МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра дискретной математики и информационных технологий

АЛГОРИТМЫ ПОСТРОЕНИЯ МИНИМАЛЬНЫХ ОСТОВНЫХ ДЕРЕВЬЕВ В КОМПЬЮТЕРНЫХ СЕТЯХ

АВТОРЕФЕРАТ МАГИСТЕРСКОЙ РАБОТЫ

студента 2 курса 271 группы
направления 09.04.01 — Информатика и вычислительная техника
факультета КНиИТ
Кузьмина Романа Александровича

Научный руководитель
д. ф.-м. н. , профессор _____ В. А. Молчанов

Заведующий кафедрой
к. ф.-м. н., доцент _____ Л. Б. Тяпаев

Саратов 2023

ВВЕДЕНИЕ

Идеи теории графов широко используются в компьютерных науках. В частности, в исследовательских областях информатики, таких как интеллектуальный анализ данных, сегментация изображений, кластеризация, захват изображений, создание сетей и т. д. Например, структура данных может быть спроектирована в виде дерева, которое, в свою очередь, использует вершины и ребра. Аналогичным образом моделирование сетевых топологий может быть выполнено с использованием концепций графов. Кроме того, такие понятия как пути, обходы и контуры из теории графов используются в приложениях больших размеров, таких как задачи коммивояжера, концепции проектирования баз данных, сети ресурсов. Это приводит к разработке новых алгоритмов и новых теорем, которые можно использовать в приложениях больших размеров.

В последние годы мы стали свидетелями существенно нового движения в исследованиях, которые так или иначе связаны с теорией графов, когда акцент сместился с анализа отдельных небольших графов и свойств отдельных вершин или ребер внутри таких графов на рассмотрение статистических свойств графов больших размерностей. Этот новый подход во многом обусловлен доступностью компьютеров и сетей связи, которые позволяют нам собирать и анализировать данные в гораздо большем масштабе, чем это было возможно ранее. Если раньше в исследованиях рассматривали графы с десятками, а в крайнем случае и сотнями вершин, то сейчас нередко можно увидеть графы с миллионами или даже миллиардами вершин.

Целью работы является сравнение различных алгоритмов построения минимальных остовных деревьев для моделирования компьютерных сетей на различных наборах данных.

Задачи работы:

- Изучить необходимые теоретические основы теории графов.
- Ознакомиться с основными системами и методами обработки графов больших размерностей.
- Реализовать основные алгоритмы обработки больших графов на системах Giraph и Graphlab.
- Сравнить работу алгоритмов PageRank, нахождения кратчайшего пути и сопоставления с шаблоном на системах обработки графов, упомянутых ранее.

— Изучить основные алгоритмы построения остовных деревьев.
— Изучить и реализовать алгоритмы построения минимальных остовных деревьев.
— Сравнить скорость работы алгоритмов построения минимальных остовных деревьев.

Выпускная квалификационная работа состоит из введения, семи глав, заключения и списка использованных источников. Общий объем работы – 59 страниц. Первая глава имеет название «Теоретические основы», вторая – «MapReduce», третья – «Системы обработки графов», четвертая – «Программная реализация систем обработки графов и их сравнение», пятая – «Остовные деревья», шестая – «Минимальные остовные деревья», седьмая – «Практические результаты»

Результаты исследований, проведенных в данной работе, могут быть использованы в качестве основы для применения рассмотренных алгоритмов в различных практических областях, включающие в себя проектирование сетевой инфраструктуры, маршрутизацию, оптимизацию сетевых протоколов и др.

КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

В первой главе работы приводятся основные определения из теории графов, а также других терминов, используемых по ходу работы. Ключевые понятия из этого раздела:

- *Граф* как математический объект есть совокупность двух множеств — множества самих объектов, называемого множеством вершин, и множества их парных связей, называемого множеством рёбер. Элемент множества рёбер есть пара элементов множества вершин.
- *Сеть (компьютерная сеть)* — система, обеспечивающая обмен данными между вычислительными устройствами — компьютерами, серверами, маршрутизаторами и другим оборудованием или программным обеспечением.
- *Остовное дерево связного неориентированного графа* $G(V, E)$ — это его подграф $T(V, E)$, покрывающий все вершины графа G. Остовное дерево связного графа G также можно определить как максимальное множество ребер графа G, не содержащих циклов, или как минимальное множество ребер, соединяющих все вершины.

Во второй главе рассматривается алгоритм MapReduce. Глава состоит из 2 подразделов.

За последние годы исследователи и разработчики реализовали сотни специализированных вычислений, которые обрабатывают большие объемы "сырых"данных. Большинство таких вычислений концептуально просты. Однако входные данные обычно имеют большой размер, и вычисления приходится распределять по сотням или тысячам машин, чтобы завершить их в разумные сроки. В связи с этим, исследователи и разработчики из Google предложили рассматриваемый в этой главе алгоритм MapReduce.

В первом подразделе этой главы описывается программная модель алгоритма MapReduce.

На вход алгоритму MapReduce подается набор пар ключ/значение и на выходе алгоритма получается также набор пар ключ/значение. Пользователь MapReduce выражает вычисления в виде двух функций: Map и Reduce.

Функция Map, написанная пользователем, берет входную пару и создает набор промежуточных пар ключ/значение. Алгоритм MapReduce группирует вместе все промежуточные значения, связанные с одним и тем же промежуточ-

ным ключом I, и передает их функции Reduce.

Функция Reduce, также написанная пользователем, принимает промежуточный ключ I и набор значений для этого ключа. Эта функция объединяет все эти значения, чтобы сформировать возможно меньший набор значений.

Во втором подразделе главы приводится пример решения проблемы подсчета количества вхождений каждого слова в большой коллекции документов с помощью MapReduce.

В третьей главе рассматриваются системы обработки графов большой размерности. Для решения проблемы производительности среды MapReduce были предложены несколько специализированных платформ, которые предназначены для удовлетворения уникальных требований обработки графов больших размерностей. Эти платформы (системы) предоставляют собой программные абстракции для выполнения итеративного параллельного анализа больших графов в кластерных системах. В данной главе рассматриваются два основных наиболее популярных семейства систем обработки графов, а именно Pregel и GraphLab.

Третья глава состоит из двух подразделов, каждый из которых состоит из трех подпунктов.

В первом подразделе рассматривается семейство обработки графов Pregel. Система Pregel признана первой реализацией BSP, которая предоставляет собственный API специально для программирования графовых алгоритмов с использованием вычислительной парадигмы "мыслить как вершина"("think like a vertex").

В первом подпункте описывается популярный представитель этого семейства Apache Giraph. В Giraph программы обработки графов выражаются в виде последовательности итераций, называемых супершагами.

Во втором подпункту рассматривается другой представитель семейства Pregel – Mizan. Mizan - это проект с открытым исходным кодом, разработанный на C++ компанией KAUST в сотрудничестве с IBM Research.

Во втором подразделе главы описывается семейство GraphLab. В отличие от Pregel, GraphLab полагается на абстракцию разделяемой памяти и GAS (Gather, Apply, Scatter - "Собрать, применить, разбросать").

В первом подпункте подробно описывается сам GraphLab. Во втором подпункте описывается система PowerGraph. PowerGraph представил схему разде-

ления, которая разрезает набор вершин таким образом, что ребра вершины с большой степенью обрабатываются несколькими рабочими.

В четвертой главе приводится программная реализация систем обработки графов больших размерностей, а именно системы Apache Giraph и GraphLab. Помимо этого, производится сравнение производительности разных конфигураций этих двух систем. Глава состоит из шести подразделов.

В первом подразделе приводятся сходства и различия между двумя системами. В целом и Giraph, и GraphLab применяют модель GAS (Gather, Apply, Scatter), которая представляет собой три концептуальных этапа вершинно-ориентированной программы. Однако они отличаются тем, как они собирают и распространяют информацию. Принципиальное различие между Giraph и GraphLab заключается в том, что Giraph опирается на синхронную вычислительную модель, основанную на принципе "push a GraphLab — на асинхронную модель, основанную на принципе "pull".

Во втором подразделе описываются исходные данные для экспериментов. В последующих экспериментах будем использовать набор данных, состоящий из отзывов с популярного веб-сайта Amazon. ля анализа масштабируемости алгоритмов, был использован исходный набор данных для создания трех меньших наборов данных, которые сохраняют характеристики исходного набора данных. В таблице 1 описаны детали четырех наборов данных, использованных в дальнейших экспериментах.

Таблица 1 – Набор данных для экспериментов

Название	Количество узлов	Количество ребер	Размер на диске
Набор данных 1	2,643,669	28,396,350	2 GB
Набор данных 2	4,312,178	34,275,120	4 GB
Набор данных 3	5,165,714	36,275,374	8 GB
Набор данных 4	6,643,669	40,015,189	12 GB

В третьем подразделе описывается экспериментальная нагрузка. Чтобы разнообразить подходы к анализу различных характеристик производительности оцениваемых систем, в качестве рабочей нагрузки были взяты существующие программные реализации следующих трёх основных алгоритмов вычисления и обработки графов:

1. *PageRank*: алгоритм, который присваивает значение каждой вершине графа в соответствии с количеством его входящих/исходящих ребер.

2. *Кратчайший путь*: операция обработки графа для нахождения пути между двумя вершинами в графе, при котором сумма весов (т. е. количества ребер) составляющих его ребер минимальна.
3. *Сопоставление с шаблоном*: операция обработки графа для обнаружения существования определенного подграфа (например, пути, звезды) в большом графе.

В четвертом подразделе описываются характеристики системы, на которой проводились эксперименты.

В пятом подразделе описывается процесс измерения результатов. В Giraph и GraphLab выполнение графовых алгоритмов проходит через три основных этапа: чтение входного потока графа через механизм выполнения, обработка графа и запись результата в виде выходных графов или значений.

В шестом подразделе приводятся результаты экспериментов. Результаты показывают, что Giraph и GraphLab имеют очень сопоставимую производительность либо по общему времени выполнения, либо по времени выполнения трех разных фаз (чтение, обработка и запись), и среди них нет явного победителя. В частности, GraphLab немного превосходит Giraph в задаче PageRank. Однако чем больше размер графа, тем меньше разрыв в производительности между двумя системами.

В пятой главе подробно излагаются основные алгоритмы построения остовных деревьев графа, а также приводятся псевдокоды реализаций алгоритмов. Рассмотренные алгоритмы включают в себя алгоритм лавинной рассылки, асинхронный алгоритм с обнаружением завершения, алгоритм обхода Тарри. Глава состоит из четырех подразделов.

В первом подразделе описывается алгоритм лавинной рассылки. Естественный способ выполнения широковещательной рассылки в сети без какой-либо сформированной структуры состоит в том, чтобы просто пересылать любое входящее сообщение всем соседним узлам, кроме соседа, отправившего сообщение. Также приводится анализ временной сложности алгоритма.

Во втором подразделе рассматривается построение асинхронного остовного дерева на основе лавинной рассылки. Строится он на основе алгоритма Flood с некоторыми модификациями для построения остовного дерева, исходящего из корня инициатора для широковещательной рассылки. Предположим, что требуется, чтобы каждый узел в дереве, кроме листовых, знал идентифика-

торы своих потомков, а все узлы, кроме корня, в конце должны были знать своих родителей. Любой узел, который хочет построить широковещательное дерево, инициирует алгоритм и становится корнем формируемого остовного дерева. К этому алгоритму также прилагается анализ временной сложности.

В третьем подразделе рассматривается асинхронный алгоритм с обнаружением завершения. В качестве дальнейшей попытки построить остовное дерево асинхронно предыдущий алгоритм изменяется так, чтобы завершение построения обнаруживалось узлами. Модификация достигается за счет того, что узлы откладывают отправку сообщения подтверждения своим родителям до тех пор, пока они не получат сообщения подтверждения или отклонения от своих соседей, вместо того, чтобы немедленно ответить своим родителям. Таким образом, когда узел получает все ответы от своих соседей, он может определить, что все узлы в поддереве, в котором он является корнем, прекратили работу. В конце подраздела приводится пример построения остовного дерева по алгоритму, а также анализ временной сложности.

В четвертом подразделе рассматривается алгоритм обхода Тарри. Алгоритм Тарри Tarry — один из самых первых распределенных алгоритмов, который строит остовное дерево путем обхода токена с использованием двух простых правил, показанных ниже. Как и прежде, существует назначенный корневой узел, и когда узел i впервые получает токен от узла j, он помечает его как своего родителя. Поскольку включен только узел с токеном, в любое время существует единственная точка активности.

1. Процесс никогда не пересылает токен дважды по одному и тому же каналу.
2. Неинициатор пересылает токен своему родителю, узлу, от которого он получил токен в первый раз, только если не осталось другого канала согласно Правилу 1.

В шестой главе рассматриваются классические последовательные алгоритмы построения минимальных остовных деревьев, в частности алгоритм Крускала и алгоритм Прима. Помимо этого, в главе подробно описываются различные модификации классический последовательных алгоритмов. Глава состоит из четырех подразделов.

В первом подразделе описываются последовательные алгоритмы построения минимальных остовных деревьев. В частности, описываются классические алгоритмы Прима и Крускала.

Во втором подразделе описывается модификация классического алгоритма Прима, а именно синхронный распределенный алгоритм Прима. В данном случае, существует единственный инициатор, который завершает структурирование дерева за несколько раундов. В каждом раунде корень собирает все ИРМВ из листьев частичного дерева T, находит минимум $\{u, v\}$ этих ИРМВ и передает $\{u, v\}$ в T в следующем раунде, чтобы его можно было добавить к T.

Каждый узел, кроме корня, может находиться в промежуточном (interm) узле, в листе ($leaf_p$) частичного дерева T, в узле из $\{V \setminus T\}$ (other) или в состоянии конечного листа ($leaf_f$).

В конце подраздела приводится анализ временной сложности алгоритмов.

В третьем подразделе описывается синхронный алгоритм GHS. Галлагер, Хамбельт и Спира (Gallager, Humblet, Spira) предложили синхронный алгоритм (SGHS_MST) с параллельными инициаторами для нахождения МОД графа. Общая идея этого алгоритма состоит в том, чтобы инициаторы расширяли свои собственные фрагменты синхронно поэтапно, находя свои исходящие ребра минимального веса (далее ИРМВ) фрагментов и соединяя фрагменты параллельно. Этот алгоритм требует синхронизации каждой фазы, которая может быть обеспечена синхронизатором, или может быть заранее построено остовное дерево всей сети, как в предыдущем алгоритме Prim_Synch за счет дополнительных сообщений. Кроме того, в отличие от синхронных алгоритмов, которые были рассмотрены ранее, этот алгоритм также требует синхронизации всех фрагментов в раунде, так что узлы в каждом фрагменте должны одновременно выполнять один и тот же раунд.

В четвертом подразделе описывается асинхронный алгоритм GHS. Используя ту же идею независимого "выращивания"фрагментов и их соединения, Галлагер, Хамбельт и Спира предложили асинхронную версию алгоритма SGHS_MST, которая называется AGHS_MST. Общая идея алгоритма состоит в том, что каждый узел i является фрагментом, изначально содержащим сам себя.

В седьмой главе приводятся результаты самостоятельно проведенных экспериментов по сравнению производительности классических последовательных алгоритмов Прима, Крускала, а также параллельных алгоритмов Прима, синхронного алгоритма GHS и асинхронного алгоритма GHS. Сравнение производится на 6 различных наборах данных, два из которых представляют собой

данные из реальной жизни. Также сравнение параллельных алгоритмов проводится на 1 и на 144 потоках.

В таблицах 2 и 3 приведены результаты исследований.

Набор данных	Prim_MST	Kruskal_MST
rand_20M	33,041	18,3
uniform_20M_ 20M	12,52	3,7
stars_20M	25,281	3,285
chain_20M	0,246	1,315
USA_roads	6,159	6,086
livejournal	4,112	5,459

Таблица 2 – Время выполнения последовательных алгоритмов в секундах.

Набор данных	DistPrim_MST	AGHS_MST	SGHS_MST	DistPrim_MST	AGHS_MST	SGHS_MST
#потоков	1	1	1	144	144	144
rand_20M	30,6	34,5	54,823	1,026	0,765	2,564
uniform_20M_20M	5,569	9,369	18,509	0,207	0,242	2,292
stars_20M	2,29	8,143	2,305	0,08	0,176	0,1
chain_20M	0,66	1,623	1,475	0,04	0,163	3,2
USA_roads	3,645	12,21	11,025	0,171	0,329	2,02
livejournal	3,6	10,57	14,941	0,153	0,234	4,05

Таблица 3 – Время выполнения параллельных алгоритмов с использованием 1 и 144 потоков в секундах. Обозначения: алгоритм Прима (DistPrim_MST), синхронный алгоритма GHS (SGHS_MST) и асинхронный алгоритма GHS (AGHS_MST)

ЗАКЛЮЧЕНИЕ

В работе рассмотрены различные алгоритмы для построения остовного дерева T графа G, в том числе синхронные и асинхронные.

Результаты экспериментов показали, что модернизация существуюших классических алгоритмов может существенно ускорить скорость их выполнения. В частности, асинхронный алгоритм GHS показал наибольшее ускорение среди других модификаций.

Полученные результаты исследований алгоритмов построения минимальных остовных деревьев докладывались на Студенческой Научной Конференции факультета компьютерных наук и информационных технологий, а также на научной конференции механико-математического факультета «Актуальные проблемы математики и механики». Практические результаты также подготовлены в форме статьи для публикации в сборнике "Математика. Механика".

Основные источники информации:

1. Kshemkalyani, A. D. Distributed computing: principles, algorithms, and systems / A. D. Kshemkalyani, M. Singhal. — Cambridge University Press, 2011.
2. J Dean, S Ghemawa,. Mapreduce: simplified data processing on large clusters / J Dean, S Ghemawa // OSDI. — 2004. — Vol. 1. — Pp. 137–150.
3. Sakr, S. The family of mapreduce and large-scale data processing systems / S. Sakr, A. Liu, A. G. Fayoumi // ACM Computing Surveys. — 2013. — Vol. 46, no. 1. — Pp. 1–44.
4. Sakr, S. Graph indexing and querying: a review / S. Sakr, G. Al-Naymat // International Journal of Web Information Systems. — 2010. — Vol. 6, no. 2. — Pp. 101–120.
5. Valiant, L. G. A bridging model for parallel computation / L. G. Valiant // Communications of the ACM. — 1990. — Vol. 33, no. 8. — Pp. 103–111. 6 Clinger, W. D. Foundations of Actor Semantics / W. D. Clinger. — 1981.
6. Low, Y. Distributed graphlab: A framework for machine learning in the cloud / Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, J. M. Hellerstein // Proceedings of the VLDB Endowment (PVLDB), Vol. 5, No. 8, pp. 716-727
7. Erciyes, K. Distributed graph algorithms for computer networks / K. Erciyes. — 2013.
8. Amir, T. Message complexity of population protocols / T. Amir, J. Aspnes, D. Doty, M. Eftekhari, E. Severson // arXiv preprint arXiv:2003.09532. — 2020.