МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

 **«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Кафедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ДРЕВОВИДНОГО ПОИСКА РЕШЕНИЯ ЗАДАЧИ О Р-МЕДИАНЕ ГРАФА**

Работу выполнила \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.В.Степанова

 (подпись, дата) (инициалы, фамилия)

Факультет компьютерных технологий и прикладной математики 3 курс

Направление 02.03.02 – «Фундаментальная информатика и информационные технологии»

Научный руководитель, доц.

канд. физ-мат. наук \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Е.В.Кособуцкая

 (подпись, дата) (инициалы, фамилия)

Нормоконтролер, доц.

канд. физ-мат. наук \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Е.В.Кособуцкая

 (подпись, дата) (инициалы, фамилия)

Краснодар 2018

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc513978682)

[1 Задача о p-медиане 4](#_Toc513978683)

[2 Алгоритмы решения задачи о p-медиане 6](#_Toc513978684)

[2.1 Полный перебор 6](#_Toc513978685)

[2.2 Решение методом линейного программирования 6](#_Toc513978686)

[2.3 Алгоритм направленного древовидного поиска 7](#_Toc513978687)

[2.4 Приближенный алгоритм 9](#_Toc513978688)

[3 Выбор технологии параллельного программирования 10](#_Toc513978689)

[4 Реализация алгоритма направленного древовидного поиска 11](#_Toc513978690)

[4.1 Отсечение неподходящих ветвей 11](#_Toc513978691)

[4.2 Оценка возможности распараллеливания 12](#_Toc513978692)

[4.3 Детали реализации 14](#_Toc513978693)

[4.3.1 Генерация графа 14](#_Toc513978694)

[4.3.2 Распределение вычислений между потоками 15](#_Toc513978695)

[4.4 Анализ результатов 19](#_Toc513978696)

[ЗАКЛЮЧЕНИЕ 23](#_Toc513978697)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 24](#_Toc513978698)

[ПРИЛОЖЕНИЕ А Последовательная реализация алгоритма древовидного поиска решения задачи о р-медиане на языке С++ 25](#_Toc513978699)

[ПРИЛОЖЕНИЕ Б Текст программы, осуществляющий генерацию графов для исследований 28](#_Toc513978700)

[ПРИЛОЖЕНИЕ В Текст процедуры, выполняющей заполнение буфера 29](#_Toc513978701)

[ПРИЛОЖЕНИЕ Г Фрагмент программы, осуществляющий распределение вычислений между потоками 32](#_Toc513978702)

[ПРИЛОЖЕНИЕ Д Таблица с результатами исследований 33](#_Toc513978703)

# ВВЕДЕНИЕ

Методы теории графов применимы во многих областях науки. Графы используются для моделирования реальных систем, например, для моделирования сети дорог.

Одной из известных NP-трудных задач является задача о p-медиане. Она возникает во многих приложениях. Например, при размещении предприятий бытового обслуживания, складов, пунктов автосервиса на дорогах, коммутаторов в телефонной сети для минимизации расхода проводов. При этом не учитываются стоимости на открытие предприятий и есть ограничение на число предприятий, которые могут быть открыты. [1]. Требуется, чтобы сумма минимальных расстояний от каждой вершины до ближайшего к ней пункта обслуживания была наименьшей. Оптимальное в этом смысле расположение p пунктов обслуживания и будет являться p-медианой графа. [3]

На первый взгляд кажется, что нахождение р-медианы графа – довольно простая задача, которую можно решить перебором всех подмножеств множества вершин графа V мощностью p и запоминанием подмножества с минимальной суммой расстояний от каждой вершины до ближайшей к ней медианной. Это справедливо для небольших графов. Но с увеличением числа вершин этот метод становится громоздким с вычислительной точки зрения.

Одним из алгоритмов решения задачи о p-медиане является метод направленного древовидного поиска, он лучше всего соответствует структуре рассматриваемой задачи. [3]

Целью курсовой работы является реализация алгоритма направленного древовидного поиска, параллельная реализация этого алгоритма и оценка эффективности распараллеливания.

# 1 Задача о p-медиане

Рассмотрим понятие медианы графа. Пусть есть неориентированный граф G со множеством вершин V и матрицей кратчайших расстояний d. Тогда медианой графа будет такая вершина i, для которой величина

, (1)

то есть сумма кратчайших расстояний от всех вершин до заданной, была бы наименьшей среди всех вершин графа. Эта величина называется передаточным числом вершины [3]

p-медианой графа называется такое множество вершин Vp, что |Vp|=p, и величина

, (2)

то есть сумма кратчайших расстояний от всех вершин до ближайшей вершины из множества Vp, была бы минимальной для всех подмножеств мощности р множества вершин V. Эта величина называется передаточным числом для множества Vp.

Пусть j – некоторая произвольная вершина графа, v – некоторая вершина из множества Vp и для вершины j достигается минимум

 (3)

при i=v, то будем говорить, что вершина j прикреплена к вершине v. [3].

Решение задачи о p-медиане состоит в нахождении множества Vp, для которого передаточное число было бы наименьшим среди всех подмножеств вершин графа мощностью р. Также в некоторых случаях требуется нахождение передаточного числа множества медианных вершин.

# 2 Алгоритмы решения задачи о p-медиане

## 2.1 Полный перебор

 Алгоритм заключается в полном переборе всех подмножеств множества вершин графа, вычислении для каждого из них передаточного числа и запоминании подмножества с наименьшим передаточным числом. Достоинствами этого алгоритма являются простота в понимании и реализации, недостатком – высокая вычислительная сложность для графов большой размерности.

## 2.2 Решение методом линейного программирования

 Линейное программирование — математическая дисциплина, посвящённая теории и методам решения экстремальных задач на множествах n-мерного векторного пространства, задаваемых системами линейных уравнений и неравенств. [2].

 Задачу о р-медиане можно сформулировать в терминах линейного программирования следующим образом.

Пусть [dij] – матрица кратчайших расстояний, [eij] – матрица распределения, где



Примем eii=1, если вершина i является медианной и 0 в противном случае. Тогда задача о р-медиане может быть сформулирована как минимизировать функцию

(4)

при ограничениях

 (5)

 (6)

 (7)

(8)

 Полученную задачу линейного программирования нетрудно решить, однако решение не обязательно будет целочисленным. [3]

## 2.3 Алгоритм направленного древовидного поиска

 В этом алгоритме используется дерево поиска. Подзадача, возникающая при ветвлении на каком-либо узле дерева, заключается в том, что просматриваемая вершина j прикрепляется к какой-нибудь вершине i, причём вершина i является медианной.

 Вначале строится матрица M=[mkj], в которой j-й столбец содержит все вершины графа в порядке неубывания расстояния до вершины j. Очевидно, что m1j равно j, поскольку расстояние от вершины до самой себя равно нулю, и таких вершин в графе больше нет. [3]

 Ниже приведена реализация алгоритма на псевдокоде. Параметрами являются текущая просматриваемая вершина current\_vertex, median\_set - множество вершин, входящих в предполагаемую p-медиану, not\_median\_set – множество вершин, не входящих в p-медиану. Индексации всех элементов начинаются с нуля.

 ПРОЦЕДУРА tree\_search\_algorithm (current\_vertex, median\_set, not\_median\_set)

1. ЕСЛИ размерность median\_set равна p, то

2. Для всех вершин v ≥ current\_vertex ВЫПОЛНЯТЬ

 3. Найти минимальное i, что M[i,v] входит в median\_set

 4. прикрепить вершину v к вершине M[i,v]

 5. посчитать сумму расстояний от всех вершин к вершинам, к которым они прикреплены

 6. ЕСЛИ сумма расстояний оказалась меньше ранее найденной или это первая найденная предполагаемая медиана

 7. ЗАПОМНИТЬ текущее множество set\_median и сумму в глобальную переменную

 8. Возврат вверх по рекурсии для проверки других веток

9. ЕСЛИ current\_vertex равно общему числу вершин (это означает, что все вершины просмотрены)

 10. ВОЗВРАТ

11. Положить ind=0

12. ПОКА ind меньше числа вершин И M[ind, current\_vertex] НЕ лежит в set\_median ВЫПОЛНЯТЬ

 13. ЕСЛИ M[ind, current\_vertex] НЕ лежит в not\_set\_compsub ТО

 14. Положить M[ind, current\_vertex] в median\_set

 15. Прикрепить current\_vertex к M[ind, current\_vertex]

 16. Рекурсивный вызов tree\_search\_algorithm (current\_vertex +1, median\_set, not\_median\_set)

 17. Извлечь M[ind, current\_vertex] в median\_set и положить в not\_median\_set

 18. Увеличить ind на 1

19. ЕСЛИ M[ind, current\_vertex]лежит в set\_median

 20. Прикрепить current\_vertex к M[ind, current\_vertex]

21. Рекурсивный вызов tree\_search\_algorithm (current\_vertex +1, median\_set, not\_median\_set)

22. ВОЗВРАТ

## 2.4 Приближенный алгоритм

Алгоритм заключается в следующем. Случайным образом выбирается подмножество вершин S мощностью p, аппроксимирующая p-медианное множество. Затем проверяется, может ли некоторая вершина графа j, не входящая в S, заменить какую-нибудь вершину i из S так, чтобы передаточное число этого множества стало меньше. Если такая вершина есть, строится новое множество S’, в котором вершина i заменена на вершину j, и затем аналогично исследуется множество S’. Когда на некотором шаге получится такое множество, что никакую его вершину нельзя заменить другой, не входящей в это множество, то такое множество считается искомым приближением p-медианы. [3].

 Достоинством алгоритма является его относительно невысокая вычислительная сложность, а недостатком является то, что алгоритм не является точным.

# 3 Выбор технологии параллельного программирования

 Для организации параллельных вычислений была выбрана технология OpenMP. Она встроена в Visual C и имеет много преимуществ.

 Во-первых, она позволяет реализовывать возможности многопроцессорных систем с общей памятью, без трудоёмких межпроцессорных передач сообщений.

 Во-вторых, распараллеливание относительно несложных программ не требует больших усилий со стороны разработчика – иногда достаточно включить лишь несколько директив OpenMP в последовательную программу.

 В-третьих, технология обеспечивает возможность поэтапной разработки – директивы могут добавляться в программу постепенно, что позволяет на ранних этапах разработки получить параллельную программу.

 В-четвёртых, программный код последовательного и параллельного варианта программы является единым, что упрощает проблему совершенствования программ. [4]

# 4 Реализация алгоритма направленного древовидного поиска

## 4.1 Отсечение неподходящих ветвей

 Можно попытаться несколько сократить перебор вариантов, если сразу отсекать некоторые ветви, которые заведомо не приведут к искомому результату. Например:

1) при прикреплении некоторой вершины к какой-нибудь медианной вершине можно добавлять расстояние от прикреплённой до медианной вершины в локальную сумму. Так, если просматривается k-я вершина, то локальная сумма будет содержать сумму расстояний от первых k вершин до ближайших медианных (считать, что нумерация вершин начинается с нуля). Если в какой-то момент времени локальная сумма окажется больше ранее найденного передаточного числа для некоторой предполагаемой медианы, то дальнейшее исследование этой ветки не приведёт к нахожденияю множества с меньшим передаточным числом, и эту ветку можно отбросить;

2) если при просмотре некоторой ветки множество not\_set\_median будет обладать мощностью m>n-p, где n – число вершин графа, то мощность set\_median не достигнет значения p при дальнейшем просмотре ветки, и такую ветку также можно отбросить;

3) пусть исследуемый граф связный, то есть существует путь (а, значит, и кратчайший путь) между всеми парами вершин. Тогда в матрице кратчайших путей в каждом столбце будет ровно n ячеек, где n – число вершин графа. Очевидно, что если медиана содержит p вершин, то в ячейках с индексами от нуля до по n-p, то есть всех, кроме последних p-1 ячейки столбца, встретится хотя бы одна медианная вершина, то есть последние p-1 ячейку каждого столбца можно не рассматривать. [3]

 Программная реализация алгоритма с учётом отсечений веток, заведомо не приводящих к решению, представлена в приложении А.

## 4.2 Оценка возможности распараллеливания

 Рассмотрим информационные зависимости переменных и структур процедуры в пределах одного цикла её работы. Рассматривать будем только случай, когда искомое множество вершин ещё не найдено и текущая ветка не отсекается как заведомо неудачная.

 Процедура содержит следующие параметры:

 1) distance\_matrix – матрица, [i,j] элемент которого содержит пару: номер j-й ближайшей к i-й вершине и расстояние между этими вершинами.

 2) current\_vertex – номер текущей прикрепляемой вершины, равен ярусу просматриваемого узла дерева поиска.

 3) median\_set – множество вершин, предположительно входящих в медиану.

 4) not\_median\_set – множество вершин, не входящих в искомую медиану.

 5) p – размерность медианы.

 6) current\_sum – текущая сумма расстояний от уже просмотренных вершин до тех вершин, к которым они были прикреплены.

К локальным переменным относятся:

1) vertex – номер прикрепляемой вершины, используется, если найдено множество из p вершин и к ним надо прикрепить ещё не рассмотренные вершины.

2) num\_cell – номер просматриваемой ячейки матрицы, второй индекс просматриваемой ячейки distance\_matrix (первый – номер текущей вершины current\_vertex).

 Размерность медианы p и матрица кратчайших расстояний distance\_matrix не меняются во время выполнения алгоритма, поэтому они не порождают информационных зависимостей между узлами дерева поиска. Локальная переменная vertex используется только если найдено множество из p вершин, рассматривать этот случай не будем.

 На рисунке 1 представлены информационные зависимости между переменными в пределах одной итерации цикла while.



Рисунок 1 – Информационные зависимости между ветвями дерева поиска

Пояснение к дуге (1). Здесь из медианного множества извлекается только что помещённая в него вершина, и поэтому множество остаётся таким же, каким было в начале выполнения итерации. Выходные переменные этой итерации являются входными для следующей итерации цикла.

Видно, что текущая вершина, текущая сумма расстояний и множество медианных вершин не меняется от итерации к итерации цикла, номер просматриваемой ячейки столбца увеличивается на 1 с каждой итерацией, а множество not\_median\_set на пополняется очередной вершиной, которая уже была рассмотрена как медианная и исключается из дальнейшего осмотрения.

Также на рисунке 1 видно, что рекурсивный вызов не порождает никаких информационных зависимостей, то есть информация передаётся только вниз по рекурсии.

Дерево рекурсивных вызовов процедуры изображено на рисунке 2.

 

Рисунок 2 – Дерево рекурсивных вызовов процедуры

На рисунке 2 узлы дерева представляют собой рекурсивные вызовы процедуры, а стрелки означают, из какого вызова процедуры был произведён очередной рекурсивный вызов. Корень дерева означает вызов процедуры из основной программы.

Очевидно, что ветки, не имеющие общих узлов, не имеют между собой никаких информационных зависимостей и могут быть выполнены параллельно. Например, две выделенные ветки могут быть выполнены параллельно.

## 4.3 Детали реализации

### 4.3.1 Генерация графа

 Алгоритм реализован только для связных графов, поэтому необходимо реализовать процедуру, которая генерирует связный граф. Процедура получает на вход требуемое число вершин Ver и максимальный вес ребра max\_weight и генерирует граф с

 (9)

рёбрами. Это минимальное количество рёбер, необходимое, чтобы граф наверняка был связным.

 Граф представляется в виде списков смежности, то есть для каждой вершины создаётся список, куда будут вноситься смежные с ней вершины при генерации графа. Затем функция генерирует три псевдослучайных числа a, b и weight, являющиеся соответственно предполагаемыми концами нового ребра и его весом. Вес ребра является целым числом от одного до max\_weight.

Затем проверяется, не является ли это ребро петлёй (то есть должно выполняться условие неравенства a и b) и не содержится ли оно уже в графе (для этого достаточно просмотреть, не содержится ли вершина b в списке смежности вершины a). Если оба условия выполнены, то в список смежности вершины a добавляется вершина b и в список смежности вершины b добавляется вершина a, а ребро считается сгенерированным. Генерация пар чисел будет происходить до тех пор, пока не будет сгенерировано нужное число рёбер. Сгенерированные рёбра заносятся в файл в виде троек a, b, weight через пробел. Каждое ребро записывается в файл с новой строки.

 Полученный файл затем считывается основной программой.

 Текст программы, осуществляющий генерацию графов, представлен в приложении Б.

### 4.3.2 Распределение вычислений между потоками

 Если посмотреть на рисунок 2, видно, что ветви, начинающиеся с различных узлов одного яруса, независимы между собой и могут быть параллельно выполнены. Распараллеливание выполняется следующим образом.

 Пусть распараллеливание начинается с k-го яруса. Сначала один поток просматривает узлы дерева вплоть до k-го яруса аналогично тому, как они бы просматривались при последовательном алгоритме. Иначе говоря, выполняются рекурсивные вызовы до глубины k. Считается, что вызов из основной программы имеет глубину 0.

Как только достигается узел k-го яруса (то есть выполнен рекурсивный вызов глубины k), информация об этом узле, а именно множества median\_set, not\_median\_set и текущая сумма расстояний current\_sum, запоминается в буфер-очередь. При этом если во время заполнения буфера найдётся какое-нибудь медианное множество (а оно найдётся, если p≤k), оно обработается так же, как и в последовательном алгоритме. Множество и его передаточной число запишутся в ячейки нулевого потока. Текст процедуры, выполняющей заполнение буфера, приведён в приложении В.

 Таким образом при заданном ярусе k в буфер запишется информация обо всех узлах k-го уровня дерева поиска. Отметим, что для каждого такого узла выполнено прикрепление первых k вершин к некоторым вершинам предполагаемой медианы. Дальнейший просмотр каждого узла необходимо начинать с k-й вершины (вершины нумеруются с нуля и последняя просмотренная вершина – это вершина с индексом k-1). Визуально заполнение буфера представлено на рисунке 3.



Рисунок 3 – заполнение буфера

 Заполнение буфера осуществляется одним потоком, который просматривает только верхнюю часть дерева.

 После заполнения буфера производится распределение вычислений между потоками следующим образом. Поток извлекает некоторый элемент буфера – информацию о каком-то узле дерева поиска. Отметим, что буфер относится к критическим данным (это такие данные, при доступе к которым одновременно нескольких потоков, результат выполнения программы будет непредсказуем), и при доступе к буферу организована критическая секция. При извлечении информации об узле эта информация из буфера удаляется, чтобы одна и та же ветвь не была просмотрена несколько раз. Критическая секция завершается после извлечения информации из буфера, и другой поток может извлечь информацию о другом узле.

 Когда поток извлёк информацию об узле, он просматривает все узлы дерева, являющиеся потомками данного узла, то есть для этой ветви вызывается обычный последовательный алгоритм с той разницей, что если найдена какая-то предполагаемая медиана и передаточное число для неё, оно сравнивается с ранее найденным только этим потоком передаточным числом. После просмотра ветви поток извлекает из буфера информацию об очередном узле, если буфер не пуст. Фрагмент программы, осуществляющий распределение вычислений, представлен в приложении Г.

 Алгоритм завершает работу, когда буфер пуст и все потоки просмотрели свои ветви. Визуально распределение вычислений между потоками представлено на рисунке 4.



Рисунок 4 – Распределение вычислений между потоками

 Выделенные ветви могут быть выполнены параллельно. Поскольку они независимы, неважно, в каком порядке они будут выполняться и какой поток выполнит каждую ветвь.

 Для каждого потока хранится множество с наименьшим передаточным числом и само это число. После окончания просмотра дерева один из потоков просматривает найденные множества и выбирает то, у которого наименьшее передаточное число. Если таких множеств несколько, выбирается то, которое было найдено потоком с наименьшим номером. Отметим, что таких множеств может быть несколько и при запуске программы может быть выбрано любое из них, поэтому множество, найденное при помощи последовательного и параллельного алгоритмов и даже множества, найденные параллельным алгоритмом при выборе разного количества ярусов, могут отличаться, однако передаточное число этих множеств будет одним и тем же.

## 4.4 Анализ результатов

 С одной стороны, увеличение номера яруса увеличивает время выполнения первого этапа – заполнение буфера, на котором распараллеливание не производится. С другой стороны, увеличение яруса позволит разбить дерево поиска на более мелкие ветви, что позволит распределить вычисления более равномерно между потоками.

 Полная таблица с результатами приведена в приложении Д. Было проведено 184 опыта, на графах с разным числом вершин и для различной размерности искомой p-медианы. Ярус, информация о котором заносилась в таблицу результата, определялся следующим образом:

1) выполнить запуск последовательного алгоритма, засечь время его выполнения;

2) выбрать номер яруса=1;

3) выполнить запуск для заданного яруса, засечь время;

4) если засечённое время больше ранее засеченного времени – завершить выполнение, вывести номер яруса на 1 меньше текущего, иначе перейти к шагу 5;

5) запомнить засечённое время;

6) увеличить номер яруса на 1, перейти к шагу 3.

Таким образом, номер яруса увеличивается, пока наблюдается уменьшение времени выполнения алгоритма. Наилучшее время заносится в таблицу как время выполнения параллельного алгоритма.

На рисунке 5 представлено соотношение числа экспериментов, когда ускорения достигнуто не было, когда было достигнуто ожидаемое ускорение и когда было достигнуто сверхускорение.



Рисунок 5 – Соотношение числа экспериментов, когда было достигнуто различное ускорение

Здесь сверхускорением считается ускорение более чем в четыре раза, поскольку вычисления выполнялись на четырёхъядерной машине. Большое число экспериментов, на которых было достигнуто сверхускорение можно объяснить тем, что при последовательном выполнении ядро, выполняющее вычисления, также использовалось и другими программами в то время как остальные три не использовались ничем. При параллельном выполнении алгоритма каждое из этих трёх ядер давало больший вклад, чем то, которое использовалось и другими программами.

Ускорения не было только при размерности медианы, равной 1, при больших размерностях ускорение было получено во всех экспериментах. Ниже представлен график зависимости ускорения от размерности медианы для графов с разным числом вершин.



Рисунок 6 – Зависимость ускорения от размерности медианы

Видно, что с увеличением размерности медианы растёт ускорение, а начиная от размерности 6 и выше начинает проявляться сверхускорение.

На рисунке 7 представлена зависимость ускорения от числа вершин в графе при неизменной размерности искомой р-медианы.



Рисунок 7 – График зависимости ускорения от размерности графа

Здесь зависимость неочевидна, можно проследить небольшое уменьшение ускорения, однако однозначных выводов сделать нельзя.

 Также интересно проследить зависимость наиболее эффективного для распараллеливания яруса от размерности графа и медианы. Эти зависимости представлены на рисунках 8 и 9.



Рисунок 8 – График зависимости оптимального яруса от размерности графа



Рисунок 9 – График зависимости оптимального яруса от размерности искомой медианы

Ни в том, ни в другом случае очевидная зависимость не прослеживается.

# ЗАКЛЮЧЕНИЕ

В процессе работы были изучены различные алгоритмы нахождения р-медианы графа, возможности использования технологии OpenMP для практической реализации параллельного алгоритма на языке С++ в среде Visual Studio 2015.

Было рассмотрено распараллеливание с использованием буфера, хранящего информацию об узлах одного яруса дерева поиска. Однозначного вывода о том, какой ярус брать для наиболее эффективного распараллеливания в зависимости от размерности графа и количества вершин в искомой р-медиане, сделать нельзя.

Однако можно сказать, что в подавляющем большинстве экспериментов – около 95%, было достигнуто ускорение, причём в 44% экспериментов случаев наблюдалось сверхускорение. Поэтому можно сделать вывод, что распараллеливание было проведено эффективно.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Задача о p-медиане [Электронный ресурс]. URL: http://www.math.nsc.ru/AP/benchmarks/P-median/p-med.html [дата обращения 5 мая 2018].
2. Линейное программирование [Электронный ресурс]. URL: [https://ru.wikipedia.org/wiki/Линейное\_программирование#Транспортная\_задача](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5#Транспортная_задача) [дата обращения 5 мая 2018].
3. Н. Кристофидес Теория графов. Алгоритмический подход. ; изд-во «Мир», 1978. – 432 c.
4. Гергель В.П. Высокопроизводительные вычисления для многоядерных многопроцессорных систем. – Учебное пособие – Нижний Новгород; Изд-во ННГУ им. Н.И. Лобачевского, 2010 [Электронный ресурс]. URL: http://www.unn.ru/pages/e-library/methodmaterial/2010/7.pdf [Дата обращения [12 мая 2018].

ПРИЛОЖЕНИЕ А

Последовательная реализация алгоритма древовидного поиска решения задачи о р-медиане на языке С++

void improved\_serial\_tree\_search\_algorithm(

 const vector<vector<pair<int, int>>> & distance\_matrix, //таблица

 int current\_vertex,//текущая вершина

 set<int> median\_set,//множество вершин p-медианы

 set<int> not\_median\_set,//множество вершин, не входящих в медиану

 int p,//размерность медианы

 long current\_sum//текущая сумма расстояний

)

{

 if (median\_set.size() == p)//если уже найдено множество из p вершин

 {

 //надо прикрепить остальные вершины

 for (int vertex = current\_vertex; vertex < distance\_matrix.size(); vertex++)

 {

 int num\_cell = 0;

 while (num\_cell < distance\_matrix[vertex].size() && !median\_set.count(distance\_matrix[vertex][num\_cell].second))

 num\_cell++;

 /\*if (num\_cell == distance\_matrix[vertex].size()) //значит не нашли вершину, к которой можно прикрепить данную, такое возможно только в несвязных графах

 return;

 else\*/

 current\_sum += distance\_matrix[vertex][num\_cell].first;//добавили расстояние

 }

 if (current\_sum < sum\_distance)

 {

 sum\_distance = current\_sum;

 current\_median = median\_set;

 }

 return;//выходим из функции, дальше искать нельзя

 }

 if (current\_vertex == distance\_matrix.size()) //если уже исследовали все вершины, то выход

 return;

 //первое условие

 if (current\_sum >= sum\_distance)

 return;

 //второе условие

 if (not\_median\_set.size() > distance\_matrix.size() - p)

 return;

 int num\_cell = 0;

 //третье условие

 while (num\_cell <= distance\_matrix[current\_vertex].size() - p && !median\_set.count(distance\_matrix[current\_vertex][num\_cell].second))

 {

 if (!not\_median\_set.count(distance\_matrix[current\_vertex][num\_cell].second))

 {

 median\_set.insert(distance\_matrix[current\_vertex][num\_cell].second);

 serial\_tree\_search\_algorithm(distance\_matrix, current\_vertex + 1, median\_set, not\_median\_set, p, current\_sum + distance\_matrix[current\_vertex][num\_cell].first);

 median\_set.erase(distance\_matrix[current\_vertex][num\_cell].second);

 not\_median\_set.insert(distance\_matrix[current\_vertex][num\_cell].second);

 }

 ++num\_cell;

 }

 if (num\_cell < distance\_matrix[current\_vertex].size() - p + 1)//если наткнулись на вершину, которая уже помещена в медиану на предыдущем шаге

 serial\_tree\_search\_algorithm(distance\_matrix, current\_vertex + 1, median\_set, not\_median\_set, p, current\_sum + distance\_matrix[current\_vertex][num\_cell].first);

}

# ПРИЛОЖЕНИЕ Б

Текст программы, осуществляющий генерацию графов для исследований

void Generate(int Ver, int Ed, int max\_weight)//число вершин и рёбер

{

 ofstream wr("graph" + to\_string(Ver) + ".txt");

 int sum = 0; vector<list<int>> vec (Ver);

 uniform\_int\_distribution<int> uid (0, Ver-1);

 uniform\_int\_distribution<int> weight\_dist(1, max\_weight);

 random\_device rnd;

 while (sum < Ed)

 {

 int a = uid(rnd); int b = uid(rnd); int w = weight\_dist(rnd);

 if (find(vec[a].begin(), vec[a].end(), b) == vec[a].end() && a != b)

 {

 vec[a].push\_back(b);

 vec[b].push\_back(a);

 ++sum;

 wr << a+1 << " " << b+1 <<" "<< w<<endl;

 }

 }

 wr.close();

}

int main()

{

 for (int i = 10; i <= 50; i+=1)

 {

 Generate(i, (i-1)\*(i-2)/2+1, 10);

 }

}

# ПРИЛОЖЕНИЕ В

Текст процедуры, выполняющей заполнение буфера

void fill\_buffer(

 const vector<vector<pair<int, int>>> & distance\_matrix, //таблица

 int current\_vertex,//текущая вершина

 set<int> median\_set,//множество вершин p-медианы

 set<int> not\_median\_set,//множество вершин, не входящих в медиану

 int p,//размерность медианы

 long current\_sum, //текущая сумма расстояний

 int requed\_vertex //требуемая глубина

)

{

 if (requed\_vertex == current\_vertex)

 {

 params new\_params;

 new\_params.current\_sum = current\_sum;

 new\_params.median\_set = median\_set;

 new\_params.not\_median\_set = not\_median\_set;

 buffer.push(new\_params);

 return;

 }

 if (median\_set.size() == p)//если уже найдено множество из p вершин

 {

 //надо прикрепить остальные вершины

 for (int vertex = current\_vertex; vertex < distance\_matrix.size(); vertex++)

 {

 int num\_cell = 0;

 while (num\_cell < distance\_matrix[vertex].size() && !median\_set.count(distance\_matrix[vertex][num\_cell].second))

 num\_cell++;

 current\_sum+= distance\_matrix[vertex][num\_cell].first;//добавили расстояние

 }

 if (current\_sum < current\_sums[0])

 {

 current\_sums[0] = current\_sum;

 current\_medians[0] = median\_set;

 }

 return;//выходим из функции, дальше искать нельзя

 }

 if (current\_vertex == distance\_matrix.size()) //если уже исследовали все вершины, то выход

 return;

 //первое условие

 if (current\_sum >= current\_sums[0])

 return;

 //второе условие

 if (not\_median\_set.size() > distance\_matrix.size() - p)

 return;

 int num\_cell = 0;

 //третье условие

 while (num\_cell <= distance\_matrix[current\_vertex].size() - p && !median\_set.count(distance\_matrix[current\_vertex][num\_cell].second))

 {

 if (!not\_median\_set.count(distance\_matrix[current\_vertex][num\_cell].second))

 {

 median\_set.insert(distance\_matrix[current\_vertex][num\_cell].second);

 fill\_buffer(distance\_matrix, current\_vertex + 1, median\_set, not\_median\_set, p, current\_sum + distance\_matrix[current\_vertex][num\_cell].first, requed\_vertex);

 median\_set.erase(distance\_matrix[current\_vertex][num\_cell].second);

 not\_median\_set.insert(distance\_matrix[current\_vertex][num\_cell].second);

 }

 ++num\_cell;

 }

 if (num\_cell < distance\_matrix[current\_vertex].size() - p + 1)//если наткнулись на вершину, которая уже помещена в медиану на предыдущем шаге

 fill\_buffer(distance\_matrix, current\_vertex + 1, median\_set, not\_median\_set, p, current\_sum + distance\_matrix[current\_vertex][num\_cell].first, requed\_vertex);

}

# ПРИЛОЖЕНИЕ Г

Фрагмент программы, осуществляющий распределение вычислений между потоками

 params thread\_params;

 bool flag;

#pragma omp parallel shared(res\_matrix) private(thread\_params, flag)

 {

 int num\_thread = omp\_get\_thread\_num();

 flag = false;

 while (!buffer.empty())

 {

#pragma omp critical

 {

 if (!buffer.empty())

 {

 thread\_params = buffer.front();

 buffer.pop();

 flag = true;

 }

 }

 //конец критической секции

 if (flag)

 {

 parallel\_tree\_search\_algorithm(res\_matrix, deep, thread\_params.median\_set, thread\_params.not\_median\_set, p, thread\_params.current\_sum, num\_thread);

 flag = false;

 }

 }

 }

# ПРИЛОЖЕНИЕ Д

Таблица с результатами исследований

| Число вершин графа | Размерность медианы | Время выполнения последовательного алгоритма, с | Оптимальный номер яруса | Время выполнения параллельного алгоритма, с | Ускорение, раз |
| --- | --- | --- | --- | --- | --- |
| 10 | 1 | 0,0074 | 0 | 0,0074 | 1,00 |
| 10 | 2 | 0,0350 | 1 | 0,0265 | 1,32 |
| 10 | 3 | 0,0987 | 1 | 0,0319 | 3,09 |
| 10 | 4 | 0,1871 | 2 | 0,0453 | 4,13 |
| 10 | 5 | 0,2714 | 2 | 0,0452 | 6,00 |
| 10 | 6 | 0,3302 | 2 | 0,0438 | 7,54 |
| 10 | 7 | 0,3481 | 1 | 0,0258 | 13,50 |
| 10 | 8 | 0,3059 | 1 | 0,0131 | 23,34 |
| 10 | 9 | 0,2509 | 1 | 0,0068 | 37,04 |
| 10 | 10 | 0,1893 | 1 | 0,0052 | 36,37 |
| 11 | 1 | 0,0111 | 0 | 0,0111 | 1,00 |
| 11 | 2 | 0,0674 | 1 | 0,0282 | 2,39 |
| 11 | 3 | 0,1646 | 1 | 0,0522 | 3,15 |
| 11 | 4 | 0,2949 | 2 | 0,0829 | 3,56 |
| 11 | 5 | 0,4506 | 2 | 0,0871 | 5,18 |
| 11 | 6 | 0,5784 | 3 | 0,0771 | 7,50 |
| 11 | 7 | 0,6445 | 3 | 0,0565 | 11,41 |
| 11 | 8 | 0,6545 | 1 | 0,0268 | 24,38 |
| 11 | 9 | 0,5625 | 1 | 0,0130 | 43,30 |
| 11 | 10 | 0,4913 | 1 | 0,0075 | 65,44 |
| 12 | 1 | 0,0172 | 1 | 0,0153 | 1,13 |
| 12 | 2 | 0,0850 | 1 | 0,0341 | 2,50 |
| 12 | 3 | 0,1936 | 1 | 0,0658 | 2,94 |
| 12 | 4 | 0,3946 | 2 | 0,0982 | 4,02 |
| 12 | 5 | 0,6604 | 2 | 0,0823 | 8,02 |
| 12 | 6 | 0,9275 | 2 | 0,0747 | 12,42 |
| 12 | 7 | 1,1680 | 3 | 0,0640 | 18,25 |
| 12 | 8 | 1,2624 | 3 | 0,0507 | 24,92 |
| 12 | 9 | 1,3279 | 3 | 0,0354 | 37,51 |
| 12 | 10 | 1,3279 | 1 | 0,0228 | 58,16 |
| 13 | 1 | 0,0216 | 1 | 0,0188 | 1,15 |
| 13 | 2 | 0,1155 | 1 | 0,0472 | 2,45 |
| 13 | 3 | 0,2896 | 1 | 0,1002 | 2,89 |
| 13 | 4 | 0,6042 | 1 | 0,1836 | 3,29 |
| 13 | 5 | 1,1647 | 2 | 0,2535 | 4,60 |
| 13 | 6 | 1,7730 | 1 | 0,3631 | 4,88 |
| 13 | 7 | 2,4850 | 4 | 0,2738 | 9,07 |
| 13 | 8 | 2,6324 | 4 | 0,2092 | 12,58 |
| 13 | 9 | 2,8442 | 4 | 0,1210 | 23,50 |
| 13 | 10 | 2,7237 | 3 | 0,0760 | 35,83 |
| 14 | 1 | 0,0220 | 0 | 0,0220 | 1,00 |
| 14 | 2 | 0,1339 | 1 | 0,0593 | 2,26 |
| 14 | 3 | 0,3991 | 1 | 0,1568 | 2,54 |
| 14 | 4 | 0,9754 | 1 | 0,3202 | 3,05 |
| 14 | 5 | 1,9674 | 2 | 0,4236 | 4,64 |
| 14 | 6 | 3,3173 | 3 | 0,4511 | 7,35 |
| 14 | 7 | 5,1826 | 3 | 0,4170 | 12,43 |
| 14 | 8 | 6,1449 | 2 | 0,2483 | 24,75 |
| 14 | 9 | 6,5506 | 2 | 0,1935 | 33,85 |
| 14 | 10 | 6,7094 | 2 | 0,0923 | 72,69 |
| 15 | 1 | 0,0266 | 2 | 0,0236 | 1,13 |
| 15 | 2 | 0,1638 | 1 | 0,0768 | 2,13 |
| 15 | 3 | 0,5452 | 1 | 0,2161 | 2,52 |
| 15 | 4 | 1,5576 | 2 | 0,4983 | 3,13 |
| 15 | 5 | 3,2630 | 2 | 0,7694 | 4,24 |
| 15 | 6 | 6,1577 | 2 | 0,9257 | 6,65 |
| 15 | 7 | 8,9574 | 2 | 1,0267 | 8,72 |
| 15 | 8 | 13,3041 | 3 | 0,9499 | 14,01 |
| 15 | 9 | 13,7360 | 3 | 0,5467 | 25,13 |
| 15 | 10 | 14,4054 | 3 | 0,3648 | 39,49 |
| 16 | 1 | 0,0335 | 1 | 0,0279 | 1,20 |
| 16 | 2 | 0,1808 | 1 | 0,0867 | 2,09 |
| 16 | 3 | 0,6241 | 1 | 0,2351 | 2,65 |
| 16 | 4 | 1,7517 | 2 | 0,4667 | 3,75 |
| 16 | 5 | 3,9807 | 2 | 0,7809 | 5,10 |
| 16 | 6 | 8,8738 | 2 | 1,1345 | 7,82 |
| 16 | 7 | 12,6396 | 2 | 1,0875 | 11,62 |
| 16 | 8 | 17,1592 | 5 | 1,0955 | 15,66 |
| 16 | 9 | 21,1599 | 4 | 0,7683 | 27,54 |
| 16 | 10 | 23,2258 | 2 | 0,5852 | 39,69 |
| 17 | 1 | 0,0351 | 1 | 0,0279 | 1,26 |
| 17 | 2 | 0,2330 | 1 | 0,1239 | 1,88 |
| 17 | 3 | 0,9698 | 1 | 0,4258 | 2,28 |
| 17 | 4 | 3,4850 | 1 | 1,1110 | 3,14 |
| 17 | 5 | 8,4448 | 2 | 2,0254 | 4,17 |
| 17 | 6 | 17,4943 | 4 | 3,0212 | 5,79 |
| 17 | 7 | 28,0380 | 2 | 3,2796 | 8,55 |
| 17 | 8 | 40,7404 | 2 | 3,2752 | 12,44 |
| 17 | 9 | 51,4562 | 3 | 3,0159 | 17,06 |
| 17 | 10 | 59,6728 | 3 | 2,3398 | 25,50 |
| 18 | 1 | 0,0467 | 1 | 0,0310 | 1,50 |
| 18 | 2 | 0,2507 | 1 | 0,1419 | 1,77 |
| 18 | 3 | 1,0884 | 1 | 0,4994 | 2,18 |
| 18 | 4 | 3,7217 | 1 | 1,3026 | 2,86 |
| 18 | 5 | 10,4059 | 4 | 2,7038 | 3,85 |
| 18 | 6 | 20,8881 | 3 | 4,1472 | 5,04 |
| 18 | 7 | 38,7898 | 3 | 5,3843 | 7,20 |
| 18 | 8 | 63,1066 | 3 | 5,8374 | 10,81 |
| 18 | 9 | 81,2892 | 6 | 4,3786 | 18,57 |
| 18 | 10 | 97,7089 | 3 | 3,2509 | 30,06 |
| 19 | 1 | 0,0542 | 1 | 0,0390 | 1,39 |
| 19 | 2 | 0,3114 | 1 | 0,1845 | 1,69 |
| 19 | 3 | 1,4182 | 1 | 0,6947 | 2,04 |
| 19 | 4 | 5,1857 | 1 | 1,9900 | 2,61 |
| 19 | 5 | 15,0620 | 1 | 4,2898 | 3,51 |
| 19 | 6 | 35,2791 | 2 | 6,8575 | 5,14 |
| 19 | 7 | 69,7734 | 2 | 9,4098 | 7,41 |
| 19 | 8 | 118,0180 | 3 | 10,7626 | 10,97 |
| 19 | 9 | 171,9410 | 2 | 10,6746 | 16,11 |
| 19 | 10 | 228,1840 | 6 | 8,6070 | 26,51 |
| 20 | 1 | 0,0467 | 1 | 0,0423 | 1,10 |
| 20 | 2 | 0,3954 | 1 | 0,2286 | 1,73 |
| 20 | 3 | 1,8678 | 1 | 0,9402 | 1,99 |
| 20 | 4 | 7,0728 | 1 | 2,7444 | 2,58 |
| 20 | 5 | 21,0380 | 3 | 6,0986 | 3,45 |
| 20 | 6 | 47,3203 | 2 | 9,8939 | 4,78 |
| 20 | 7 | 97,1005 | 2 | 12,6857 | 7,65 |
| 20 | 8 | 168,1200 | 3 | 13,8307 | 12,16 |
| 20 | 9 | 265,2830 | 4 | 12,2790 | 21,60 |
| 20 | 10 | 363,7160 | 4 | 10,2265 | 35,57 |
| 21 | 1 | 0,0487 | 0 | 0,0487 | 1,00 |
| 21 | 2 | 0,4309 | 1 | 0,2706 | 1,59 |
| 21 | 3 | 2,2429 | 1 | 1,1691 | 1,92 |
| 21 | 4 | 8,9834 | 1 | 3,7385 | 2,40 |
| 21 | 5 | 28,7655 | 2 | 9,0585 | 3,18 |
| 21 | 6 | 75,2796 | 2 | 17,1077 | 4,40 |
| 21 | 7 | 172,5270 | 5 | 27,9071 | 6,18 |
| 21 | 8 | 325,2880 | 2 | 40,2459 | 8,08 |
| 21 | 9 | 517,8940 | 2 | 47,1047 | 10,99 |
| 21 | 10 | 723,1200 | 2 | 51,1854 | 14,13 |
| 22 | 1 | 0,0576 | 1 | 0,0553 | 1,04 |
| 22 | 2 | 0,4695 | 1 | 0,2942 | 1,60 |
| 22 | 3 | 2,2328 | 1 | 1,1778 | 1,90 |
| 22 | 4 | 8,9492 | 1 | 3,7989 | 2,36 |
| 22 | 5 | 29,7756 | 1 | 9,6051 | 3,10 |
| 22 | 6 | 86,4523 | 3 | 20,9259 | 4,13 |
| 22 | 7 | 190,0700 | 4 | 36,4846 | 5,21 |
| 22 | 8 | 386,0630 | 2 | 53,2145 | 7,25 |
| 22 | 9 | 717,3200 | 2 | 69,2120 | 10,36 |
| 22 | 10 | 1131,4200 | 4 | 60,3052 | 18,76 |
| 23 | 1 | 0,0803 | 1 | 0,0702 | 1,14 |
| 23 | 2 | 0,7215 | 1 | 0,3923 | 1,84 |
| 23 | 3 | 3,8206 | 1 | 1,8773 | 2,04 |
| 23 | 4 | 15,8301 | 1 | 6,7158 | 2,36 |
| 23 | 5 | 54,3001 | 1 | 18,0913 | 3,00 |
| 23 | 6 | 157,8760 | 4 | 36,3151 | 4,35 |
| 23 | 7 | 361,9280 | 4 | 58,5770 | 6,18 |
| 24 | 1 | 0,0740 | 0 | 0,0740 | 1,00 |
| 24 | 2 | 0,7839 | 1 | 0,4450 | 1,76 |
| 24 | 3 | 4,7464 | 1 | 2,2609 | 2,10 |
| 24 | 4 | 20,4042 | 1 | 8,8149 | 2,31 |
| 24 | 5 | 76,8693 | 1 | 27,2455 | 2,82 |
| 24 | 6 | 211,1100 | 3 | 64,4327 | 3,28 |
| 24 | 7 | 511,8420 | 3 | 120,5390 | 4,25 |
| 25 | 1 | 0,0690 | 0 | 0,0690 | 1,00 |
| 25 | 2 | 0,7197 | 1 | 0,5250 | 1,37 |
| 25 | 3 | 4,4925 | 1 | 2,6759 | 1,68 |
| 25 | 4 | 21,1283 | 1 | 10,2778 | 2,06 |
| 25 | 5 | 78,1923 | 1 | 30,5250 | 2,56 |
| 25 | 6 | 234,6170 | 3 | 70,6251 | 3,32 |
| 25 | 7 | 592,1420 | 1 | 141,8090 | 4,18 |
| 26 | 1 | 0,1893 | 1 | 0,1729 | 1,09 |
| 26 | 2 | 0,9515 | 1 | 0,6181 | 1,54 |
| 26 | 3 | 5,2128 | 1 | 3,0512 | 1,71 |
| 26 | 4 | 24,4044 | 1 | 12,0056 | 2,03 |
| 26 | 5 | 93,7163 | 1 | 38,0680 | 2,46 |
| 26 | 6 | 335,3130 | 2 | 116,4310 | 2,88 |
| 26 | 7 | 832,2690 | 2 | 209,1360 | 3,98 |
| 27 | 1 | 0,1020 | 1 | 0,1010 | 1,01 |
| 27 | 2 | 1,0133 | 1 | 0,7067 | 1,43 |
| 27 | 3 | 6,8228 | 1 | 3,9057 | 1,75 |
| 27 | 4 | 34,1902 | 1 | 16,7847 | 2,04 |
| 27 | 5 | 144,1780 | 2 | 57,0010 | 2,53 |
| 27 | 6 | 523,8720 | 3 | 157,3380 | 3,33 |
| 27 | 7 | 1417,1700 | 1 | 337,0460 | 4,20 |
| 28 | 1 | 0,0961 | 0 | 0,0961 | 1,00 |
| 28 | 2 | 1,2681 | 1 | 0,7884 | 1,61 |
| 28 | 3 | 8,6796 | 1 | 4,9073 | 1,77 |
| 28 | 4 | 44,8793 | 1 | 20,9715 | 2,14 |
| 28 | 5 | 207,2420 | 3 | 79,9897 | 2,59 |
| 28 | 6 | 695,8890 | 1 | 212,8090 | 3,27 |
| 28 | 7 | 2086,7100 | 5 | 440,0330 | 4,74 |
| 29 | 1 | 0,2234 | 1 | 0,1298 | 1,72 |
| 29 | 2 | 2,1065 | 1 | 1,0359 | 2,03 |
| 29 | 3 | 11,0873 | 1 | 5,8998 | 1,88 |
| 29 | 4 | 61,9728 | 2 | 28,0214 | 2,21 |
| 29 | 5 | 251,1930 | 3 | 102,8600 | 2,44 |
| 29 | 6 | 929,3170 | 3 | 301,7790 | 3,08 |
| 30 | 1 | 0,1448 | 0 | 0,1448 | 1,00 |
| 30 | 2 | 1,7438 | 1 | 1,0705 | 1,63 |
| 30 | 3 | 12,6747 | 2 | 7,2144 | 1,76 |
| 30 | 4 | 72,3704 | 2 | 33,6973 | 2,15 |
| 30 | 5 | 304,0870 | 4 | 130,5950 | 2,33 |
| 30 | 6 | 1202,0400 | 3 | 422,9000 | 2,84 |