**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра САПР**

**ОТЧЕТ**

**по курсовой работе**

**по дисциплине «Алгоритмы и Структуры Данных»**

**Тема: «Потоки в сетях. Алгоритм Форда-Фалкерсона»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8301 | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | Бурундуков П. Г. |
| Преподаватель | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | Тутуева А. В. |

Санкт-Петербург

2020

**Цель**

Реализовать алгоритм, находящий максимальный поток в сети с помощью алгоритма Форда-Фалкерсона.

**Описание программы**

Идея алгоритма состоит в следующем.

Сначала ищется пункт, в который можно пойти из истока. Выбирается путь с наибольшей пропускной способностью, если таких несколько – выбирается первый найденный. Если такой путь найден, то пытаемся попасть в сток.

Если оказывается, что через выбранный путь нельзя дойти до стока, то путь ищется заново, при этом старый путь отмечается как посещённый, для того чтобы по нему нельзя было пройти ещё раз. Посещенные пункты сохраняютсядля последующей обработки.

После попадания в сток начинается обработка посещённых пунктов. Вначале находится наименьшая пропускная способность в пути и вычитается из остальных. Затем накапливается максимальный поток на основе пройденного пути.

Алгоритм завершается после того как были перепробованы все способы добраться до стока или если пропускная способность путей в сток становится равной 0.

В программе используются следующие структуры данных:

* Класс List:

Шаблонный двусвязный список. Используется для хранения посещённых пунктов. *Взят из-за простоты реализации(взят из прошлых лабораторных работ) и взятии данных из конца, можно было бы использовать и стэк, но список был уже реализован раньше.*

* Класс Map:

Обычный шаблонный ассоциативный массив. Используется для хранения индексов пункта и пропускной способности до него, путей, названий пунктов и их индексов. *Взят из-за эффективности Map в хранении пар данных. Благодаря этому снижается временная сложность вставки и доступа к данным(O(logN)) по сравнению с реализацией на списках(O(N))*.

* Класс Queue:

Шаблонная очередь. Используется для хранения строк, получаемых в текстовом файле, для последующей их обработки. *Была взята из-за удобства извлечения данных, так как данные позже обрабатывались в порядке вставки.*

* Класс flowStruct:

Внутренний класс, класса Graph. Используется в нескольких методах класса Graph. Имееттри поля: **checked** (проверка на то, посещался ли данный пункт), **channel\_width** (мап, хранящий пропускные способности между предыдущими пунктами), **ways** (мап пунктов, до которых можно добраться из данного пункта).

* Класс Graph:

Данный класс реализует поиск максимального потока в сети с помощью алгоритма Форда-Фалкерсона.

Этот класс хранит очередь file\_lines, список way\_to\_finished, и мапу items (хранит названия пунктов и их индекс).

Данный класс имеет два публичных метода. Первый – **take\_file**, который получает текстовый файл и передаёт его в метод считывающий его. Также запускает функцию создания графа. Второй метод – **get\_flowSize**, который возвращает максимальный поток, считающийся в методе **counting\_flow**.

Этот класс также имеет несколько приватных методов:

**read\_file**

Метод построчно считывает файл и сохраняет строки в очередь **file\_lines**. Также с помощью метода **get\_word** из строк выделяются пункты и сохраняются в мапе **items.** В качестве ключа сохраняются их названия, в качестве значения сохраняется их индивидуальный индекс, при этом индекс истока всегда 0, а стока – максимальный индекс.

**read\_line**

Метод получает на вход строку из очереди **file\_lines** и делит её на два пункта и пропускную способность, которая переводится в целочисленный тип данных.

**get\_word**

Метод получает из строки слова, находя его по разделительному знаку (пробел или символ окончания строки).

**сreate\_graph**

Метод создает массив **flow**, затем метод получает строки о пунктах и их пропускных способностях из очереди **file\_lines** и заполняет данными массив **flow** (если из пункта в пункт есть два пути, то пропускные способности суммируются). *Массив был взят из-за удобства использования, также он экономичен, благодаря его использованию не нужно создавать большую матрицу.*

**counting\_flow** и **create\_way**

Метод **counting\_flow** ищет максимальный поток.

Сначала ищется пункт, в который можно пойти из истока. Выбирается путь с наибольшей пропускной способностью (если таких несколько – выбирается первый найденный). Если такой путь найден, то вызывается рекурсивный метод **create\_way**.

Данный метод работает по аналогии с **counting\_flow**, но при этом рекурсивно пытается попасть в сток. Если оказывается, что через выбранный путь нельзя дойти до стока, то путь ищется заново (при этом по старому пути пойти нельзя, поскольку он отмечается как посещенный). Посещенные пункты помещаются в список **way\_to\_finished** для последующей его обработки.

После попадания в сток начинается обработкасписка **way\_to\_finished**. Сперва находится наименьшая пропускная способность и вычитается из остальных (если пропускная способность путей в сток становится равной 0, то поиск дальнейших путей прекращается). Затем накапливается максимальный поток на основе пройденного пути.

Таким образом, методы работают до тех пор, пока можно добраться от истока до стока, а затем возвращают максимальный поток.

**update\_flow**

Данный метод используется в **counting\_flow.** Возвращает *true*, если нет пути в сток, и *false* если путь есть. Также в этом методе увеличивается пропускная способность.

**Unit тесты**

Проверка исключения, вылетающего при пустом файле.

Проверка исключения, вылетающего при отсутствии истока или стока.

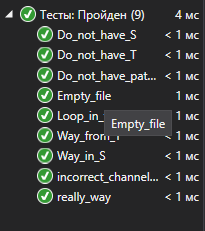
Проверка исключения, вылетающего при наличии петель.

Проверка исключения, вылетающего при некорректной пропускной способности.

Проверка исключения, вылетающего при входящих/выходящих путях в/из исток/сток.

Проверка работоспособности правильно заданных данных.

Проверка исключения, если исток и сток не связаны друг с другом.



**Временная сложность**

**take\_file** – O(1);(содержит **read\_file и create\_graph** )

**read\_file** – O();

**get\_word** – O();

**create\_ graph** – O();

**read\_line** – O(1);

**get\_flowSize** – O() (содержит **counting\_flow**);

**counting\_flow** – O().

**Примеры работы**

