Теория графов. Первая презентация

Ермолович Анна, Шмаков Александр, Зайцев Дмитрий

Введение

Остовное дерево — дерево, подграф исходного графа, имеющий такое же число вершин.

Минимальное остовное дерево — остовное дерево взвешенного графа, имеющее минимальный возможный вес.

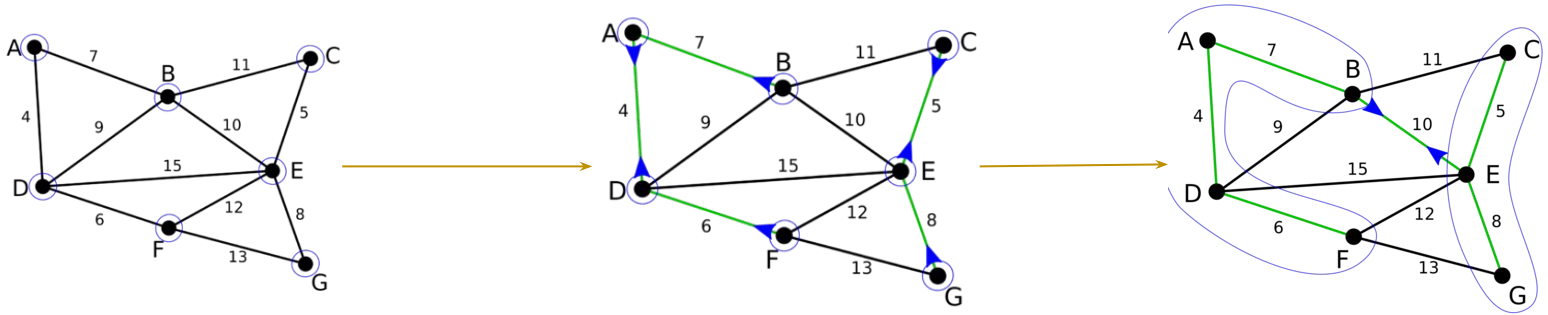
Алгоритм Борувки — алгоритм нахождения минимального остовного дерева в графе.

Алгоритм Борувки

ВХОД: Взвешенный граф G = (V, E).

ВЫХОД: Подграф F, минимальное остовное дерево.

1. Каждая вершина графа G — дерево, никакие ребра им не принадлежат.
2. Каждому дереву добавим минимальное инцидентное ему ребро.
3. Повторяем предыдущий шаг, пока не останется только одна компонента связности.



Борувка и линейная алгебра

Ребро — (вес, номер компоненты связности).
S — матрица смежности.
edge — вектор, содержащий минимальные рёбра инцидентные каждой вершине.
cedge — вектор, содержащий минимальные рёбра для потомков каждой вершины (только для корней деревьев).
parent — вектор, содержащий корень дерева для каждой вершины.

Combine — упаковывает вес и номер компоненты связности в единое значение (w, i).
Min — находит минимальное ребро.
CombMin — полукольцо из этих операций.

Борувка и линейная алгебра 2

Шаги:

1. Каждая вершина изначально является своим собственным родителем *parent[i] = i*.
2. Находим вектор *edge* как *S x parent*, используя операции полукольца.
3. Находим *cedge[parent[i]] = min(cedge[parent[i]], edge[i])*.
4. Обновляем *parent*, объединяя компоненты связности.
5. Удаляем из *S* рёбра, находящиеся в одной компоненте связности.
6. Повторяем пункты *2-5*, пока *S* не опустеет.

$$S = \begin{pmatrix} \infty & 5 & 3 & \infty & \infty \\ 5 & \infty & 2 & 6 & \infty \\ 3 & 2 & \infty & 4 & 8 \\ \infty & 6 & 4 & \infty & 1 \\ \infty & \infty & 8 & 1 & \infty \end{pmatrix}$$

$$\text{parent} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \end{pmatrix}^T$$

$$\begin{pmatrix} \infty \cdot 0 + 5 \cdot 1 + 3 \cdot 2 + \infty \cdot 3 + \infty \cdot 4 \\ 5 \cdot 0 + \infty \cdot 1 + 2 \cdot 2 + 6 \cdot 3 + \infty \cdot 4 \\ 3 \cdot 0 + 2 \cdot 1 + \infty \cdot 2 + 4 \cdot 3 + 8 \cdot 4 \\ \infty \cdot 0 + 6 \cdot 1 + 4 \cdot 2 + \infty \cdot 3 + 1 \cdot 4 \\ \infty \cdot 0 + \infty \cdot 1 + 8 \cdot 2 + 1 \cdot 3 + \infty \cdot 4 \end{pmatrix} = \begin{pmatrix} (3,2) \\ (2,2) \\ (2,1) \\ (1,4) \\ (1,3) \end{pmatrix}$$

Gunrock

Gunrock — это высокопроизводительная библиотека для обработки графов на GPU с использованием CUDA, которая упрощает разработку параллельных графовых алгоритмов за счет встроенных в библиотеку готовых механизмов.

Почему Gunrock подходит для реализации алгоритма?

Основная часть алгоритма Борувки (работа с очередным фронтом соседних вершин/ребер, поиск минимального исходящего ребра в компоненте) может быть эффективно реализована с помощью стандартных средств библиотеки

Борувка и Gunrock

Шаги:

1. Загрузка графа.
2. Инициализация меток компонент — каждой вершине присваивается своя метка.
3. Основной цикл (пока >1 компонент):

   - `advance` — построение фронта ребер от текущих компонент.
   - `filter` — выбор минимального ребра для каждой компоненты, ребра сохраняются.
   - обновление меток компонент по выбранным минимальным ребрам.
   - исключение внутренних ребер – удаление ребер, соединяющих вершины одной компоненты

4. Накопление рёбер `MST` — выбранные минимальные рёбра накапливаются на каждом шаге.
5. Завершение, когда осталась одна компонентная метка.

Термины и операции:

1. `advance` – создает новый фронт из текущего путем обхода соседних элементов, работая либо с вершинами, либо с ребрами
2. `filter` – создает новый фронт из текущего путем выбора подмножества текущего фронта на основе заданного критерия
3. `MST` – Minimum Spanning Tree

Pregel+

Pregel+ — это улучшенная распределённая система для обработки графов, основанная на модели Pregel. Она сокращает объем передаваемых данных с помощью vertex mirroring и request-respond, обеспечивая эффективные вычисления на больших графах.

Почему Pregel+ подходит для реализации алгоритма:

- Эффективное сокращение сообщений:
  - Vertex mirroring уменьшает количество сетевых запросов.
  - Request-respond позволяет запрашивать только необходимую информацию, вместо массовой рассылки.
- Модель BSP (Bulk Synchronous Parallel) обеспечивает четкую организацию супершагов.
- Поддержка распределенных вычислений для работы с большими графами.

Борувка и Pregel+

Шаги:

1. *Инициализация:* каждая вершина становится отдельной компонентой и указывает на себя как на корень. Используется зеркалирование вершин — вершины хранят копии соседей локально, что сокращает количество сетевых запросов.
2. *Поиск минимального ребра:* каждая вершина запрашивает данные у соседей и выбирает минимальное исходящее ребро. Оптимизировано за счет локальных копий (зеркалирования), что снижает сетевой трафик.
3. *Объединение компонент:* вершины обновляют родительские метки и начинают указывать на новый корень. Конфликты (циклы) разрешаются локально, уменьшая количество супершагов.
4. *Обновление структуры:* вершины синхронизируют информацию о своих компонентах, помечая рёбра, входящие в остовное дерево (MST). Используется оптимизированный механизм передачи сообщений (запрос-ответ).
5. *Итерация:* шаги 2-4 повторяются, пока все вершины не объединятся в одну компоненту и в MST не перестанут появляться новые ребра.

Датасет DIMACS 9th

Description	Nodes	Edges
Great Lakes	2,758,119	3,397,404
California and Nevada	1,890,815	2,315,222
Northeast USA	1,524,453	1,934,010
Northwest USA	1,207,945	1,410,387
Florida	1,070,376	1,343,951
Colorado	435,666	521,200
San Francisco Bay Area	321,270	397,415
New York City	264,346	365,050

Эксперимент 1

Цель: Попробуем оценить, как плотность графа влияет на производительность библиотек.

Ход эксперимента:

1. Берём граф из датасета.
2. Начинаем *менять количество рёбер* в нём так, чтобы можно было построить график зависимости производительности библиотеки от плотности графа (5%, 20%, … от исходного числа рёбер).
3. Сохраняем эти графы в том же формате, что и исходные графы и используем для тестов.
4. Проверяем, будет ли сохраняться эта зависимость на всех графах из датасета.

Проблемы:

- Уменьшается практическая ценность графа.
- Может получиться, что на одном графе у нас зависимость чёткая, а на другом нет.

Как будем менять графы

Какие рёбра лучше не трогать:

- Мосты, найдём их заранее.
- Рёбра с весами сильно выше среднего, скорее всего это важные дороги.
- С высокой мерой "центральности", они точно важны в графе.

Как это эффективно вычислить:

- Работать с подграфами, а не всем графом.

Как контролировать, не испортили ли мы граф:

- Graph kernel поможет сравнивать *варианты* и выбрать лучший граф с точки зрения сохранения практической ценности.

Эксперимент 2

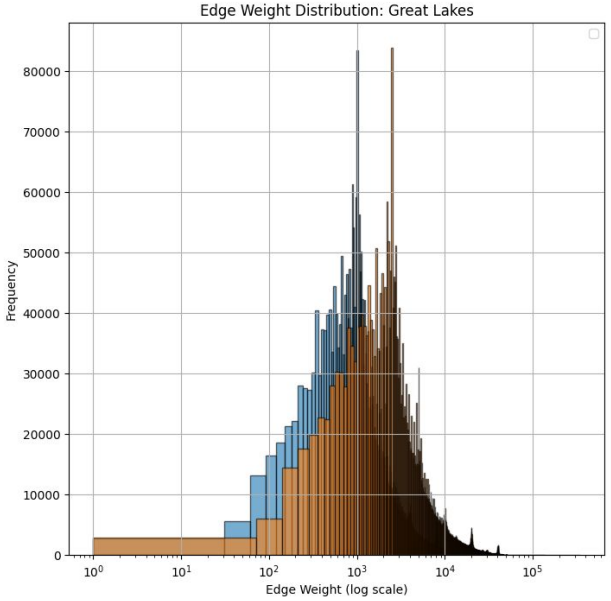
Цель: Попробуем оценить, как распределение весов влияет на производительность библиотек.

Ход эксперимента:

1. Берём из датасета пару: одинаковые по топологии графы с разным распределением весов.
2. Запускаем алгоритм и смотрим, будет ли различие в производительности (за вычетом погрешности).
3. Проверяем, будет ли сохраняться разница на всех парах графов из датасета.

Проблемы:

- Может получиться, что на одной паре графов результат есть, а на другой нет.



Эксперимент 3

Цель: Посмотрим, при каком количестве вычислительных узлов (контейнеров) на нашей вычислительной машине будет оптимальная производительность Pregel+ для “среднего” графа из датасета.

Ход эксперимента:

1. Берём граф из датасета.
2. Запускаем Pregel+ с 1, 2, 4, 8 и 16 контейнерами на этом графе.
3. Измеряем время выполнения (смотрим через `docker inspect`).
4. Оцениваем результаты для всех графов.

Проблемы:

- Может получиться, что для разных графов результаты будут отличаться.
- Разные контейнеры могут завершить работу в разное время.
- Погрешности при измерениях из-за docker.

Немного подробнее

Эксперименты 1-2:

1. Установим гипотезы (например, H_0 — нет зависимости между свойством графа “X” и скоростью библиотеки “Y”).
2. Вычислим необходимые значения на графах (как было описано на слайдах до этого) и сохраним их.
3. Чтобы определить, что есть зависимость, используем коэффициент корреляции (например, r близко к 0 — не получилось опровергнуть гипотезу H_0 для данной группы графов, r далеко от 0 и P-значение мало — доказали гипотезу H_1).
4. Получаем табличку с результатами анализа для каждой группы графов: получилось ли установить зависимость, коэффициент корреляции и P-значение.
5. Делаем мета-анализ результатов и какой-то вывод.

Эксперимент 3:

1. Для каждого графа находим *оптимальное* число вычислительных узлов (с учётом погрешностей, построением доверительных интервалов).
2. Визуализируем результаты и делаем выводы.
3. Фиксируем число контейнеров для остальных экспериментов.

Характеристики вычислительной машины

- **Процессор**: AMD Ryzen 7 5800H (8 ядер, 16 потоков)
  - L1-кэш: 64 KB (на ядро)
  - L2-кэш: 512 KB (на ядро)
  - L3-кэш: 16 MB (общий)
- **RAM**: 32 GB
- **GPU**: RTX 3070 (8 GB видеопамяти)
- **Операционная система**: Ubuntu 24.04.2