­­­МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информатики и прикладной математики

Кафедра прикладной математики и экономико-математических методов

**КУРСОВАЯ РАБОТА**

по дисциплине:

**«Системы компьютерной математики»**

Тема: «Решение задачи о кратчайшей надстроке с помощью жадного алгоритма»

Направление: 01.03.02 Прикладная математика и информатика

Студент: Матвеева Валерия Владимировна

Группа: ПМ-2001 Подпись

Проверил: Фридман Григорий Морицович

Должность: д.т.н., профессор

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2021 г.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc73314546)

[1. ТОЧНЫЕ АЛГОРИТМЫ ДЛЯ РЕШЕНИЯ ЗАДАЧИ О КРАТЧАЙШЕЙ НАДСТРОКЕ 5](#_Toc73314547)

[1.1. Исчерпывающий алгоритм (An exhaustive algorithm) 5](#_Toc73314548)

[1.2. Целочисленное программирование (Integer programming) 5](#_Toc73314549)

[2. РЕАЛИЗАЦИЯ ЖАДНОГО АЛГОРИТМА. ВИЗУАЛИЗАЦИЯ РЕШЕНИЙ НА ГРАФЕ ПЕРЕКРЫТИЙ 8](#_Toc73314550)

[2.1. Постановка задачи и методика жадного алгоритма при решении задачи о кратчайшей надстроке. Реализация жадного алгоритма и функций, необходимых для его работы 8](#_Toc73314551)

[2.2. Отображение решений на графе перекрытий 12](#_Toc73314552)

[3. CВЕДЕНИЕ ЗАДАЧИ О КРАТЧАЙШЕЙ НАДСТРОКЕ К TSP 16](#_Toc73314553)

[ЗАКЛЮЧЕНИЕ 18](#_Toc73314554)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 19](#_Toc73314555)

# **ВВЕДЕНИЕ**

Задача о кратчайшей надстроке (The Shortest Superstring Problem) – это задача комбинаторной оптимизации, которая привлекла интерес многих исследователей из-за ее прикладного характера. Описание задачи: дан непустой набор строк . Необходимо найти кратчайшую строку, содержащую все строки из исходного набора. Пусть в исходном наборе ни одна строка не является подстрокой какой-либо другой строки из набора, в противном случае, можно преобразовать набор строк к набору без подстрок, просто удалив их.

Различные версии задачи о кратчайшей надстроке играют важную роль в сферах исследования и применяются в различных областях молекулярной биологии, например, при секвенировании ­­ДНК путем сборки фрагментов (нахождение последовательности генома) [9, 15] и компрессии вирусного генома [6, 7], и информатики, например, при сжатии и эффективном хранении данных [1]. Помимо этого, некоторые алгоритмы для решения SSP могут быть применены при планировании [10].

Описанная задача является NP-трудной [1], то есть может быть решена за полиномиальное время. Несмотря на это, было реализовано большое количество различных алгоритмов. Задача о кратчайшей надстроке не может быть решена эффективно и оптимально для относительно больших исходных данных, но ее можно решить с помощью приближенных алгоритмов, которые дают неплохой результат. Одним из лучших и широко применяющихся на практике в данный момент алгоритмов является жадный алгоритм (The Greedy Algorithm), выдающий надстроку, которая не более, чем в 2,5 раза длиннее оптимальной надстроки [7, 12]. В то же время существует до сих пор недоказанная гипотеза 30-летней давности о том, что жадный алгоритм является 2-приближенным.

Целью курсовой работы является решение задачи о кратчайшей надстроке с использованием жадного алгоритма.

Задачи курсовой работы:

1. реализация жадного алгоритма для решения задачи о кратчайшей надстроке;
2. визуализация решения на графе перекрытий;
3. выявление комбинаторной сущности задачи и сведение ее к задаче о коммивояжере (TSP).

# **ТОЧНЫЕ АЛГОРИТМЫ ДЛЯ РЕШЕНИЯ ЗАДАЧИ О КРАТЧАЙШЕЙ НАДСТРОКЕ**

Существует несколько точных алгоритмов для решения задачи о кратчайшей надстроке. Их мало из-за вычислительной сложности задачи и отсутствия необходимости в оптимальных решениях для основных применений в молекулярной биологии. Точные алгоритмы в основном используются для сравнения в вычислительных экспериментах с небольшим количеством экземпляров, а не в реальных практических задачах, где количество в приоритете.

Пусть перекрытие двух строк будет самой длинной строкой , такой что и для некоторых непустых , причем . Также пусть префикс слова относительно – строка в приведенном выше определении. Итак, . На самом деле - это кратчайшая строка, содержащая и в таком случае. Эта строка называется строкой слияния и и обозначается, как , а длина слияния соответственно - . Другими словами, строка слияния и - это объединение этих двух строк с учетом их перекрытия. Таким образом, можно обозначить формулу

|  |  |
| --- | --- |
| , | (1.1) |
|  |  |

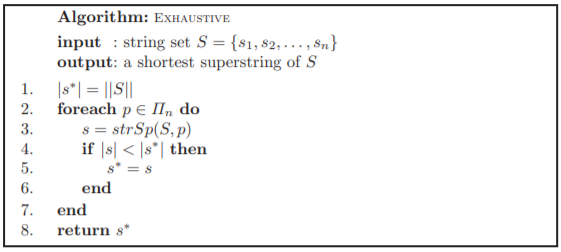
где длина слияния строк ;

длина перекрытия строк ;

длины строк соответственно.

## **Исчерпывающий алгоритм (An exhaustive algorithm)**

SSP может быть тривиально решена путем исчерпывающего перечисления всех возможных расположений строк. Слияние строк в некоторых из этих расположений соответствует самой короткой надстроке. Учитывая набор из строк, с помощью рассмотрения надстрок , соответствующих всем перестановкам, мы точно сможем найти самую короткую. Эта процедура описана в псевдокоде на рис. 1, где оптимальное решение.

  
Рисунок 1 – Исчерпывающий алгоритм [4]

Оптимальные решения для небольших экземпляров задачи о кратчайшей надстроке, полученные с помощью исчерпывающего алгоритма, приведены в [5, 6].

## **Целочисленное программирование (Integer programming)**

Для заданного набора из строк, рассмотрим связный граф перекрытий , где – вершины графа (строки), – ребра, а – функция, сопоставляющая каждой паре вершин длину их общего перекрытия. Здесь стоит упомянуть задачу поиска гамильтонова пути (Hamiltonian Path Problem) на взвешенном ориентированном графе, состоящую в нахождении оптимального гамильтонова пути в соответствии с его общим весом. Гамильтонов путь – это путь в неориентированном или ориентированном графе, при прохождении которого каждая вершина посещается ровно один раз. Если цель состоит в том, чтобы минимизировать общий вес, тогда рассматривается Min-HPP, а если цель состоит в максимизации общего веса, тогда рассматривается Max-HPP. Решение задачи нахождения гамильтонова пути на ориентированном графе требует существование гамильтонова пути на .

Аналогично имеют место задачи максимизации и минимизации гамильтонова цикла, которые также известны как задачи коммивояжера (Min-Travelling Salesman Problem и Max-TSP). Для обоих задач подразумевается, что они асимметричны, т.е. применяются к ориентированным графам, и вес дуги из вершины в вершину не обязательно равен весу дуги . И HPP, и TSP являются NP-трудными [9].

Оптимальное решение для набора может быть получено путем нахождения максимального гамильтонова пути на , поскольку найдя максимальный гамильтонов путь, все вершины (строки) будут помечены в таком порядке, что в итоге получится максимальное общее перекрытие. Из-за связи между Max-HPP и задачей о коммивояжере решение задачи о кратчайшей надстроке эквивалентно оптимальному решению задачи поиска максимального гамильтонова цикла на графе , где

* ;
* ;

Максимальный гамильтонов цикл на содержит максимальный гамильтонов путь на , который, в свою очередь, соответствует кратчайшей настроке набора . Следовательно, оптимальное решение задачи о кратчайшей надстроке может быть получено с помощью формулировки задачи коммивояжера, которая описана здесь в терминах графов и :

|  |  |  |
| --- | --- | --- |
|  | | (1.1.1) |
|  | | (1.1.2) |
|  | | (1.1.3) |
|  | (1.1.4) | |
|  | | (1.1.5) |
|  | | (1.1.6) |
|  | |  |

В формуле (1.1.1) указана целевая функция задачи, значение которой равно величине полученного перекрытия. Матрица перестановок – это матрица, в каждой строке и в каждом столбце которой только один элемент отличен от нуля и равен единице. Ограничения (1.1.2) - (1.1.4) указывают на то, что матрица должна быть матрицей перестановок без присваиваний на главной диагонали, а ограничения (1.1.5) - (1.1.6) – ограничения для нахождения максимального гамильтонова цикла. Дуговые ограничения (1.1.5) для пары вершин , , вынуждают , когда . Обратите внимание, что , соответствующее , не участвует ни в одном таком ограничении. Следовательно, если возможное решение содержало бы более одного цикла, то хотя бы один из этих циклов не содержал бы вершину , и в этом цикле значения привели бы к противоречию.

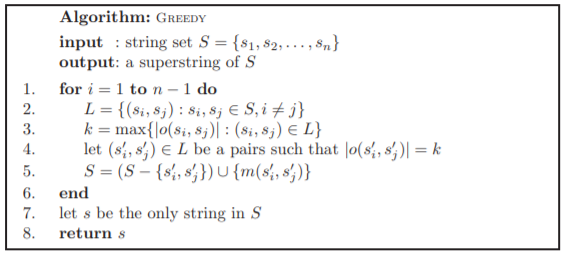
Многие задачи могут быть решены с помощью формулировки целочисленного программирования, в частности, могут быть реализованы алгоритмы для решения задачи коммивояжера и получения оптимальных результатов при решении задачи о кратчайшей надстроке.

# **РЕАЛИЗАЦИЯ ЖАДНОГО АЛГОРИТМА. ВИЗУАЛИЗАЦИЯ РЕШЕНИЙ НА ГРАФЕ ПЕРЕКРЫТИЙ**

## **Постановка задачи и методика жадного алгоритма при решении задачи о кратчайшей надстроке. Реализация жадного алгоритма и функций, необходимых для его работы**

Известный и широко изученный алгоритм для решения задачи о кратчайшей надстроке – жадный алгоритм, выдающий строку, длина которой (по жадной гипотезе) не более чем в 2 раза больше длины оптимальной надстроки [7, 12]. Более того, в некоторых случаях вычислительные эксперименты дают результаты намного лучше, чем это предположение [8]. Теоретически и экспериментально нет доказательств того, что любой другой из ныне существующих алгоритмов приближения в конечном итоге лучше, чем жадный.

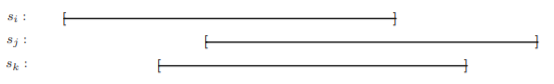
Подход к решению задачи о кратчайшей надстроке, реализованный с помощью жадного алгоритма, работает на удивление хорошо. На вход подается набор строк . На каждом шаге из выбираются две строки с наибольшим перекрытием и заменяются их слиянием . Алгоритм прекращает работу, когда остается только одна строка, и, очевидно, эта строка содержит все строки из исходного набора. Псевдокод жадного алгоритма представлен на рис. 2.

  
Рисунок 2 – Жадный алгоритм [4]

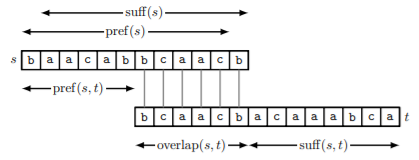
Следующая теорема показывает, что жадный алгоритм правильно определен согласно утверждению об отсутствии подстрок любых строк в исходном наборе:

На каждом шаге жадного алгоритма никакая строка не является частью строки в формирующемся наборе [2].

Доказательство основано на следующем наблюдении: если на каком-то этапе строка в (кроме уже объединенных) является подстрокой объединенной строки, то две выбранные строки не имеют наибольшего перекрытия, что противоречит жадному выбору. На рисунке 3 изображена ситуация, когда выбранными для слияния строками являются и , а строка , – подстрока , но не подстрока каждой из них.

  
Рисунок 3 – Расположение строк для жадного алгоритма [4]

При реализации жадного алгоритма необходимо каждый раз сравнивать перекрытия соответствующих пар строк, чтобы находить пару с максимальным перекрытием на каждом шаге. В строке любая подстрока является префиксом . По аналогии, любая подстрока называется суффиксом строки . На рисунке 4 можно увидеть изображение префиксов, суффиксов и перекрытия строк .

  
Рисунок 4 – Отображение префиксов, суффиксов и перекрытия строк [5]

Таким образом, наибольшее перекрытие – это самая длинная строка, которая является суффиксом одной строки и префиксом другой, соответственно, нахождение максимально возможных перекрытий и их длин в конечном счете сводится к сравнению суффиксов и префиксов пар строк из исходного набора.

Реализация алгоритма и функций, необходимых для его работы, выполнена на языке Wolfram Language [15].

В реализации жадного алгоритма используется функция maxOverlapLength, с помощью которой можно найти длину максимально возможного перекрытия 2-х строк. Код представлен на рис. 5. В коде используются такие встроенные функции системы Wolfram Mathematica, как Module, позволяющая локализировать все переменные внутри блока, и StringLength, выдающая длину аргумента-строки. Проходя по 2-м строкам с помощью цикла For, каждый раз берется префикс второй строки и сравнивается с суффиксом первой. На каждом шаге проверяется: если суффикс совпадает с префиксом, переменной, в которой будет храниться итоговая длина максимально возможного перекрытия, присваивается значение счетчика . Примеры работы функции maxOverlapLength приведены на рис. 6.

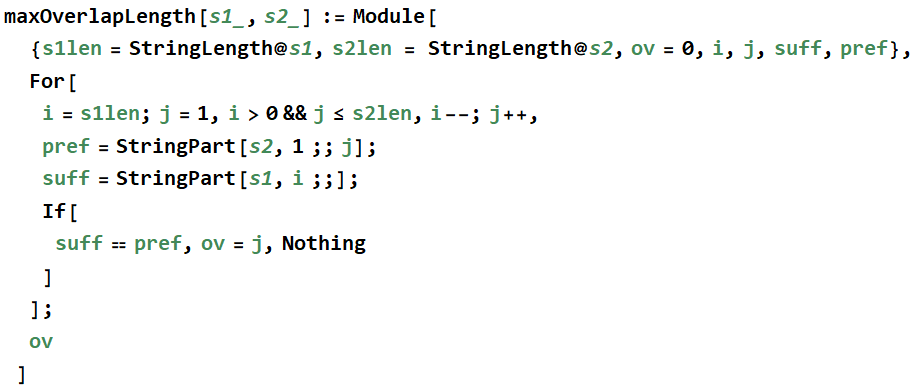


Рисунок 5 – Код с функцией maxOverlapLength

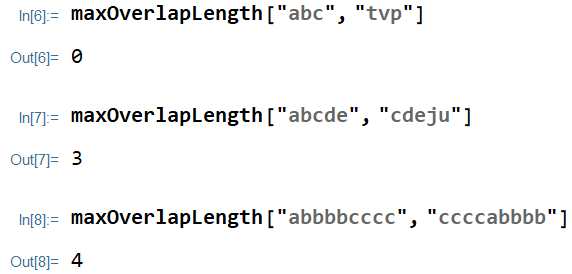


Рисунок 6 – Примеры работы функции maxOverlapLength

В основной функции solveSSP, принимающей на вход набор строк в виде списка, с помощью 2-х циклов ищется максимально возможное перекрытие пары строк (просматриваются все строки попарно, сравниваются длины перекрытий). На каждом шаге после нахождения строки с максимальным перекрытием формируется итоговая строка: с помощью функции ReplaceAll из набора удаляются строки и , далее добавляется строка слияния . Добавление с использованием функций AppendTo, StringJoin и StringPart. После этого обнуляется длина максимального перекрытия на исходном шаге. В конце работы алгоритма выдается 1 строка, являющаяся надстрокой всех строк в наборе.

Примеры работы функции solveSSP изображены на рис. 7, 8, 9.

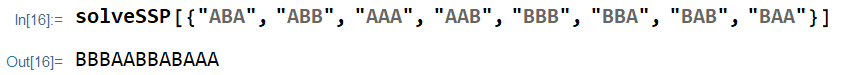


Рисунок 7 – Пример работы функции solveSSP

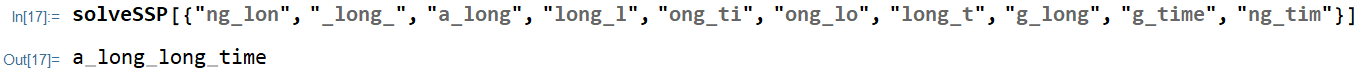


Рисунок 8 – Пример работы функции solveSSP

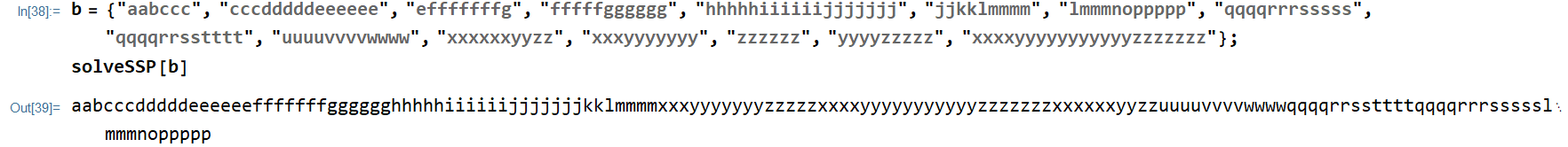


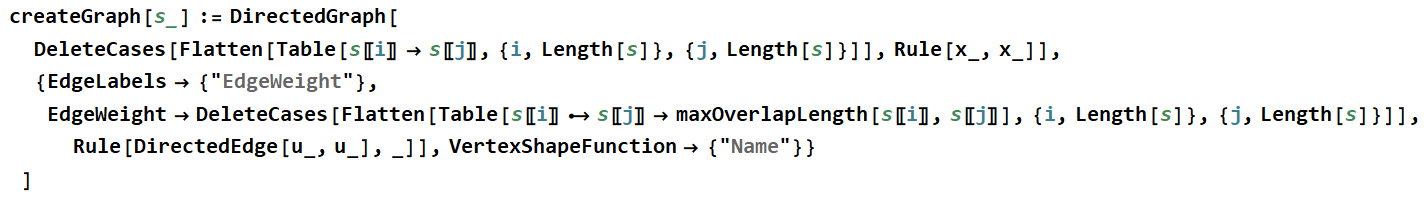
Рисунок 9 – Пример работы функции solveSSP

## **Отображение решений на графе перекрытий**

Каждому набору строк как уже упоминалось, можно поставить в соответствие полный взвешенный ориентированный граф перекрытий , где , , а весовая функция, сопоставляющая каждым 2-м вершинам (строкам) длину их общего перекрытия. Весом ребра, выходящего из вершины и заходящего в вершину , будем считать длину перекрытия строк , а вес ребра, выходящего из вершины и заходящего в вершину , будем считать длину перекрытия строк *,* т.е. порядок выбора строк для их объединения важен. Вес ребра – всегда положительное целое число. На примере: пусть имеется набор строк . В данном наборе , , а .

Визуализация решений на графах также выполнена при помощи реализации функций на языке Wolfram Language [15].

Функция createGraph принимает на вход набор строк и строит по нему полный ориентированный граф с множеством вершин и ребрами с длинной . Код, реализующий построение, представлен на рис. 10. Необходимо использование функции DirectedGraph, т.к. граф ориентированный. С помощью цикла создается список ребер от каждой вершины в другие вершины списка. Т.к. отображение петель («самоперекрытий») на графе не требуется, с помощью DeleteCases они исключаются. Далее, передавая дополнительные опции во 2 аргумент DirectedGraph, можно задавать вес ребер, который опять же создается с помощью цикла, пометки ребер, а также пометки вершин. Пример работы функции представлен на рисунке 11.

  
Рисунок 10 – Функция createGraph

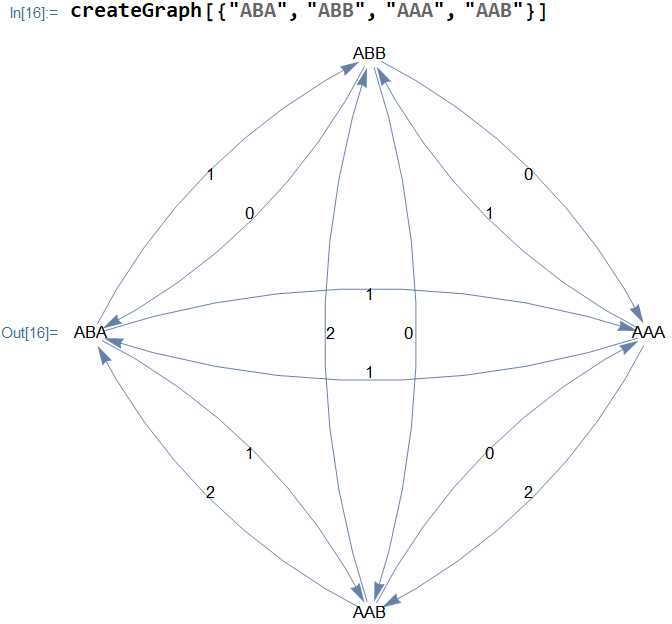
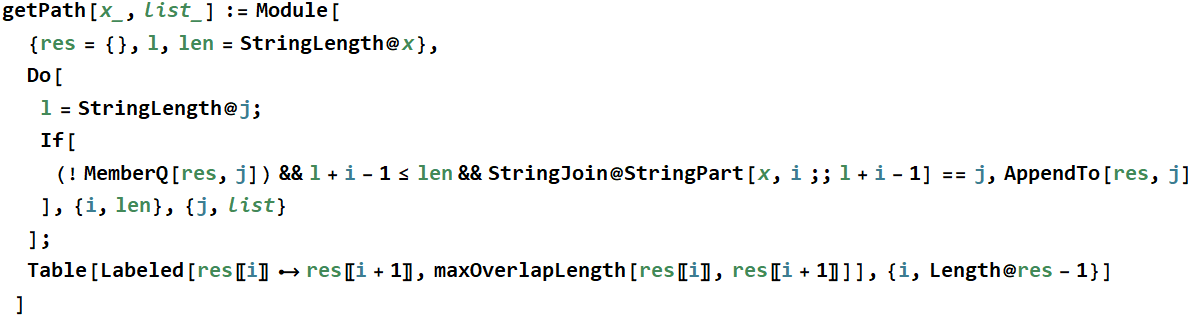


Рисунок 11 – Пример работы функции createGraph

Каждому решению задачи о кратчайшей надстроке можно сопоставить путь с максимальным итоговым перекрытием. Получение пути, соответствующего решению задачи о кратчайшей надстроке с помощью жадного алгоритма, реализовано внутри функции getPath, которая принимает на вход кратчайшую надстроку, полученную с помощью жадного алгоритма и исходный список строк. Код функции представлен на рис. 12. В цикле, перебирая поэлементно итоговую надстроку, рассматривая подстроки итоговой надстроки, проверяется наличие какой-либо подстроки в качестве строки исходного набора. Если такая строка находится, она добавляется в результирующий список, из которого потом строится итоговый путь. Также важно, чтобы строка входила в итоговый список не более 1 раза, так как в случае повторного добавления строки в список отображается повторное вхождение ребра в вершину, а такие случаи должны быть исключены. Проверка повторного вхождения строки в список обеспечивается с помощью функции MemberQ. Пример работы реализованной функции представлен на рисунке 13.

  
Рисунок 12 – Функция getPath

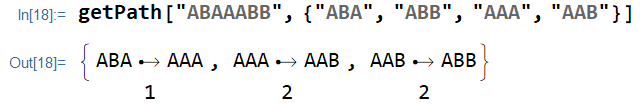
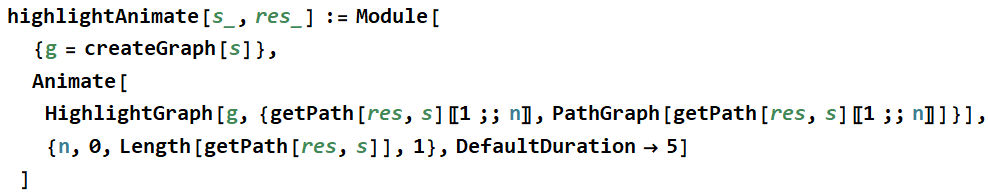
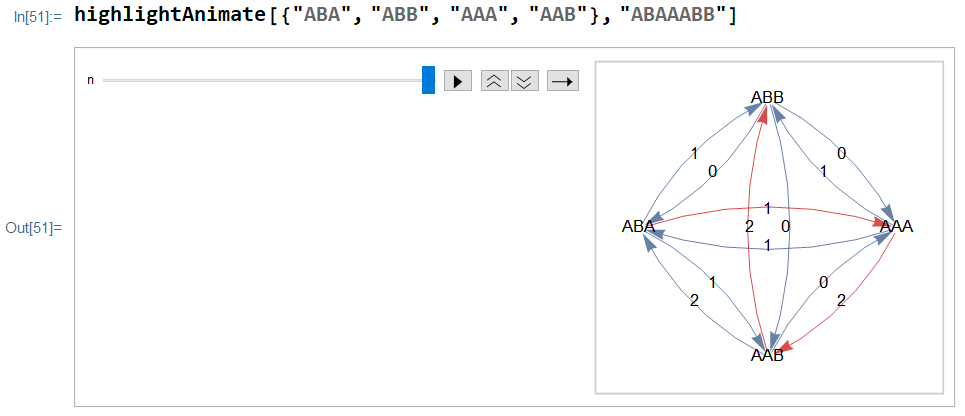


Рисунок 13 – Пример работы функции getPath

Получив данные о пройденном пути при решении задачи о кратчайшей надстроке и имея функцию построения графа по набору строк, можно визуализировать решение. Для этого была реализована еще одна функция highlightAnimate, использующая реализованные функции, описанные ранее. Код функции представлен на рис. 14. Изначально создаем граф перекрытий с помощью функции createGraph, далее отображаем необходимые ребра с использованием функции HighlightGraph, передавая в нее путь, определенный созданной функцией getPath. Помещая весь код внутрь функции Animate, анимируем прорисовку ребер на графе.

  
Рисунок 14 – Функция highlightAnimate

Итоговый результат работы функции, выдающей анимированный граф, можно увидеть на рис. 15.

  
Рисунок 15 – Визуализация решения задачи на графе

# **CВЕДЕНИЕ ЗАДАЧИ О КРАТЧАЙШЕЙ НАДСТРОКЕ К TSP**

Задача о кратчайшей надстроке является задачей о поиске оптимальной перестановки: чтобы найти кратчайшую строку, содержащую все , необходимо найти такую перестановку строк, что после соединения строк результирующее перекрытие будет максимальным. Очевидно, чтобы найти кратчайшую строку, содержащую все в заданном порядке, нужно просто перекрыть строки в этом порядке, как изображено на рисунке 16. Это простое наблюдение связывает задачу о кратчайшей надстроке с другими задачами перестановки, включая различные версии задачи коммивояжера.

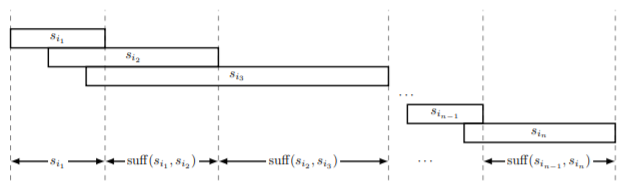


Рисунок 16 – Соотношение задачи о кратчайшей надстроке и задачи о перестановках [5]

Следует отметить, что соответствие между перестановками и надстроками не однозначно: настроки могут не соответствовать перестановкам. Например, катенация входных строк явно является надстрокой, но игнорирует тот факт, что соседние строки могут имеют нетривиальные перекрытия. Но по-прежнему ясно, что любая кратчайшая надстрока соответствует перестановке входных строк.

Следующие два ориентированных графа являются очень хорошими примерами того, как строки в перекрываются друг с другом. Граф перекрытия , который уже упоминался, и граф префиксов (также называемый графом расстояний), который определяется аналогично, только ребра имеют длину .

Пусть – строка , самая короткая строка, содержащая . Длина равна

|  |  |
| --- | --- |
| , | (3.1) |

что является длиной цикла в префиксном графе с добавлением В терминах графов перекрытий:

|  |  |
| --- | --- |
|  | (3.2) |
|  |  |

Вычитаемое в формуле (3.2) – путь в графе перекрытий , таким образом, самый длинный путь коммивояжера (Max-TSP) в графе перекрытия соответствует оптимальному решению для задачи о кратчайшей надстроки.

Помимо того, SSP может быть сведена к задаче о минимальном пути коммивояжера (Min-TSP): если в графе вес ребер будет не , а , то решение задачи о кратчайшей надстроки будет соответствовать пути с минимальным общим весом.

# **ЗАКЛЮЧЕНИЕ**

В ходе работы была изучена задача о кратчайшей надстроке, являющейся классической NP-трудной задачей, которая находит свое применение в различных областях науки. По мере изучения задачи были рассмотрены точные алгоритмы ее решения, а также приближенные алгоритмы, которые дают приемлемый результат за меньшее количество времени. Особое внимание было уделено жадному алгоритму, была разобрана его идея и методика, написана реализация в виде кода на языке Wolfram Language, на котором также были реализованы некоторые вспомогательные функции, необходимые для работы алгоритма. Была рассмотрена возможность решения поставленной задачи на графах, изучены функции построения графов в системе Wolfram Mathematica и реализована визуализация решений задачи на графе наложений.

Помимо этого, был проведен анализ связи задачи о кратчайшей настроке с другими перестановочными задачами: задачей о поиске оптимального гамильтонова пути в соответствии с его общим весом и задачей нахождения оптимального гамильтонова цикла (задача о коммивояжере). Выяснилось, что при рассмотрении на графе наложений, кратчайшая строка может быть найдена с помощью сведения данной задачи к различным версиям вышеупомянутых задач, причем разными способами. Более того, была выявлена комбинаторная сущность поставленной задачи и обнаружилось, что поставленная задача – задача нахождения такой перестановки строк, что в итоге получалась бы надстрока с максимальным результирующим перекрытием.

# **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Gallant J., Maier D., Storer J.A. On finding minimal length superstrings // Journal of Computer and System Sciences, 20(1), pp. 50 – 58 (1980)
2. Gevezes T., Pitsoulis L. A greedy randomized adaptive search procedure with path relinking for the shortest superstring problem // Journal of Combinatorial Optimization (2013)
3. Gevezes T. and Pitsoulis L. The shortest superstring problem // Optimization in Science and Engineering. Springer New York (2014)
4. Golovnev A., Kulikov A., Logunov A., Mihajlin I. and Nikolaev M. Collapsing Superstring Conjecture // Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019). // Leibniz International Proceedings in Informatics Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany (2019)
5. Ilie L., Popescu C. The shortest common superstring problem and viral genome compression // Fundam. Inf. 73, pp. 153–164 (2006)
6. Ilie L., Tinta L., Popescu C., Hill K.A. Viral genome compression // C. Mao, T. Yokomori (eds.) DNA Computing, Lecture Notes in Computer Science, vol. 4287, pp. 111–126. Springer Berlin / Heidelberg (2006)
7. Kaplan H., Lewenstwein M., Shafrir N. and Sviridenko M. Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs // Proc. 44th Annual IEEE Symp. on Foundations ­­of Computer Science (FOCS), p. 56 (2003)
8. Kaplan H., Shafrir N. The greedy algorithm for shortest superstrings // Inf. Process. Lett. 93, pp. 13–17 (2005)
9. Karp R.M. Reducibility Among Combinatorial Problems // In: R.E. Miller, J.W. Thatcher (eds.) Complexity of Computer Computations. Plenum Press. pp. 85–103 (1972)
10. Middendorf M. Shortest common superstrings and scheduling with coordinated starting times // Theoretical Computer Science, 191(1-2), pp. 205 – 214 (1998)
11. Miller C.E., Tucker A.W., Zemlin R.A. Integer programming formulation of traveling salesman problems // J. ACM 7, pp. 326–329 (1960)
12. Mucha M. A tutorial on shortest superstring approximation (2007)
13. Sweedyk Z. 2,5-approximation algorithm for shortest superstring // SIAM J. Comput., 29 (3), pp. 954–986 (2014)
14. Tarhio J., Ukkonen E. A greedy approximation algorithm for constructing shortest common superstrings // Theoretical Computer Science, 57(1), pp. 131 – 145 (1988)
15. Wolfram [Электронный ресурс] Wolfram – Режим доступа: https://www.wolfram.com/ свободный. Загл. с экрана – Яз. англ.