Теория графов. Первая презентация

Ермолович Анна

Введение

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

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

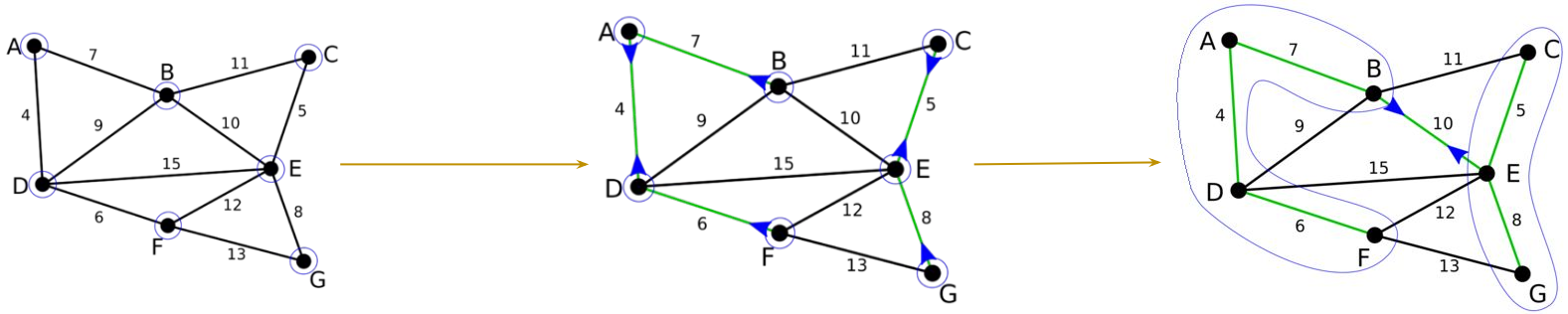
Алгоритм Борувки — алгоритм нахождения минимального остовного дерева в графе.

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.



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
- **Операционная система**: Ubuntu 24.04.2

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

Эксперимент

Цель: Исследовать масштабируемость Pregel+ для алгоритма Борувки — как изменяется время выполнения и эффективность при увеличении числа процессов — на разных графах из датасета.

Ход эксперимента:

1. Для каждого выбранного графа из датасета запускаем Pregel+ с использованием mpirun на 1, 2, 4, 8 и 16 процессах.
2. Измеряем время выполнения, фиксируем результаты.
3. Строим графики для каждого графа:
   a. Зависимость времени выполнения от числа процессов
   b. Зависимость ускорения и эффективности от числа процессов
4. Сравниваем результаты и делаем выводы о масштабируемости Pregel+.

Проблемы:

- Масштабируемость может различаться для разных графов (разная структура и размер).
- Из-за параллельной природы вычислений возможны колебания времени между запусками.

Метрики оценки масштабируемости

Масштабируемость — это способность системы к увеличению производительности при добавлении новых ресурсов или способность эффективно перераспределять имеющиеся ресурсы при увеличении нагрузки.

Ускорение:

Показывает, во сколько раз выполнение стало быстрее при p процессах, чем при 1:

$$S(p) = \frac{T(1)}{T(p)}$$

Эффективность:

Показывает, насколько эффективно используются вычислительные ресурсы:

$$E(p) = \frac{S(p)}{p}$$

Критерии анализа:

1. Строим графики зависимости $T(p), S(p), E(p)$ от p.
2. Ищем, при каком p эффективность заметно падает *(например, < 0.5)*.
3. Оцениваем влияние аппаратной многопоточности (SMT) на масштабируемость Pregel-алгоритмов.

Реализация на Pregel+

Pregel+

Pregel+ — библиотека для параллельной обработки графов по модели Bulk Synchronous Parallel (BSP).
Она оптимизирует передачу сообщений между вершинами за счет:

- *Vertex mirroring* — локального хранения информации о соседях,
- *Request-respond* — отправки сообщений только при необходимости, а не всем вершинам подряд.

Почему Pregel+ подходит для реализации алгоритма:

- Поддерживает параллельное выполнение алгоритмов на графах и легко масштабируется на разное число процессов.
- Снижает накладные расходы на передачу сообщений между процессами за счет *vertex mirroring* и *request-respond*.

Pregel+ и MPI

Как будем обеспечивать распределенность на одной машине:

1. Мы будем использовать mpirun для запуска нескольких независимых процессов на одной машине.
2. Каждый процесс имеет свою отдельную память. Общей памяти между процессами нет.
3. Pregel+ внутри использует MPI для обмена сообщениями между worker`ами.
4. Таким образом, запрос-ответ (request-respond) и зеркалирование вершин (vertex mirroring) продолжают работать, просто передача сообщений идёт через MPI-сообщения.

Это означает, что даже без кластера мы можем корректно изучать поведение Pregel+ при разном числе процессов

Борувка и Pregel+

Шаги:

1. *Инициализация:* каждая вершина становится отдельной компонентой и указывает на себя как на корень. Pregel+ создает локальные копии (зеркала) соседних вершин, чтобы сократить количество обменов сообщениями между процессами.
2. *Поиск минимального ребра:* каждая вершина получает данные о своих соседях из локальных копий и выбирает минимальное исходящее ребро. Это уменьшает количество межпроцессных сообщений.
3. *Объединение компонент:* вершины обновляют родительские метки и начинают указывать на новый корень. Конфликты (циклы) разрешаются локально в каждом процессе, что сокращает число итераций.
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 во всех компонентах не перестают появляться новые рёбра.