АННОТАЦИЯ

Пояснительная записка на «Теория симплициальных сетей и ее применения для анализа данных: Программная реализация библиотеки SimplicialX на языке C++» содержит следующие разделы: «Глоссарий», «Введение», «Назначение разработки», «Технические характеристики», «Технико-экономические показатели».

В разделе «Глоссарий» содержатся определения терминов и понятий, используемых в пояснительной записке.

В разделе «Введение» указано наименование и документ, на основе которого ведется разработка.

В разделе «Назначение разработки» указано функциональное и эксплуатационное назначение программного продукта.

Раздел «Технические характеристики» содержит следующие подразделы: «Постановка задачи на разработку программы», «Описание алгоритма и функционирования программы», «Организация входных данных», «Организация выходных данных», «Описание и обоснование выбора и состава технических и программных средств».

Раздел «Технико-экономические показатели» содержит ориентировочную экономическую эффективность, предполагаемую годовую потребность, экономические преимущества разработки программы.

Настоящий документ разработан в соответствии с требованиями:

1) ГОСТ 19.101-77 Виды программ и программных документов [1];
2) ГОСТ 19.102-77 Стадии разработки [2];
3) ГОСТ 19.103-77 Обозначения программ и программных документов [3];
4) ГОСТ 19.104-78 Основные надписи [4];
5) ГОСТ 19.105-78 Общие требования к программным документам [5];
6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом [6];
7) ГОСТ 19.201-78 Пояснительная записка. Требования к содержанию и оформлению [7].

Изменения к Пояснительной записке оформляются согласно ГОСТ 19.603-78 [8], ГОСТ 19.604-78 [9].

СОДЕРЖАНИЕ

ГЛОССАРИЙ

1) Граф — это совокупность вершин и ребер между ними.
2) Гиперграф – это обобщение графа, где рёбрами могут соединяться подмножества вершин.
3) Симплициальный комплекс – это вид гиперграфа, который состоит только из симплексов.
4) Симплекс размерности k (k-симплекс) состоит из k + 1 вершины, которые соединены попарно, кроме этого, он содержит все симплексы меньших размерностей на этих вершинах. Каждый симплекс обладает некоторым неотрицательным весом.
   Например, 0-симплекс – это просто вершина; 1-симплекс – это два 0-симплекса, которые соединены ребром; 2-симплекс – это три 1-симплекса, образующие треугольник.
   Получается, чтобы добавить k-симплекс в симплициальный комплекс, надо сначала убедиться, что присутствуют все его симплексы меньших размерностей, только после этого добавить новый.
5) Грань (граница) k-симплекса – любой симплекс меньшей размерности, который в нем присутствует.
6) Когрань (кограница) k-симплекса – любой симплекс большей размерности, содержащий симплекс в качестве грани.
7) (p, q)-граф – это граф, который строится следующим образом: его вершины – p-симплексы исходного комплекса, две вершины соединяются ребром, если есть q-симплекс такой, что оба p-симплекса его грани (когда $p < q$), оба p-симплекса его кограни (когда $p > q$). При этом вес ребра равен весу соответствующего q-симплекса.
8) Открытая звезда симплекса – это все симплексы, в которые симплекс входит в качестве грани, сам симплекс тоже входит в свою звезду.
9) Закрытая звезда симплекса – это минимальный подкомплекс, который содержит открытую звезду симплекса.
10) Линк симплекса – это все симплексы, которые входят в закрытую звезду симплекса и не входят в открытую звезду.

1. ВВЕДЕНИЕ

1.1. Наименование программы

Наименование программы – «Библиотека SimplicialX для C++».

Наименование программы на английском языке – «SimplicialX library for C++».

Краткое наименование программы – «SimplicialX».

1.2. Документы, на основании которых ведется разработка

Основанием для разработки является учебный план подготовки бакалавров по направлению 09.03.04 “Программная инженерия” и утвержденная академическим руководителем тема курсового проекта.

2. НАЗНАЧЕНИЕ РАЗРАБОТКИ

2.1. Функциональное назначение

Моделирование данных симплициальными комплексами и извлечение признаковых описаний.

2.2. Эксплуатационное назначение

В наше время всё больше задач решается с помощью машинного обучения и нейронных сетей. Важную роль при этом играет способ хранения информации, который позволит максимально эффективно с ней работать. Часто для этого используют графы, но у них есть один недостаток – в них отображены только отношения между парами объектов, это проблему помогают решить гиперграфы, где могут быть отношения на любых множествах вершин. Симплициальные комплексы – это частный случай гиперграфов, они облегчают работу с моделями, в которых есть не только отношения между парами объектов, но и попарные отношения на некоторых множествах.

3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

3.1. Постановка задачи на разработку программы

Библиотека должна позволять пользователю создавать, модифицировать, удалять симплициальные комплексы. При этом она должны поддерживать два способа хранения информации: с помощью симплекс дерева и с помощью диаграммы Хассе. Также пользователь может с помощью вызова соответствующих функций получать информацию о симплициальном комплексе и его симплексах: список всех симплексов с их весами, значения центральностей для вершин в (p, q)-графе, матрицу Лапласа, её спектр и т. д.

3.2. Описание алгоритма и функционирования программы

3.2.1. Общий алгоритм работы программы

Для работы с симплициальным комплексом пользователю после подключения библиотеки нужно создать новый объект класса SimplicialComplex, при этом он может указать предпочитаемую структуру для хранения информации, поменять её после этого будет нельзя. Обе структуры реализованы через свои классы, которые в свою очередь реализуют интерфейс SimplexInterface, в нём содержатся объявления всех методов, которые доступны пользователю. У объекта класса SimplicialComplex есть поле complex – указатель на объект класса SimplexInterface, который в конструкторе становится либо указателем на объект класса HasseDiagram (в нём реализована диаграмма Хассе), либо указателем на объект класса SimplexTrie (в нём реализовано симплекс дерево). Когда пользователь вызывает какой-либо метод объекта класса SimplicialComplex, соответствующий метод с теми же параметрами вызывается у переменной complex.

При помощи соответствующих методов пользователь может:

— добавлять симплекс в комплекс, менять вес симплекса, удалять симплекс (при этом также удаляются все его кограни). При удалении несуществующего симплекса ничего не происходит, как и при изменении веса несуществующего симплекса

— посчитать общее число симплексов, значения f-вектора (он хранит число симплексов каждой размерности), характеристику Эйлера (знакопеременная сумма значений f-вектора)

— найти степень p-симплексов в (p, q)-графе, расстояние между двумя p-симплексами в (p, q)-графе, closeness и betweenness центральности, коэффициент кластеризации графа
— получить граничную матрицу, матрицу Лапласа, собственные значения и собственные вектора матрицы Лапласа, числа Бетти
— вывести для симплекса его открытую звезду, закрытую звезду и линк.

При возникновении ошибок программа завершается с соответствующим сообщением.

3.2.2. Класс для реализации диаграммы Хассе HasseDiagram

В диаграмме Хассе каждому симплексу соответствует определённая вершина, при этом для каждой вершины хранится информация о расположении вершин-граней соответствующего симплекса и вершин-когранеи, при этом для k-симплекса нас интересуют только его грани размерности (k-1) и кограни размерности (k+1).
Чтобы не путать вершины комплекса и вершины диаграммы, отныне будем звать вершины диаграммы нодами (от английского слова node).
Каждая нода – это объект структуры HasseNode, которая хранит адреса нод-граней, адреса нод-когранеи, вес симплекса, его размерность, значение наибольшей вершины соответствующего симплекса (нода симплекса {1, 2, 3} имеет значение 3) и номер вершины (это надо для (p, q)-графа). Корнем диаграммы является специальная нода root, обозначающая пустой симплекс.
При изменениях симплициального комплекса создаются/удаляются/модифицируются определённые ноды. Для построения (p, q)-графа для всех p-симплексов находятся все подходящие q-симплексы (для этого просто запускается обход по ребрам между нодами), потом по этой информации в графе проводятся ребра между вершинами. Предварительно все ноды нумеруются, чтобы удобнее было создавать граф: если нам надо провести ребро между симплексами с номерами нод 2 и 3, то мы в нашем (p, q)-графе находим вершины 2 и 3 и уже между ними проводим ребро. Для построения открытой звезды симплекса мы просто от его ноды запускаем обход диаграммы по ребрам, отвечающим за когранеи.
Для построения закрытой звезды мы из всех симплексов открытой звезды находим те, у которых нет когранеи, запоминаем все вершины этих симплексов и потом из нод, соответствующим этим вершинам запускаем обход диаграммы по ребрам, отвечающим за когранеи, но при этом мы не

заходим в ноды, которые отвечают за симплексы, содержащие вершину не из нашего найденного списка. Чтобы построить линк, находим симплексы, которые попадают только в закрытую звезду.

3.2.3. Класс для реализации симплекс дерева SimplexTrie

В симплекс дереве каждому симплексу соответствует определённая нода (в данном контексте нода – это вершина симплекс дерева). Нода хранит значение наибольшей вершины соответствующего симплекса, размерность, адреса своих детей, адрес родителя, вес симплекса и номер вершины. Само дерево является обычным бором, в который добавляются всевозможные подмассивы полученного симплекса (симплекс поступает как vector<int>), то есть при добавлении в пустое дерево симплекса {1, 2}, мы добавим два ребра от корня (здесь это тоже специальная вершина, обозначающая пустой симплекс): одно к ноде со значением 1 (она обозначает симплекс {1}), от этой ноды будет ребро к ноде со значением 2 (эта нода обозначает симплекс {1, 2}), второе ребро от корня будет к другой ноде со значением 2 (она обозначает симплекс {2}). Получается, чтобы по ноде восстановить симплекс, который она обозначает, надо просто подняться до корня.

Также все ноды на одинаковой глубине с одинаковым значением объединены в списки, это помогает при поиске когранеи.

Удаление происходит по алгоритму, предложенному в статье [10]. Чтобы построить (p, q)-граф, нужно сначала пройти по дереву, найти все p-симплексы и q-симплексы, потом для каждого найденного p-симплекса запустить поиск подходящих q-симплексов, а для q-симплексов – поиск подходящих p-симплексов, потом по этой информации проводим ребра. Сам поиск устроен сложнее, чем в диаграмме Хассе – чтобы найти грань нужной размерности, нужно получить список всех вершин в нашем симплексе, потом рассмотреть все варианты выбрать из них нужное число вершин и найти такой симплекс в нашем дереве. Чтобы получить когрань нужной размерности, нужно просто из всех когранеи выбрать те, у которых подходящая размерность. Поиск когранеи описан в [10]. Как и в диаграмме Хассе, при построении графа все ноды нумеруются.

Для построения открытой звезды симплекса мы находим все его когранницы, как в случае его удаления. Для построения закрытой звезды мы из всех симплексов открытой звезды находим те, у

которых нет когранич, запоминаем все вершины этих симплексов и потом из нод, отвечающих за эти вершины, запускаем обход в ширину нашего дерева, но при этом мы не заходим в ноды, которые отвечают за симплексы, содержащие вершину не из нашего найденного списка. Чтобы построить линк, находим симплексы, которые попадают только в закрытую звезду.

3.2.4. (P, Q) - граф

Для нахождения степеней вершин строится граф и для каждой вершины считается число исходящих из неё ребер. Для нахождения расстояния между двумя симплексами, строится граф, затем в нём находятся вершины, которые отвечают за ноды, обозначающие заданные симплексы, затем запускается алгоритм Дейкстры. Чтобы посчитать центральности из каждой вершины полученного графа запускается алгоритм Дейкстры, после чего происходит вычисление её соответствующей центральности. Чтобы найти коэффициент кластеризации, в графе находится число A = 3 * число треугольников (троек вершин, которые попарно соединены) и число B = сумма по всем вершинам величины d_2, где d_2 = число способов выбрать двух соседей вершины. Тогда Коэффициент кластеризации = A/B.

Программа распространяется в виде электронного пакета, содержащего программную документацию, приложение (исполняемые файлы и прочие необходимые для работы файлы).

3.2.5. Граничная матрица, матрица Лапласа и числа Бетти

Граничная матрица – это матрица граничного оператора, который переводит некоторый k-симплекс в линейную комбинацию его p-симплексов, где p < k. Этот оператор вводится в статье [11]. Зная граничную матрицу, можно получить матрицу Лапласа (про формулу и нужные преобразования написано в статье [11]) и числа Бетти. Для получения чисел Бетти нужно предварительно привести граничную матрицу к нормальной форме Смита, об этом написано в статье [12]. Зная матрицу Лапласа, с помощью библиотеки armadillo можно легко получить собственные числа и собственные вектора матрицы, что помогает при анализе симплициального комплекса.

3.3. Организация входных данных

Для каждого метода класса SimplicialComplex определён свой ряд входных параметров, у некоторых методов число входных параметров может меняться, потому что допускается использование значений по умолчанию.

3.4. Организация выходных данных

Выходные данные зависят от вызываемых методов:

— методы insert, erase и changeWeight ничего не выводят и ничего не возвращают

— метод simlexCount возвращает число симплексов типа int

— метод eulerNumber возвращает Эйлерову характеристику типа int

— метод fVector ничего не возвращает и выводит число симплексов каждой размерности, пока не встретит размерность, которой нет в комплексе

— метод distancePQ возвращает расстояние между двумя симплексами типа double

— метод clusterCoeff возвращает коэффициент кластеризации (p, q)-графа типа double

— метод allSimplices выводит все симплексы и их веса, ничего не возвращает

— методы closeness, betweenness и vertexDegreePQ для всех вершин в (p, q)-графе выводят сначала симплекс, который обозначает вершина, а потом соответствующее ей посчитанное значение

— методы openStar, closeStar, link выводят все подходящие симплексы

— методы boundaryMatrix, laplacianMatrix, laplacianMatrixWeight возвращают посчитанные матрицы типа arma::mat из библиотеки armadillo

— метод laplacianSpectre возвращает пару значений, где первая часть пары – это вектор собственных значений матрицы типа arma::vec из библиотеки armadillo, а вторая часть – матрица собственных векторов типа arma::mat

— метод bettiNumbers возвращает вектор со значениями чисел Бетти типа arma::vec.

3.5. Описание и обоснование выбора и состава технических и программных средств

3.5.1. Состав технических и программных средств

3.5.1.1. Состав технических средств

Для функционирования библиотеки рекомендуется следующий состав технических средств:

— мышь или совместное указывающее устройство

— клавиатура

— монитор

— 64 МБ оперативной памяти

— Процессор: Intel Pentium 4 / Athlon 64 или новее

3.5.1.2. Состав программных средств

Библиотека написана на языке C++17

В коде используются:

— Стандартная библиотека C++

— Библиотека для операций с матрицами и векторами armadillo

— Открытый стандарт OpenMP для распараллеливания некоторых участков кода

Для работы с библиотекой пользователю нужна среда разработки кода на языке C++, которая поддерживает OpenMP. Также ему нужно установить библиотеку armadillo.

3.5.2. Обоснование выбора технических и программных средств

Язык C++ был выбран, потому что он является одним из самых быстрых и эффективных языков программирования. Библиотека armadillo очень проста в использовании и позволяет выполнять все необходимые операции. OpenMP поддерживается Visual C++, GCC, LLVM и др.

4. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

4.1. Ориентировочная экономическая эффективность

В рамках данного курсового проекта экономическая эффективность не предусмотрена.

4.2. Предполагаемая потребность

Данная библиотека будет интересна пользователям, которые занимаются топологическим анализом данных и машинным обучением.

4.3. Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами

Таблица 1 – Сравнение функциональности ПО для работы с симплициальными комплексами

	Giotto-tda	BATS	Dionysus	SimplicialX
Числа Бетти	-	-	-	+
Эйлерова характеристика	-	-	-	+
Оператор Лапласа высших порядков	-	+	-	*
Другие признаки	-	-	-	+

* - оператор Лапласа между несмежными пространствами цепей

Как видно из таблицы, библиотека SimplicialX лучше аналогов, потому что поддерживает больше функций для анализа симплициальных комплексов.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. ГОСТ 19.101-77 Виды программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2. ГОСТ 19.102-77 Стадии разработки. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3. ГОСТ 19.103-77 Обозначения программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4. ГОСТ 19.104-78 Основные надписи. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5. ГОСТ 19.105-78 Общие требования к программным документам. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

7. ГОСТ 19.201-78 Пояснительная записка. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8. ГОСТ 19.603-78 Общие правила внесения изменений. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
9. ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
10. The Simplex Tree: an Efficient Data Structure for General Simplicial Complexes [Электронный ресурс]: Режим доступа https://arxiv.org/pdf/2001.02581.pdf, свободный. (дата обращения: 11.05.2023)
11. Simplicial degree in complex networks. Applications of topological data analysis to network science [Электронный ресурс]: Режим доступа https://arxiv.org/pdf/1908.02583.pdf, свободный. (дата обращения 11.05.2023)
12. Matrix Reduction. [Электронный ресурс]: Режим доступа https://courses.cs.duke.edu/fall06/cps296.1/Lectures/sec-IV-3.pdf, свободный. (дата обращения 11.05.2023)

ПРИЛОЖЕНИЕ 1

ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ КЛАССОВ

Таблица 2. Описание и функциональное назначение классов

Класс	Назначение
SimpInterface	Является абстрактным классом, который наследуют классы HasseDiagram и SimplexTrie. В нём объявляются все методы, доступные пользователю.
HasseDiagram	Реализует диаграмму Хассе и переопределяет все методы SimpInterface
SimplexTrie	Реализует симплекс дерево и переопределяет все методы SimpInterface
SimplicialComplex	Содержит в себе указатель на объект класса SimpInterface, который при создании становится указателем на объект класса SimplexTrie или HasseDiagram. Объект этого класс создаёт пользователь и с ним взаимодействует.

ПРИЛОЖЕНИЕ 2

ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ

Таблица 3.1. Описание и функциональное назначение методов класса SimpInterface

Метод	Тип возвращаемого знач.	Аргументы	Назначение
insert	void	vector<int>&, double = 1	Объявление метода добавления
erase	void	vector<int>&	Объявление метода удаления
changeWeight	void	vector<int>&, double = 1	Объявление метода изменения веса
simplexCount	int	—	Объявление метода подсчёта симплексов
fVector	void	—	Объявление метода нахождения f-вектора
vertexDegreePQ	void	int, int	Объявление метода подсчёта степеней вершин в (p, q)-графе
allSimplices	void	—	Объявление метода для вывода всех симплексов
closeness	void	int, int	Объявление метода для подсчёта closeness центральности вершин в (p, q)-графе

betweenness	void	int, int	Объявление метода для подсчёта bbetweenness центральности вершин в (p, q)-графе
distancePQ	double	vector<int>&, vector<int>&, int, int	Объявление метода для нахождения расстояние между двумя симплексами
clusterCoeff	double	int, int	Объявление метода для нахождения коэфф. кластеризации (p, q)-графа
eulerNumber	int	—	Объявление метода для подсчёта Эйлеровой хар-ки
boundaryMatrix	arma::mat	int=1, int=1	Объявление метода для нахождения граничной матрицы
laplacianMatrix	arma::mat	int, int=1, int=1	Объявление метода для нахождения матрицы Лапласа без учёта весов
laplacianMatrixWeight	arma::mat	int, int=1, int=1	Объявление метода для нахождения матрицы Лапласа с учётом весов
laplacianSpectre	pair<arma::vec, arma::mat>	int, int=1, int=1, bool=false	Объявления метода для нахождения спектра матрицы Лапласа

openStar	void	vector<int>&	Объявление метода для нахождения открытой звезды
closeStar	void	vector<int>&	Объявление метода для нахождения закрытой звезды
link	void	vector<int>&	Объявление метода для нахождения линка
bettiNumbers	arma::vec	—	Объявление метода для нахождения чисел Бетти

Таблица 3.2. Описание полей структуры HasseNode

Поле	Тип	Назначение
value	int	Значение максимальной вершины симплекса
depth	int	Размерность симплекса
number	int	Номер симплекса при построении (p, q)-графа
weight	double	Вес симплекса
faces	map<int, HasseNode*>	Словарь с адресами всех граней симплекса, у которых размерность на 1 меньше
cofaces	map<int, HasseNode*>	Словарь с адресами всех кограней симплекса, у которых размерность на 1 больше

Таблица 3.3. Описание полей структуры TrieNode

Поле	Тип	Назначение
value	int	Значение максимальной вершины симплекса
depth	int	Размерность симплекса
number	int	Номер симплекса при построении (p, q)-графа
weight	double	Вес симплекса
parent	TrieNode*	Адрес родителя ноды в дереве
children	map<int, TrieNode*>	Словарь с адресами всех кограней симплекса, у которых размерность на 1 больше

Таблица 3.4. Описание полей класса HasseDiagram

Поле	Тип	Назначение
root	HasseNode*	Адрес корня диаграммы
f	int[]	Массив со значениями f-вектора
nums	vector<HasseNode*>	Массив с адресом ноды для каждой вершины (p, q)-графа
translation	vector<int>	Массив соответствий номера ноды после нумерации и номера вершины в (p, q)-графе

Таблица 3.5. Описание полей класса SimplexTrie

Поле	Тип	Назначение
root	TrieNode*	Адрес корня дерева
f	int[]	Массив со значениями f-вектора
nums	vector<TrieNode*>	Массив с адресом ноды для каждой вершины (p, q)-графа
translation	vector<int>	Массив соответствий номера ноды после нумерации и номера вершины в (p, q)-графе
depth_lists	vector<vector<list<TrieNode*>>>	Списки, в которых объединены ноды на одной глубине и с одинаковым значением вершины

Таблица 3.6. Описание полей класса SimplicialComplex

Поле	Тип	Назначение
complex	SimpInterface*	Выполняет все вызванные методы

Таблица 3.7. Описание и функциональное назначение методов класса HasseDiagram

Метод	Тип возвращаемого знач.	Аргументы	Назначение
makePretty	void	vector<int>&	В полученном списке вершин удаляет дубликаты и сортирует

			вершины
dfs	void	HasseNode*	Рекурсивная функция с удалением ноды и всех её кограней
dfsClose	void	HasseNode*, set<int>&	Рекурсивная функция для нахождения кограней симплекса, у которых нет кограней
bfs	void	HasseNode*, vector<HasseNode*>&	Обход в ширину диаграммы, начиная с указанной ноды
openStar	void	vector<int>&	Выводит симплексы из открытой звезды
closeStar	void	vector<int>&	Выводит симплексы из закрытой звезды
link	void	vector<int>&	Выводит симплексы из линка
numeration	void	HasseNode*, int&	Нумерует ноды диаграммы
erase	void	vector<int>&	Удаляет полученный симплекс
changeWeight	void	vector<int>&, double=1	Меняет вес полученного симплекса на указанный

insert	void	vector<int>&, double=1	Добавляет симплекс с указанным весом
simplexList	vector<HasseNode*>	—	Возвращает массив с адресами всех нод
fVector	void	—	Выводит значения f-вектора
simplexCount	int	—	Возвращает количество симплексов
simplexFinder	HasseNode*	vector<int>&	Возвращает адрес ноды, которая соотв. указанному симплексу
dimensionFinder	void	HasseNode*, int, vector<HasseNode*>&	Находит все симплексы нужной размерности
simplexFinderUpgrade	HasseNode*	vector<int>&, int	Возвращает адрес ноды, которая соотв. указанному симплексу с учётом наложенной маски
findingNeighbours	void	HasseNode*, int, vector<HasseNode*>& vector<int>&, int	Находит соседей нужной размерности для построения (p, q)-графа
	void	int, int	Выводит

vertexDegreePQ			степени вершин в (p, q)-графе
graphPQBuilder	vector<vector<pair<int, double>>>	int, int	Строит (p, q)-граф
dijkstra	vector<pair<double, int>>	int, vector<vector<pair<int, double>>>	Находит расстояния от заданной вершины до всех остальных в (p, q)-графе
dijkstraSpecial	double	int,int, vector<vector<pair<int, double>>>	Находит расстояния от заданной вершины до другой заданной в (p, q)-графе
distancePQ	double	vector<int>&, vector<int>&, int, int	Находит расстояние между двумя симплексами в (p, q)-графе
closeness	void	int, int	Выводит closeness центральности в (p,q)-графе
betweenness	void	int, int	Выводит betweenness центральности в (p,q)-графе
clusterCoeff	double	int, int	Возвращает коэффициент кластеризации для (p, q)-графа
eulerNumber	int	—	Возвращает Эйлерову хар-

			ку
printSimplex	void	HasseNode*	Выводит вершины и вес симплекса, которому соотв. нода
allSimplices	void	—	Выводит все симплексы и их веса
rec	void	int, int, int, int, vector<int>&, int	Генерирует все комбинации выбрать k чисел из n чисел (от 0 до n-1)
combinations	vector<int>	int, int	Возвращает все уникальные варианты выбрать k чисел из n чисел (от 0 до n-1)
perm_sign	double	vector<int>&	Вычисляет знак перестановки
boundaryMatrix	arma::mat	int=1, int=1	Возвращает граничную матрицу
boundaryMatrixNoPer	arma::mat	vector<HasseNode*>& vector<HasseNode*>& int, double	Возвращает граничную матрицу, когда размерности отличаются только на 1
laplacianMatrix	arma::mat	int,int=1,int=1	Возвращает матрицу Лапласа без учета весов

laplacianMatrixWeight	arma::mat	int,int=1,int=1	Возвращает матрицу Лапласа с учетом весов
simplexWeights	arma::mat	Int k	Возвращает матрицу весов для симплексов определённой размерности
laplacianSpectre	pair<arma::vec, arma::mat>	int,int=1,int=1, bool=false	Возвращает вектор собственных чисел и матрицу собственных векторов матрицы Лапласа с весом/без веса
bettiNumbers	arma::vec	void	Возвращает вектор с числами Бетти
reduce	arma::mat	arma::mat, int=0	Рекурсивная функция для приведения матрицы к нормальной форме Смита

Таблица 3.8. Описание и функциональное назначение методов класса SimplexTrie

Метод	Тип возвращаемого знач.	Аргументы	Назначение
makePretty	void	vector<int>&	В

			полученном списке вершин удаляет дубликаты и сортирует вершины
numeration	void	TrieNode*, int&	Нумерует ноды диаграммы
addToList	void	TrieNode*	Добавляет ноду в список, основываясь на значении и глубине
deleteFromList	void	TrieNode*	Удаляет ноду из списка, основываясь на значении и глубине
goodPath	bool	TrieNode*, vector<int>&, int	Проверяет, содержит ли симплекс, на который указывает нода, полученный симплекс как грань
dfs	void	TrieNode*	Рекурсивная функция с удалением ноды и её поддерева
bfsFinder	void	TrieNode*, vector<int>&, vector<TrieNode*>&	Функция с обходом в ширину дерева для построения открытой звезды
openStar	void	vector<int>&	Выводит симплексы из

			открытой звезды
closeDFS	void	TrieNode*, set<int>&	Рекурсивная функция для нахождения когранейсимплекса, у которых нет когранейй
closeStar	void	vector<int>&	Выводит симплексы из закрытой звезды
link	void	vector<int>&	Выводит симплексы из линка
erase	void	vector<int>&	Удаляет полученный симплекс
changeWeight	void	vector<int>&, double=1	Меняет вес полученного симплекса на указанный
insertRecursive	void	vector<int>&, TrieNode*, int, double	Рекурсивная функция для добавления симплекса в дерево
insert	void	vector<int>&, double=1	Добавляет симплекс с указанным весом
simplexList	vector<TrieNode*>	—	Возвращает массив с адресами всех нод
fVector	void	—	Выводит значения f-

			вектора
simplexCount	int	—	Возвращает количество симплексов
simplexFinder	TrieNode*	vector<int>&	Возвращает адрес ноды, которая соотв. указанному симплексу
dimensionFinder	void	TrieNode*, int, vector<int>&	Находит все симплексы нужной размерности, запоминая их номера
dimensionFinderNodes	void	TrieNode*, int, vector<TrieNode*>&	Находит все симплексы нужной размерности, запоминая адреса их нод
bitsCount	int	int	Считает число бит со значением 1 в двоичной записи числа
simplexFinderUpgrade	TrieNode*	vector<int>&, int	Возвращает адрес ноды, которая соотв. указанному симплексу с учётом наложенной маски
findingNeighbours	void	TrieNode*, int, vector<TrieNode*>&	Находит соседей нужной размерности для

			построения (p, q)-графа
vertexDegreePQ	void	int, int	Выводит степени вершин в (p, q)-графе
graphPQBuilder	vector<vector<pair<int, double>>>	int, int	Строит (p, q)-граф
dijkstra	vector<pair<double, int>>	int, vector<vector<pair<int, double>>>	Находит расстояния от заданной вершины до всех остальных в (p, q)-графе
dijkstraSpecial	double	int,int, vector<vector<pair<int, double>>>	Находит расстояния от заданной вершины до другой заданной в (p, q)-графе
distancePQ	double	vector<int>&, vector<int>&, int, int	Находит расстояние между двумя симплексами в (p, q)-графе
closeness	void	int, int	Выводит closeness центральности в (p,q)-графе
betweenness	void	int, int	Выводит betweenness центральности в (p,q)-графе
clusterCoeff	double	int, int	Возвращает коэффициент кластеризации

			для (p, q)-графа
eulerNumber	int	—	Возвращает Эйлерову хар-ку
printSimplex	void	TrieNode*	Выводит вершины и вес симплекса, которому соотв. нода
allSimplices	void	—	Выводит все симплексы и их веса
rec	void	int, int, int, int, vector<int>&, int	Генерирует все комбинации выбрать k чисел из n чисел (от 0 до n-1)
combinations	vector<int>	int, int	Возвращает все уникальные варианты выбрать k чисел из n чисел (от 0 до n-1)
perm_sign	double	vector<int>&	Вычисляет знак перестановки
boundaryMatrix	arma::mat	int=1, int=1	Возвращает граничную матрицу
boundaryMatrixNoPer	arma::mat	vector<TrieNode*>& vector<TrieNode*>& int, double	Возвращает граничную матрицу, когда размерности отличаются только на 1
laplacianMatrix	arma::mat	int,int=1,int=1	Возвращае

			т матрицу Лапласа без учета весов
laplacianMatrixWeight	arma::mat	int,int=1,int=1	Возвращает матрицу Лапласа с учетом весов
simplexWeights	arma::mat	Int k	Возвращает матрицу весов для симплексов определённой размерности
laplacianSpectre	pair<arma::vec, arma::mat>	int,int=1,int=1, bool=false	Возвращает вектор собственных чисел и матрицу собственных векторов матрицы Лапласа с весом/без веса
bettiNumbers	arma::vec	void	Возвращает вектор с числами Бетти
reduce	arma::mat	arma::mat, int=0	Рекурсивная функция для приведения матрицы к нормальной форме Смита

Таблица 3.9. Описание и функциональное назначение методов класса SimplicialComplex

Метод	Тип возвращаемого знач.	Аргументы	Назначение
SimplicialComplex	—	bool=true	Конструктор, который создает

			объект класса Если аргумента нет или передано значение true, то используется диаграмма Хассе, иначе — симплекс дерево
insert	void	vector<int>&, double = 1	Добавления симплекса
erase	void	vector<int>&	Удаление симплекса
changeWeight	void	vector<int>&, double = 1	Изменение веса симплекса
simplexCount	int	—	Подсчёт числа симплексов
fVector	void	—	Вывод значений f-вектора
vertexDegreePQ	void	int, int	Вывод степеней вершин в (p, q)-графе
allSimplices	void	—	Вывод всех симплексов
closeness	void	int, int	Вывод closeness центральности вершин в (p, q)-графе
betweenness	void	int, int	Вывод betweenness центральности вершин в (p, q)-графе
distancePQ	double	vector<int>&, vector<int>&, int,	Подсчёт расстояния между

		int	двумя симплексами
clusterCoeff	double	int, int	Подсчёт коэфф. кластеризации (p, q)-графа
eulerNumber	int	—	Подсчёта Эйлеровой хар-ки
boundaryMatrix	arma::mat	int=1, int=1	Нахождение граничной матрицы
laplacianMatrix	arma::mat	int, int=1, int=1	Нахождение матрицы Лапласа без учёта весов
laplacianMatrixWeight	arma::mat	int, int=1, int=1	Нахождение матрицы Лапласа с учётом весов
laplacianSpectre	pair<arma::vec, arma::mat>	int, int=1, int=1, bool=false	Нахождение спектра матрицы Лапласа (с весами или без)
openStar	void	vector<int>&	Вывод симплексов из открытой звезды
closeStar	void	vector<int>&	Вывод симплексов из закрытой звезды
link	void	vector<int>&	Вывод симплексов линка
bettiNumbers	arma::vec	—	Нахождение чисел Бетти

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц в докум.)	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
Изм.	Измененных	Замененных	Новых	Аннулированных					