в сети — это наибольшее количество материала, энергии, данных или других ресурсов, которые могут быть переданы из исходного узла (источника) в конечный узел (сток) через сеть с учетом ограничений на пропускную способность ребер.

Алгоритм нахождения максимального потока широко используется в различных областях, таких как оптимизация транспортных потоков, распределение ресурсов, задачи в теории графов и даже в некоторых областях теории игр и сетевой безопасности.

В качестве метода решения задачи предлагается реализовать алгоритм Эдмондса-Карпа, который является модификацией алгоритма Форда-Фалкерсона и использует поиск в ширину для нахождения augmenting path (увеличивающего пути).

ОПИСАНИЕ АЛГОРИТМА

Алгоритм Эдмондса-Карпа является реализацией улучшенного метода Форда-Фалкерсона для поиска максимального потока в сети. Он использует поиск в ширину (BFS, Breadth-First Search) для нахождения увеличивающих путей, что гарантирует, что алгоритм будет завершаться за полиномиальное время.

Задача

Дано направленное графовое представление транспортной сети, где:

Вершины графа представляют узлы сети.

Ребра графа имеют пропускную способность, которая ограничивает объем потока, который может быть передан по этому ребру.

Задача заключается в том, чтобы найти максимальный поток, который может быть передан из исходной вершины (источник) в сток (целевой узел), с учетом ограничений на пропускную способность.

Основные идеи алгоритма

1. **Остаточная сеть**: Для каждого ребра в графе мы поддерживаем остаточную пропускную способность, которая указывает, сколько еще потока можно провести по этому ребру. Остаточная пропускная способность может быть:
   - Положительная: если поток по ребру еще не максимален, то остаточная пропускная способность больше нуля.
   - Нулевая: если весь поток на этом ребре уже проходит, остаточная пропускная способность равна нулю.
   - Отрицательная (в случае обратных ребер): когда поток по ребру уже существует, можно пройти поток в обратном направлении.
2. **Увеличивающий путь**: Это путь от источника к стоку, по которому можно провести дополнительный поток. Для нахождения увеличивающего пути используется поиск в ширину (BFS).
3. **Алгоритм**:
   - Построение остаточной сети.
   - На каждом шаге ищется увеличивающий путь с помощью BFS.
   - По найденному пути увеличивается поток, и обновляются остаточные пропускные способности.
   - Процесс продолжается, пока увеличивающий путь существует.

Пошаговое выполнение алгоритма с примером

Рассмотрим пошаговое выполнение алгоритма Эдмондса-Карпа на примере транспортной сети. Пример включает четыре вершины и пять ребер с пропускными способностями:

Входные данные:

4 5

1 2 10

1 3 10

2 3 5

2 4 10

3 4 10

Граф сети:

```
   (10)       (10)
1 ------> 2 ------> 4
|          |          ^
|          |          |
v (10)     v (5)      v (10)
3 --------> 4
```

Источник: вершина 1

Сток: вершина 4

Инициализация:

Все потоки на ребрах равны 0.

Остаточные пропускные способности равны исходным пропускным способностям:

- $(1 \to 2)$: 10
- $(1 \to 3)$: 10
- $(2 \to 3)$: 5
- $(2 \to 4)$: 10
- $(3 \to 4)$: 10

Шаг 1: Первый проход через алгоритм (поиск увеличивающего пути)

Шаг 1.1: ищем увеличивающий путь с помощью поиска в ширину (BFS).

Путь: $1 \to 2 \to 4$

Минимальная пропускная способность на пути $1 \to 2 \to 4$ равна 10.

Шаг 1.2: обновляем поток:

Увеличиваем поток на 10 по пути $1 \to 2 \to 4$.

Обновляем остаточные пропускные способности:

- o $(1 \to 2)$: 0 (поток = 10)
- o $(2 \to 4)$: 0 (поток = 10)
- o Обратные ребра:
- o $(2 \to 1)$: 10
- o $(4 \to 2)$: 10

Текущий поток: 10

Остаточные пропускные способности:

$(1 \to 2)$: 0

$(1 \to 3)$: 10

$(2 \to 3)$: 5

$(2 \to 4)$: 0

$(3 \to 4)$: 10

Обратные ребра:

- o $(2 \to 1)$: 10
- o $(4 \to 2)$: 10

Шаг 2: Второй проход через алгоритм (поиск увеличивающего пути)

Шаг 2.1: ищем увеличивающий путь с помощью BFS.

Путь: $1 \to 3 \to 4$

Минимальная пропускная способность на пути $1 \to 3 \to 4$ равна 10.

Шаг 2.2: обновляем поток:

Увеличиваем поток на 10 по пути $1 \to 3 \to 4$.

Обновляем остаточные пропускные способности:

- o $(1 \to 3)$: 0 (поток = 10)
- o $(3 \to 4)$: 0 (поток = 10)
- o Обратные ребра:
- o $(3 \to 1)$: 10

o $(4 \to 3)$: 10

Текущий поток: 20

Остаточные пропускные способности:

$(1 \to 2)$: 0

$(1 \to 3)$: 0

$(2 \to 3)$: 5

$(2 \to 4)$: 0

$(3 \to 4)$: 0

Обратные ребра:

o $(2 \to 1)$: 10

o $(4 \to 2)$: 10

o $(3 \to 1)$: 10

o $(4 \to 3)$: 10

Шаг 3: Третий проход через алгоритм (поиск увеличивающего пути)

Шаг 3.1: ищем увеличивающий путь с помощью BFS.

Путь: $1 \to 2 \to 3 \to 4$

Минимальная пропускная способность на пути $1 \to 2 \to 3 \to 4$ равна 5 (на ребре $2 \to 3$).

Шаг 3.2: обновляем поток:

Увеличиваем поток на 5 по пути $1 \to 2 \to 3 \to 4$.

Обновляем остаточные пропускные способности:

o $(1 \to 2)$: 0 (поток = 10)

o $(2 \to 3)$: 0 (поток = 5)

o $(3 \to 4)$: 5 (поток = 5)

o Обратные ребра:

o $(2 \to 1)$: 10

o $(3 \to 2)$: 5

o $(4 \to 3)$: 5

Текущий поток: 25

Остаточные пропускные способности:

$(1 \to 2)$: 0

$(1 \to 3)$: 0

$(2 \to 3)$: 0

$(2 \to 4)$: 0

$(3 \to 4)$: 5

Обратные ребра:

- o $(2 \to 1)$: 10
- o $(4 \to 2)$: 10
- o $(3 \to 1)$: 10
- o $(4 \to 3)$: 5
- o $(3 \to 2)$: 5

Шаг 4: Четвертый проход через алгоритм (поиск увеличивающего пути)

Шаг 4.1: ищем увеличивающий путь с помощью BFS.

Путь: $1 \to 2 \to 3 \to 4$

Этот путь теперь невозможен, так как остаточная пропускная способность на ребре $2 \to 3$ равна 0. Мы не можем передать поток через это ребро.

Шаг 5: Завершение

Мы не можем найти увеличивающий путь, так как остаточная пропускная способность на всех возможных путях из источника в сток стала нулевой.

Итоговый максимальный поток: 25

Псевдокод:

```
// Функция добавления ребра
Функция addEdge(u, v, cap):
    capacity[u][v] += cap  (u, v)

// Функция поиска в ширину (BFS)
Функция bfs(source, sink, parent):
    Создать список visited, размером V, и установить все значения в False
    Создать очередь q
    Добавить исходную вершину source в очередь q
    Установить visited[source] = True
```

```
    Установить parent[source] = -1

    Пока очередь q не пуста:
        Извлечь элемент u из очереди q
        Для каждого соседнего элемента v (где v от 0 до V-1):
            Если v не посещена и остаточная пропускная способность на ребре (u, v)
больше 0:
                Добавить вершину v в очередь q
                Установить visited[v] = True
                Установить parent[v] = u
                Если v == sink, вернуть True
    Вернуть False
// Функция загрузки данных из файла
void loadInput(const std::string& inputFile, std::string& base, long long& exponent){


    std::ifstream input(имяФайла)
    Если файл не открыт, выбросить ошибку
    input >> base >> exponent
    Закрыть файл
}

// Функция поиска максимального потока (Алгоритм Эдмондса-Карпа)
Функция edmondsKarp(source, sink):
    maxFlow = 0

    Пока есть увеличивающий путь (bfs вернул True):
        pathFlow = бесконечность

        Для каждого v от sink до source по пути parent:
            u = parent[v]
            pathFlow = min(pathFlow, capacity[u][v] - flow[u][v])

        Для каждого v от sink до source по пути parent:
            u = parent[v]
            flow[u][v] += pathFlow
            flow[v][u] -= pathFlow

        maxFlow += pathFlow  /

    Вернуть maxFlow  }
```

```
// функция для вывода графа в формате graphiz
Функция printGraphvizToFile(filename):
    Открыть файл filename для записи
    Записать в файл начало графа "digraph G {"

    Для каждой вершины u от 0 до V-1:
        Для каждой вершины v от 0 до V-1:
            Если пропускная способность на ребре (u, v) больше 0:
                Записать в файл ребро u -> v с метками пропускной способности и потока

    Записать в файл конец графа "}"
    Закрыть файл

// функция чтения графа из файла
Функция readGraphFromFile(filename, graph):
    Открыть файл filename для чтения
    Прочитать количество вершин V и количество рёбер E
    Инициализировать граф с V вершинами

    Для каждого рёбер (i от 0 до E-1):
        Прочитать вершины u, v и пропускную способность cap
        Добавить ребро с пропускной способностью cap между вершинами u и v в граф

    Закрыть файл

// Функция записи результатов в файл
Функция writeResultToFile(filename, maxFlow):
    Открыть файл filename для записи
    Записать в файл строку "Максимальный поток: " + maxFlow
    Закрыть файл

// Основная функция
Функция main:
    Установить локаль для правильного отображения русских символов
    Если аргументы командной строки некорректны:
        Вывести сообщение об ошибке и завершить программу

    Ввести исходную вершину и стоковую вершину (source и sink)
```

Инициализировать граф с нулевым количеством вершин

Прочитать граф из файла "input.txt" и загрузить его в граф

Вычислить максимальный поток с помощью edmondsKarp(source, sink)

Записать результат (максимальный поток) в файл "output.txt"

Сгенерировать файл для визуализации графа (формат dot) и записать его в "max_flow_network.dot"

Завершить выполнение

Сложность алгоритма: O(EV+E^2)), где E — количество ребер, а V – количество вершин.

ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ

На вход подается текстовый файл input.txt, на выходе файл otput.txt и max_flow_network.dot. Для того чтобы отрисовать решение нужно в терминале написать команду – dot -Tpng max_flow_network.dot -o max_flow_network.png.

ТЕСТОВЫЕ ПРИМЕРЫ

Тест 1

Входные данные:

```
4 5
0 1 15
0 3 20
1 2 10
1 3 25
2 3 5
```

```
Введите исходную вершину: 0
Введите стоковую вершину: 3
```

Выходные данные:

```
Максимальный поток: 35
```

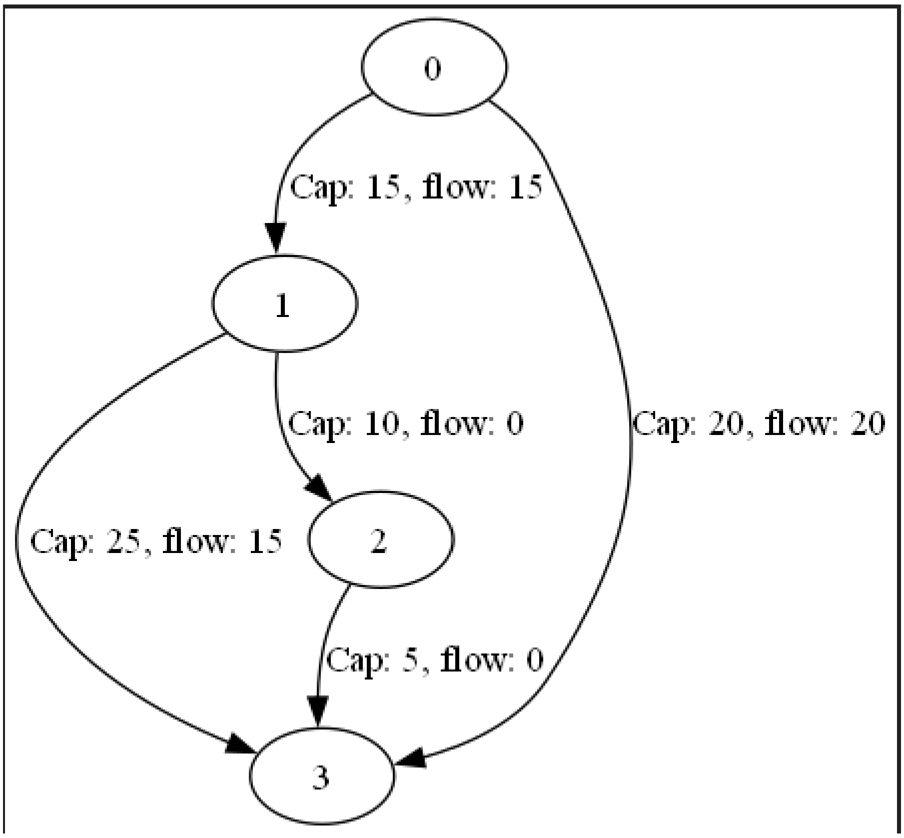

Рисунок 1.

Тест 2

Входные данные:

```
4 5
0 1 10
0 2 15
1 2 20
1 3 5
2 3 25
```

```
Введите исходную вершину: 0
Введите стоковую вершину: 3
```

Выходные данные:

```
Максимальный поток: 25
```

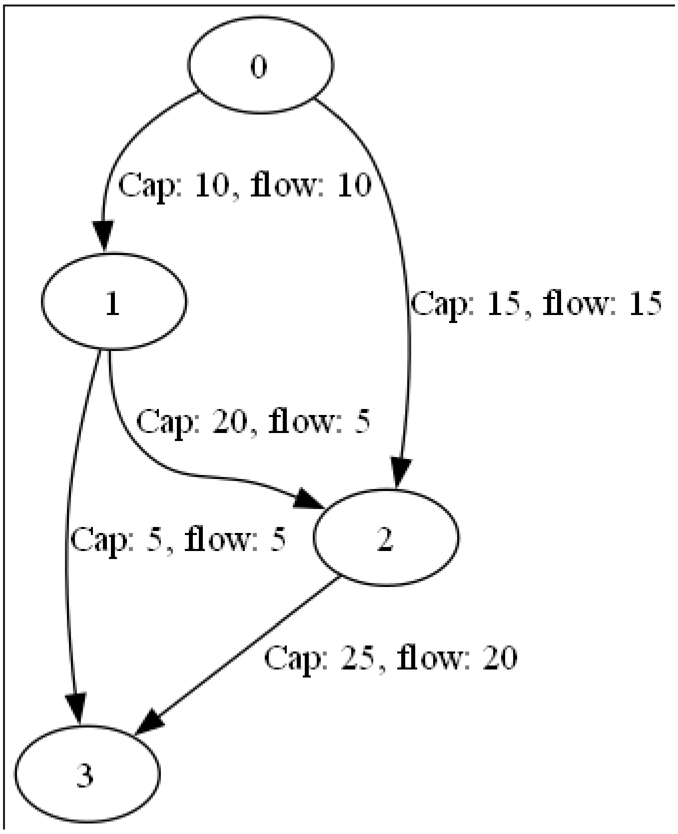

Рисунок 2.

Тест 3

Входные данные:

```
5 6
0 1 10
0 2 15
1 2 20
1 3 5
1 4 20
2 3 25
2 4 30
```

Выходные данные:

```
Введите исходную вершину: 0
Введите стоковую вершину: 4
```

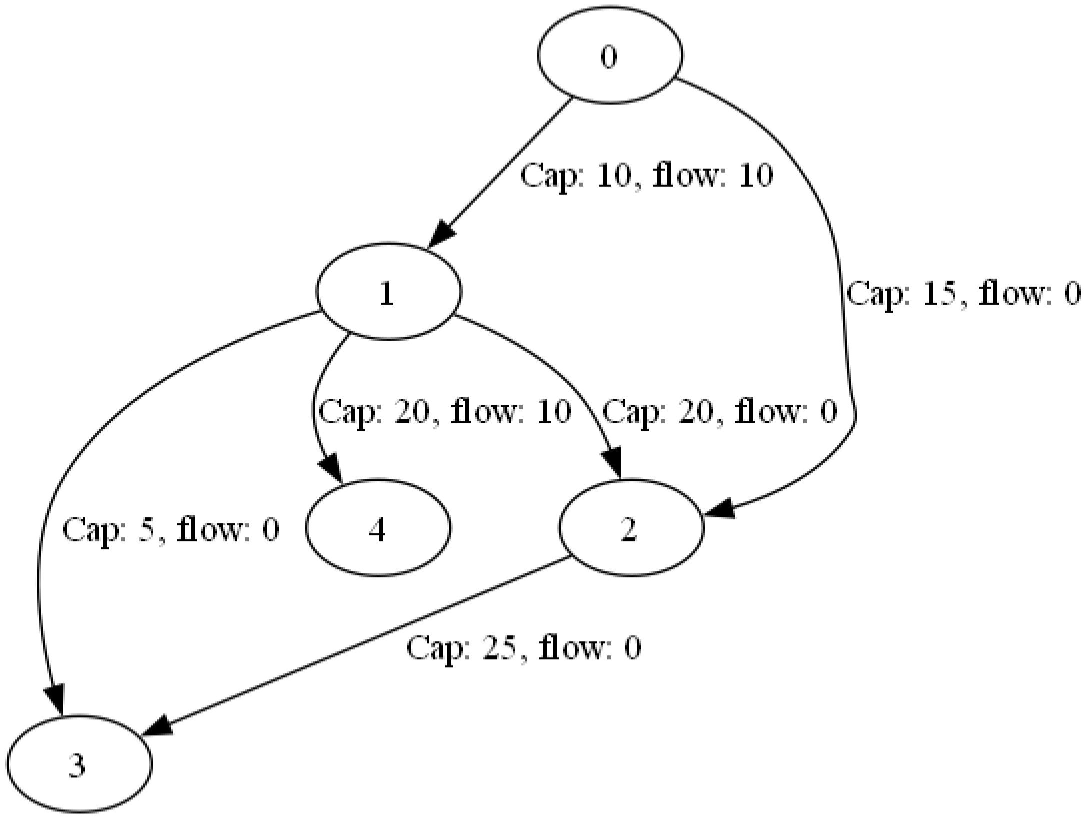
0
Cap: 10, flow: 10
Cap: 15, flow: 0
1
Cap: 20, flow: 10
Cap: 20, flow: 0
Cap: 5, flow: 0
4
2
Cap: 25, flow: 0
3

Рисунок 3.

СПИСОК ЛИТЕРАТУРЫ

- 1. А. Т. Кормен, Ч. Лейзерсон, Р. Ривест, “Алгоритмы: построение и анализ”