Санкт-Петербургский государственный университет

Кафедра системного программирования

Теория графов. Презентация 1

Команда 5: Аверин Павел, Кузнецов Арсений, Якшигулов Вадим

Санкт-Петербург
2025

Остовные деревья. Алгоритм Борувки

Поставновка задачи

Дан неориентированный взвешенный граф $\mathcal{G}$. Требуется найти **минимальное остовное дерево** в $\mathcal{G}$.

Определение

Остовный подграф — подграф, содержащий все вершины графа $\mathcal{G}$.

Определение

Остовное дерево — остовный подграф $\mathcal{G}$, являющийся деревом.

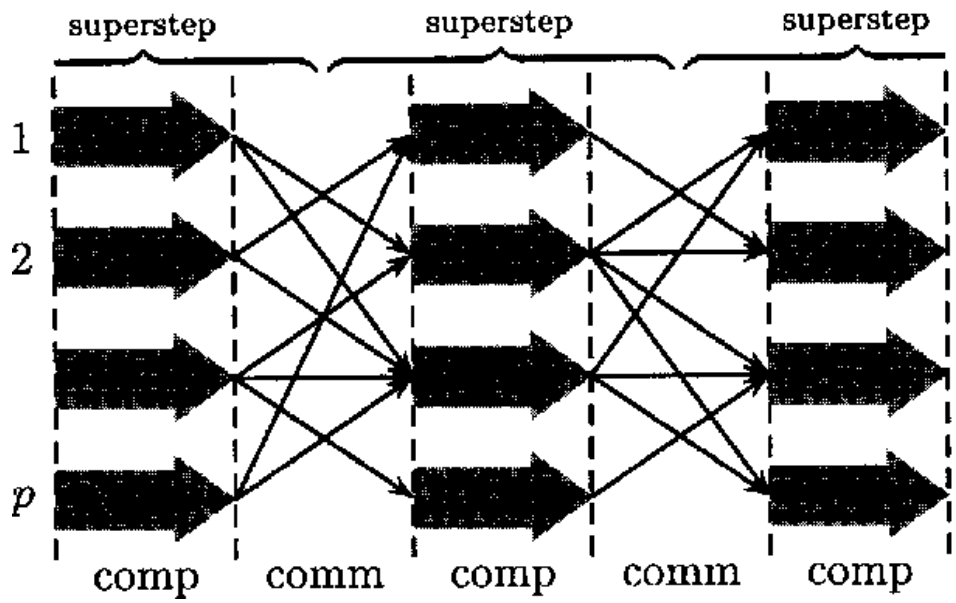
Определение

Минимальное остовное дерево — остовное дерево $\mathcal{G}$ минимального веса.

Алгоритм Борувки в контексте Google Pregel (1 из 4)

Напоминание про BSP

- Супершаги (Supersteps)
- Вычисления
- Отправка сообщений
- Синхронизация



Алгоритм Борувки в контексте Google Pregel (2 из 4)

Вершины инициализируются как тривиальные деревья, и указывают на себя как на корень.
Супершаг 0. Информация о ребрах.

- Каждая вершина (компонента) отправляет сообщения соседним вершинам (компонентам) с весами инцедентных с ней ребер

Алгоритм Борувки в контексте Google Pregel (3 из 4)

Супершаг 1. Нахождение минимальных ребер.

- Каждая вершина (компонента) из сообщений выбирает ребро с минимальным весом и начинает указывать на новый корень
- Вершины (компоненты) отправляют сообщения соседям с новым корнем, на который они указывают

Алгоритм Борувки в контексте Google Pregel (4 из 4)

Супершаг 2. Формирование новых компонентов.

- Разрешение ситуаций, когда вершины (компоненты) указывают друг на друга и формирование новых компонент
- Помечаем ребра из каждой (компоненты) как часть MST

Повторение супершагов 1-2 до тех пор, пока в MST не перестанут появляться новые ребра

Алгоритм Борувки в контексте GraphBLAS и ЛА (1 из 3)

С чем работаем:

- $\mathcal{S}$ – матрица смежности графа
- (w, i) – ребро
  - w – вес ребра
  - i – номер компоненты, с которой оно соеденено
- Вектор edge, где edge[v] – минимальное ребро, которое инцидентно v
- Вектор cedge, где cedge[v] – минимальное ребро, которое инцидентно одному из потомков v (INT_MAX если v не корень)
- Вектор, где parent[v] – корневая вершина в дереве, которому пренадлежит v

Алгоритм Борувки в контексте GraphBLAS и ЛА (2 из 3)

Полукольцо combMin:

- Множество ребер
- Операция сложения — `min`
- Операция умножения — `combine`, которая определена как `combine((w1, i1), (w2, i2)) = (w1, i2)`
- В этом полукольце считаем edge как $\mathcal{S} \times$ `parent`

Далее cedge просто считаем в цикле как
`cedge[parent[i]] := min(cedge[parent[i]], edge[i])`

Алгоритм Борувки в контексте GraphBLAS и ЛА (3 из 3)

Добавление в MST:

- Каждая вершина пытается узнать, является ли какое-то её ребро минимальным. Для каждого корня выбираем потомок с минимальным ребром (С помощью edge и cedge)
- Обновляем parent, чтобы отразить новые компоненты. Также помечаем выбранное ребро как часть MST.
- Оставляем в $\mathcal{S}$ только ребра между разными деревьями
- Повторяем процесс до тех пор пока $\mathcal{S} \neq 0$

Набор данных (1 из 2)

В ходе эксперемента предлагается сравнить производительность реализаций алгоритма Борувки на Google Pregel и GraphBLAS. Для данного эксперемента были выбраны графы дорожных сетей США DIMACS.

Почему именно эти графы?

- Прикладное значение
- Использование в научных статьях[1] и бенчмарках
- Вариативность

[1] https://ieeexplore.ieee.org/abstract/document/7092783

Набор данных (2 из 2)

Name	Description	Nodes	Edges
USA	Full USA	23,947,347	58,333,344
CTR	Central USA	14,081,816	34,292,496
W	Western USA	6,262,104	15,248,146
E	Eastern USA	3,598,623	8,778,114
LKS	Great Lakes	2,758,119	6,885,658
CAL	California and Nevada	1,890,815	4,657,742
NE	Northeast USA	1,524,453	3,897,636
NW	Northwest USA	1,207,945	2,840,208
FLA	Florida	1,070,376	2,712,798
COL	Colorado	435,666	1,057,066
BAY	San Francisco Bay Area	321,270	800,172
NY	New York City	264,346	733,846

Эксперемент

Ход эксперемента:

- Провести несколько запусков алгоритмов
- Составить информацию о среднем значении и доверительных интервалах
- Провести анализ полученных результатов

Характеристики вычислительной машины:

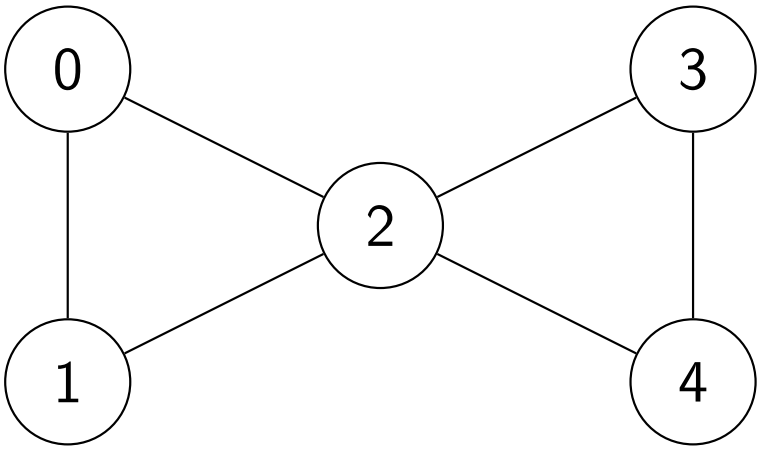
- Процессор: 12th Gen Intel Core i7-12700H
- RAM: 16 GB
- Операционная система: Linux Mint 21.3

Следующая задача

Подсчет треугольников. Алгоритмы Sandia и Burkhardt.

Постановка задачи подсчета треугольников

Для неориентированного графа G = (V, E), без петель и кратных ребер, нужно найти количество треугольников, то есть троек вершин, попарно соединенных ребрами.



Например, в данном графе два треугольника.
(0, 1, 2) и (2, 3, 4)

Кому нужна задача подсчёта треугольников?

В анализе социальных сетей подсчет треугольников используется для вычисления коэффициента кластеризации — метрики, показывающей, насколько узлы склонны образовывать плотные группы.
Twitter и Facebook используют эту метрику для анализа формирования сообществ и оценки "эффекта эхо-камеры".

Базовый алгоритм

Пусть A — это матрица смежности, тогда A^3 представляет все пути длины 3.
trace — след, сумма всех значений на главной диагонали

$$\frac{\mathrm{trace}(A^3)}{6}$$

Burkhardt

Заметим, что в базовом алгоритме последнее умножение можно заменить более простым поэлементным умножением. Это означает, что мы находим пути длиной 2 от вершины u до вершины v и проверяем наличие пути длиной 1 между ними.

$$\frac{\sum\left(A^2 .* A\right)}{6}$$

- $.*$ — поэлементное умножение
- $\sum$ — сумма всех элементов

Sandia

Алгоритм Sandia использует верхнетреугольную (также можно нижнетреугольную) матрицу U (так как исходная матрица смежности A симметрична, её можно разделить на верхнюю и нижнюю треугольные части). Тогда формула для подсчёта количества треугольников имеет вид:

$$\sum \Big((U \times U) .* U \Big)$$

где:

- $\times$ — стандартное умножение матриц
- $.*$ — поэлементное умножение
- $\sum$ — сумма всех элементов

Набор данных

В ходе эксперемента предлагается сравнить производительность реализаций подсчета треугольников.
Для данного эксперемента были выбраны реальные графы из датасета Стэнфордского университета

Почему именно эти графы?

- Прикладное значение
- Использование в научных статьях[2] и бенчмарках
- Вариативность

[2] Leskovec, J. and Sosič, R. (2016). *SNAP: A General-Purpose Network Analysis and Graph-Mining Library*.

Выбранные графы

Имя	Вершины $\times 10^3$	Ребра $\times 10^3$	Описание
loc-brightkite_edges	58	214	Граф локационной соцсети Brightkite.
amazon0302	262	1 234	Ко-покупательский граф Amazon (03/2003).
amazon0505	410	3 356	Ко-покупательский граф Amazon (05/2003).
roadNet-PA	1 088	1 541	Дорожная сеть Пенсильвании.
musae-twitch	34	429	Социальная сеть пользователей Twich
musae-github	37	289	Социальная сеть пользователей Github
soc-Epinions1	75	508	Граф доверия соцсети Epinions.
email-EuEnron	36	183	Email-коммуникации Enron.
loc-gowalla_edges	196	950	Граф локационной соцсети Gowalla.

На библиотеках SPLA и Gunrock.

SPLA

В SPLA уже содержит модифицированную версию алгоритма Sandia. Burkhardt же будет реализованы вручную с использованием функциональности SPLA для операций над разряженными матрицами.
Для представления графов в эксперименте будет использован формат матрицы acceleration CSR.

Gunrock

Для эксперимента будет взята библиотека GraphBLAST, использующая gunrock в качестве backend-а. В ней есть таким же, как в SPLA, образом реализованный алгоритм Sandia, но также нет Burkhardt.

Эксперимент

Ход эксперимента:

- Провести несколько запусков алгоритмов на GPU
- Составить информацию о среднем значении и доверительных интервалах
- Сравнить результаты для двух алгоритмов на двух библиотеках

Характеристики вычислительной машины:

- Процессор: AMD Ryzen 5 4600H (6 ядер, 12 с гипертредингом)
- RAM: 16 GB
- GPU: RTX 2060 (6 ГБ видеопамяти)
- Операционная система: Windows 10 22H2

На библиотеках SuiteSparse:GraphBLAS и Apache Spark.

Подсчет треугольников в SuiteSparse:GraphBLAS

SuiteSparse:GraphBLAS предоставляет эффективные операции для работы с матрицами.
Так

$$(A^2 .* A)$$

может быть посчитан за одну операцию

```
GrB_mxm(C, A, NULL, GxB_PLUS_TIMES_UINT32, A, A, d);
```

А верхняя треугольная матрица вычислена через смещение на 1 от главной диагонали.

```
GxB_select(U, NULL, NULL, GxB_TRIU, A, 1, NULL);
```

Подсчет треугольников в Apache Spark

Apache Spark внутри GraphX реализует алгоритм подсчета треугольников, но это не Sandia или Burkhardt.
Можно использовать MLlib, чтобы использовать матрицы, а не абстракцию `Graph[VD, ED]`. `VertexSet` для эффективного хранения вершин.

Burkhardt: эффективное вычисление общих соседей через пересечение множеств
Sandia: работа только с верхнетреугольной частью графа требует фильтрации ребер

- `collectNeighborIds` для эффективного сбора соседей всех вершин
- `aggregateMessages` для распространения информации по графу
- `triplets` для доступа к тройкам (исходная вершина, ребро, целевая вершина)

Эксперимент

Ход эксперимента:

- Рассмотреть как несколько нод, так и единственную
- Провести несколько запусков алгоритмов
- Составить информацию о среднем значении и доверительных интервалах
- Сравнить результаты для двух алгоритмов на двух библиотеках

Характеристики вычислительной машины:

- Процессор: Apple M3 Pro (12 ядер, гипертредининга нет)
- L1-кэш: 256 KB
- L2-кэш: 36 MB (32 MB big, 4 MB LITTLE)
- L3-кэш: 24 MB
- RAM/VRAM: 36 GB
- GPU: Apple M3 GPU с 18 ядрами
- Операционная система: Sonoma 14.6.1