Федеральное государственное образовательное бюджетное учреждение
высшего профессионального образования
«Нижегородский Государственный Университет им. Н.И.Лобачевского»
(ННГУ)

Институт Информационных Технологий Математики и Механики

Задача о минимальном остовном дереве

Алгоритм Прима на 2-куче и алгоритм Краскалла на массиве

Выполнил:
студент группы 3821Б1ФИ2
Казанцев Е. А.
Проверил: Уткин Г. В.

2 ноября 2023 г.

Содержание

1 Введение

В этой работе реализованы два алгоритма поиска минимального остовного дерева в полном, ориентированном, взвешенном графе. Алгоритм Прима на 2-куче и алгоритм Краскалла на массиве.
Минимальным остовным деревом (МОД) связного взвешенного графа называется его связный подграф, состоящий из всех вершин исходного дерева и некоторых его ребер, причем сумма весов ребер минимально возможная. Если исходный граф несвязен, то описываемую ниже процедуру можно применять поочередно к каждой его компоненте связности, получая тем самым минимальные остовные деревья для этих компонент.

2 Цель работы

Целью данной работы является сравнение двух алгоритмов поиска минимального остовного дерева во взвешенном неориентированном графе, реализованных на различных структурах данных. Сравнение асимптотической сложности и замеры времени работы. Для сравнения будут проведены эксперименты с различным количеством вершин в графе и различными весами, задаваемыми вручную, в функции генерации графов.

3 Постановка задач

3.1 Написать классы бинарной кучи и графов.

В бинарной куче длжны быть реализованы:

1. Метод всплытия элемента
2. Погружениz элемента
3. Добавление элемента
4. Удаление элемента
5. Возвращение значения элемента (weight)

Класс графа и дополнительные структуры:

1. Вспомогательная структура Edge
2. Вспомогательная структура UnionFind

3. Добавление ребер в граф
4. Генератор случайного, связного, полного графа
5. Алгоритм поиска минимального остовного дерева Prima
6. Алгоритм поиска минимального остовного дерева Kruskal

4 Руководство программиста

4.1 Описание структуры программы

Программа состоит из одного решения.

В решении min_spanning_tree определено 4 модуля main.cpp, Graph.h, UnionFind.h, 2_heap.h, Timer.h

- В модуле main.cpp определена стандартная функция int main(), где идет работа с остальными модулями.
- В модуле Graph.h определен класс Graph, а также объявлены все его методы, определения которых, вынесены в отдельный файл Graph.cpp.
- В модуле 2_heap.h определен класс 2_heap, и также, как в Graph все его методы вынесены в отдельный файл 2_heap.cpp.
- В модуле Timer.h определен класс Timer, в котором реализован простой секундомер, для замеров времени работы алгоритмов.

4.2 Описание структур данных

- list: Вектор векторов, используется для представления списка смежности графа.
- Edge: Структура, представляющая ребро графа, с полями from (начальная вершина), to (конечная вершина) и weight (вес ребра).

Методы:

- Edge: Структура, представляющая ребро графа, с полями from (начальная вершина), to (конечная вершина) и weight (вес ребра).
- Graph::addEdge(int from, int to, int weight): Добавляет ребро в граф.
- Graph::printGraph(): Выводит граф в консоль.
- Graph::PrimMST(): Реализация алгоритма Прима для поиска минимального остовного дерева.

- Graph::Kruskal(): Реализация алгоритма Краскала для поиска минимального остовного дерева.
- Graph::generateRandGraph(int _size, int minWeight, int maxWeight): Генерирует случайный граф.

5 Описание алгоритмов

5.1 Алгоритм Прима

На вход алгоритма подаётся связный неориентированный граф. Для каждого ребра задаётся его стоимость.

Сначала берётся произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшей стоимостью. Найденное ребро и соединяемые им две вершины образуют дерево. Затем, рассматриваются рёбра графа, один конец которых — уже принадлежащая дереву вершина, а другой — нет; из этих рёбер выбирается ребро наименьшей стоимости. Выбираемое на каждом шаге ребро присоединяется к дереву. Рост дерева происходит до тех пор, пока не будут исчерпаны все вершины исходного графа.

Результатом работы алгоритма является остовное дерево минимальной стоимости.

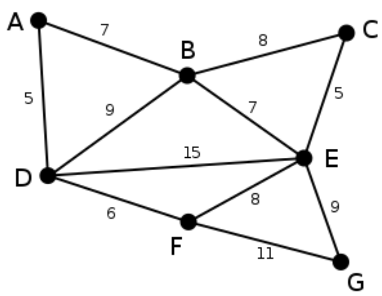
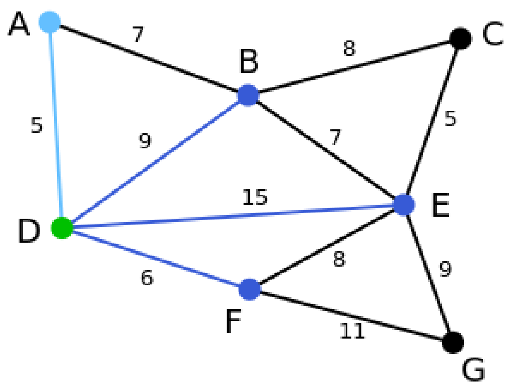
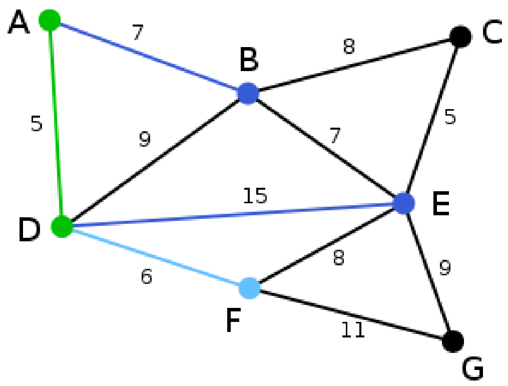


Рис. 1: Неориентированный взвешенный граф

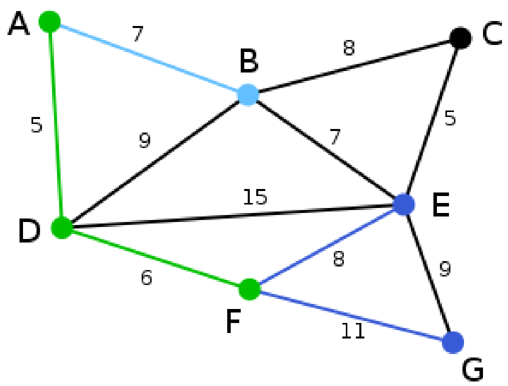
В качестве начальной произвольно выбирается вершина D. Каждая из вершин A, B, E и F соединена с D единственным ребром. Вершина A — ближайшая к D, и выбирается как вторая вершина вместе с ребром AD.



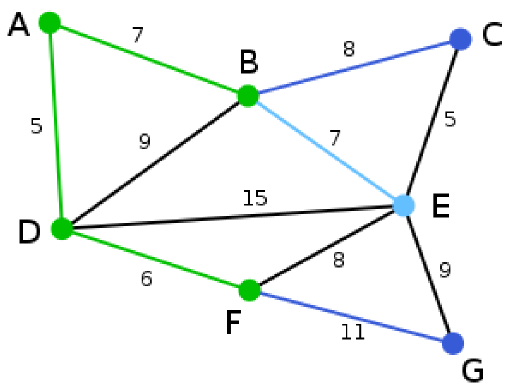
Следующая вершина — ближайшая к любой из выбранных вершин D или A. B удалена от D на 9 от A — на 7. Расстояние до E равно 15, а до F — 6. F является ближайшей вершиной, поэтому она включается в дерево F вместе с ребром DF.



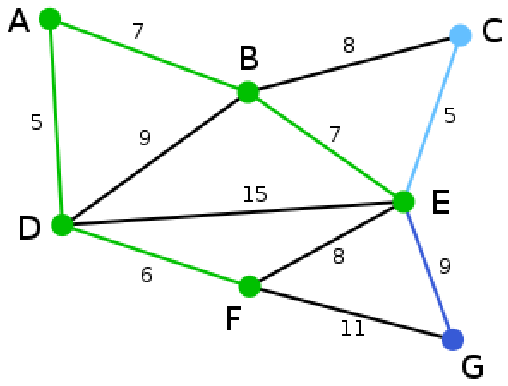
Аналогичным образом выбирается вершина B, удаленная от A на 7.



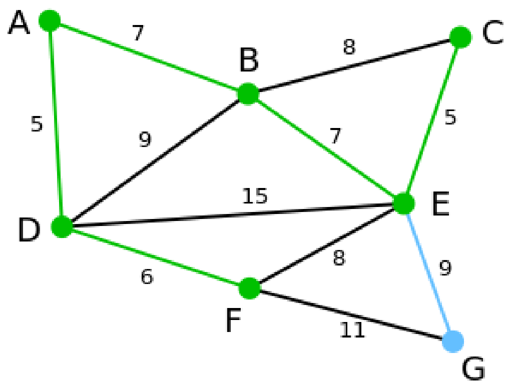
В этом случае есть возможность выбрать либо C, либо E, либо G. C удалена от B на 8, E удалена от B на 7, а G удалена от F на 11. E — ближайшая вершина, поэтому выбирается E и ребро BE.



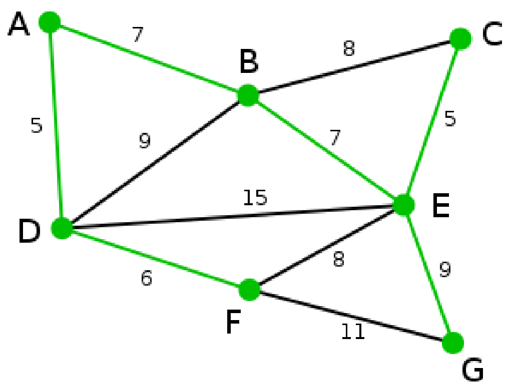
Здесь доступны только вершины C и G. Расстояние от E до C равно 5, а до G — 9. Выбирается вершина C и ребро EC.



Единственная оставшаяся вершина — G. Расстояние от F до неё равно 11, от E — 9. E ближе, поэтому выбирается вершина G и ребро EG.



Выбраны все вершины, минимальное остовное дерево построено (выделено зелёным). В этом случае его вес равен 39.



5.2 Псевдокод

Algorithm 1: PrimMST(G, w, r)

1 **for** *каждой вершины* $u \in V[G]$ **do**
2 $\quad key[u] \leftarrow \infty$;
3 $\quad \pi[u] \leftarrow \mathrm{NIL}$;
4 $key[r] \leftarrow 0$;
5 $Q \leftarrow V[G]$;
6 **while** $Q \neq \emptyset$ **do**
7 $\quad u \leftarrow \mathrm{pop}(Q)$;
8 $\quad$ **for** *каждой вершины* $v \in Adj[u]$ **do**
9 $\quad\quad$ **if** $v \in Q$ *и* $w(u,v) < key[v]$ **then**
10 $\quad\quad\quad \pi[v] \leftarrow u$;
11 $\quad\quad\quad key[v] \leftarrow w(u,v)$;

В нашей реализации, вместо приоритетной очереди будет использоваться двоичная, минимальная куча.

Производительность алгоритма Прима зависит от выбранной реализации оче- реди с приоритетами Q.В нашей реализации, вместо приоритетной очереди будет использоваться двоичная, минимальная куча minHeap. Тело цикла while выполняется |V | раз, а поскольку каждая операция pop занимает время O (lg V), об- щее время всех вызовов процедур pop составляет O (V lg V). Цикл for в строках 8–11 выполняется всего O (E) раз, поскольку сумма длин всех спис- ков смежности равна 2 |E|. Внутри цикла for проверка на принадлежность minHeap в строке 9 может быть реализована за постоянное время, если воспользоваться для каждой вершины битом, указывающим, находится ли она в minHeap, и обновлять этот бит при удалении вершины из minHeap. Присвоение в строке 11 неявно вклю- чает операцию elem_down над пирамидой. Время выполнения этой опера- ции — O (lg V). Таким образом, общее время работы алгоритма Прима составляет O (V lg V + E lg V) = O (E lg V), что асимптотически совпадает со временем ра- боты алгоритма Крускала, который будет описан позже.

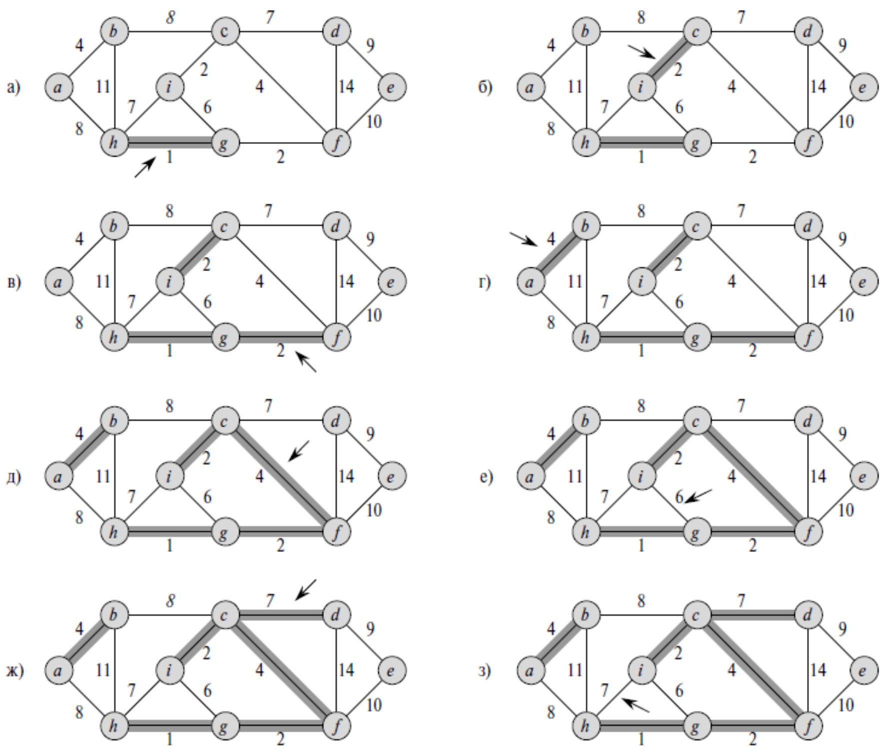
5.3 Алгоритм Краскала

Алгоритм Крускала непосредственно основан на обобщенном алгоритме по- иска минимального остовного дерева. Он находит безопасное ребро для добавления в растущий лес путем поиска ребра (u, v) с ми- нимальным весом среди всех ребер, соединяющих два дерева в лесу. Обозначим два дерева, соединяемые ребром (u, v), как C1 и C2. Поскольку (u, v) должно быть легким ребром, соединяющим C1 с некоторым другим деревом, (u, v) — безопасное для C1 ребро. Алгоритм Крускала является жадным, поскольку на каждом шаге он добавляет к лесу ребро с минимально возможным весом. Наша реализация алгоритма Крускала напоминает алгоритм для вычисления связных компонентов. Она использует структуру для представ- ления непересекающихся множеств. Каждое множество содержит вершины де- рева в текущем лесу. Операция find_root(u) возвращает представительный эле- мент множества, содержащего u. Таким образом, мы можем определить, принадлежат ли две вершины u и v одному и тому же дереву, проверяя равенство find_root(u) и find_root(u). Объединение деревьев выполняется при помощи про- цедуры union_sets.

5.4 Псевдокод

Algorithm 2: Kruskal(G, w)

1 $A \leftarrow \emptyset$;
2 **for** *каждой вершины* $v \in V[G]$ **do**
3 $\quad$ MAKE_SET(v);
4 Сортируем рёбра из E в неубывающем порядке их весов w;
5 **for** *каждого* $(u, v) \in E$ *(в порядке возрастания веса)* **do**
6 $\quad$ **if** *FIND_SET(u)* $\neq$ *FIND_SET(v)* **then**
7 $\quad\quad$ $A \leftarrow A \cup \{(u, v)\}$;
8 $\quad\quad$ UNION(u, v);
9 **return** A;



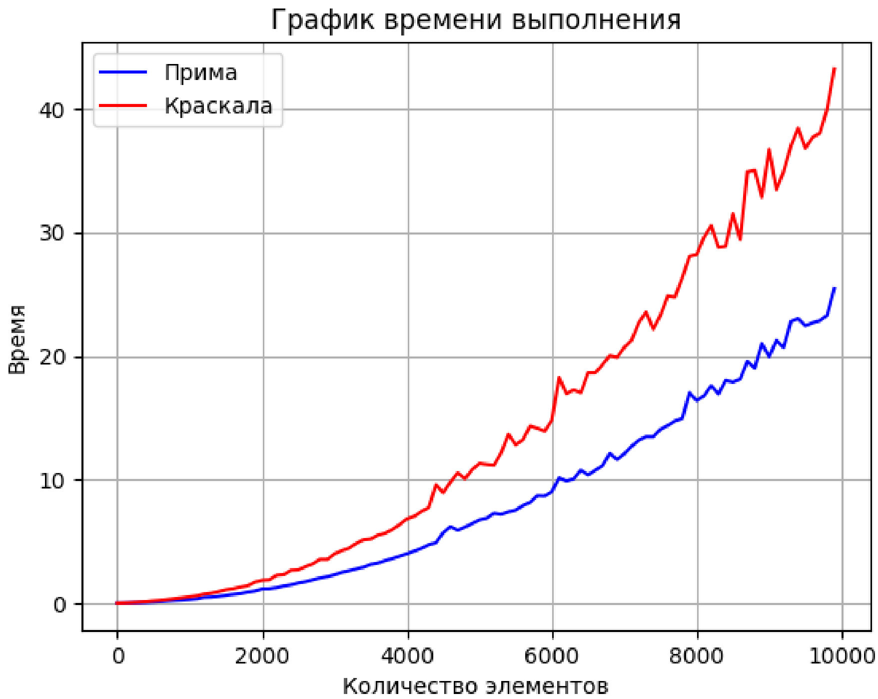
Время работы алгоритма Крускала для графа G = (V,E) зависит от реализа- ции структуры данных для непересекающихся множеств. Инициализация множества A в строке 1 занимает вермя O(1), а время, необходимое для сортировки множества в строке 4, равно $\mathcal{O}(E\log(E))$ (стоимость |V| операций MAKE_SET в цикле for в строках 2–3 мы учтем чуть позже). Цикл for в строках 5–8 выполняет $\mathcal{O}((V+E)\alpha(V))$ где $\alpha-$ очень медленно растущая функция. Поскольку мы пред- полагаем, что G — связный граф, справедливо соотношение ($|E| \geq |V| - 1$), так что операции над непересекающимися множествами требуют $\mathcal{O}(E\alpha(V))$ времени. Кроме того, поскольку $\alpha(|V|) = \mathcal{O}(\lg(V)) = \mathcal{O}(\lg(E))$, общее время работы алгоритма Крускала равно $\mathcal{O}(E\lg(E))$. Заметим, что $|E| < |V|^2$, поэтому $\lg|E| = \mathcal{O}(\lg(V))$ и время работы алгоритма Крускала можно записать как $\mathcal{O}(E\lg(V))$.

6 Эксперементы

6.1 Замеры

Проведем эксперементы на основе седующих данных:

1. Вершины от $n = 2$ до $n = 10^4 + 1$
2. Количество ребер $m = \frac{n(n-1)}{2}$
3. Веса ребер от 1 до 10^6
4. Шаг - 100 вершин

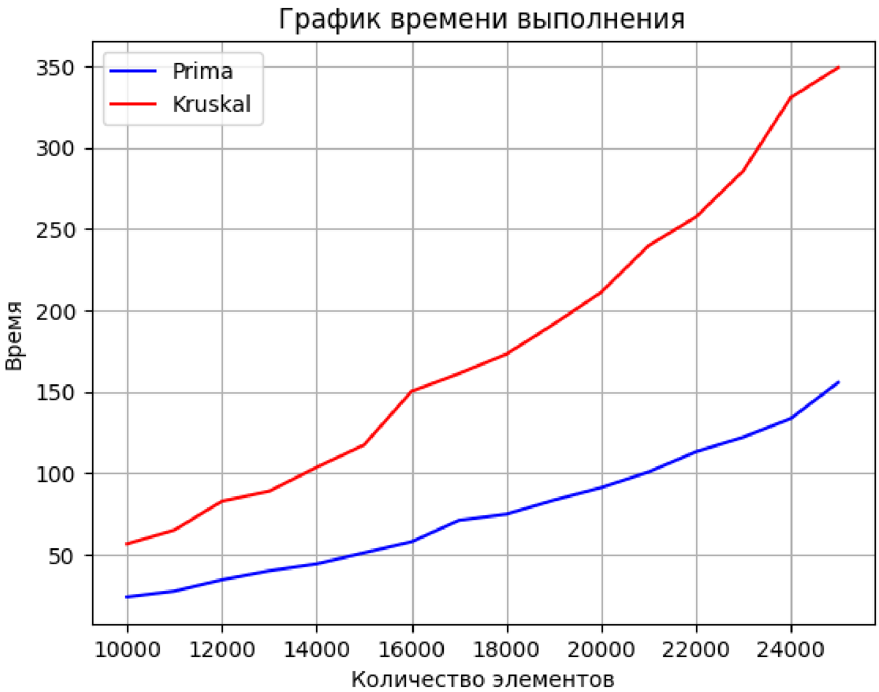


Среднее время отклонения работы двух алгоритмов составило $5.7725961c$ которое рассчитывалось по формуле:

$$mean\_time = \left| \frac{T(Prima) - T(Kruskal)}{count\_iter} \right|$$

Где $T(Prima)$ -время работы алгоритма Прима, $T(Kruskal)$ - время работы алгоритма Краскала, $count\_iter$ - количество итераций.

Попытки провести замеры на графе с количеством вершин от $10^4 + 1$ до 10^7 ресурсов компьютера и времени на исполнение хватает только до 24999 вершин с шагом в 1000.



В этих замерах, среднее время отклонения работы двух алгоритмов составило 36.0787255c

7 Результаты экспериментов

В результате экспериментов было установлено, что алгоритм Прима проявляет свою преимущественную эффективность на полных графах, особенно на графах с большим количеством вершин. Заметное ускорение работы алгоритма Прима начинается при количестве вершин, равном или превышающем 10000.

Для более общих графов, не являющихся полными, алгоритм Краскала может быть более эффективным в плане времени выполнения.

8 Вывод

Исходя из результатов экспериментов, можно сделать вывод, что выбор алгоритма для нахождения минимального остовного дерева зависит от конкретных характеристик задачи и графа. Если граф полный и содержит много вершин (более 10 000), то алгоритм Прима предпочтителен из-за своей более высокой скорости выполнения. В других случаях, алгоритм Краскала может оказаться более подходящим выбором.

Рекомендуется выбирать алгоритм нахождения минимального остовного дерева в зависимости от конкретных требований задачи и характеристик графа.

9 Литература

Список литературы

[1] Кормен, Томас Х., Лейзерсон, Чарльз И., Ривест, Рональд Л., Штайн, Клиффорд. Алгоритмы: построение и анализ, 2-е издание.

[2] Автор: Майкл Солтис. Год издания: 2019. Введение в анализ алгоритмов.

[3] Алгоритм Краскала `https://ru.wikipedia.org/wiki/`

[4] Бинарные кучи `https://habr.com/ru/articles/112222/`