ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

Вариант 6. Алгоритм Крускала.

Студент Вампилов Буда Арсаланович 411032, группа М4130.

Ссылка на исходный код (jupyter notebook).

https://github.com/kupaqu/evcalc/blob/main/lab-1-kruskal.ipynb

Описание работы.

Алгоритм состоит из следующих шагов:

1. Отсортировать список ребер в возрастающем порядке.
2. Последовательно пройтись по списку ребер добавляя их в оставное дерево.
3. Если добавленное ребро создает цикл, то не используем его.
4. Если все вершины графа содержатся в оставном дереве, то алгоритм останавливается.

Для реализации алгоритма на языке Python были использованы операторы структуры системы непересекающихся множеств, такие как:

· MAKESET

```
parent = []
rank = []
for node in range(self.V):
    parent.append(node)
    rank.append(0)
```

· FINDSET

```
def find(self, parent, i):
    if parent[i] == i:
        return i
    return self.find(parent, parent[i])
```

· UNION

```
def apply_union(self, parent, rank, x, y):
    xroot = self.find(parent, x)
    yroot = self.find(parent, y)
    if rank[xroot] < rank[yroot]:
        parent[xroot] = yroot
```

```
    elif rank[xroot] > rank[yroot]:
        parent[yroot] = xroot
    else:
        parent[yroot] = xroot
            rank[xroot] += 1
```

Сложность алгоритма.

Время работы алгоритма складывается из времени сортировки ребер (наибольшее время) и времени работы системы непересекающихся ребер.

В базовом варианте ребра сначала сортируются за время O(m^2) (с помощью пузырьковой сортировки), затем просматриваются в порядке увеличения веса за время O(m*a(m, n)), где a < 5, m – число ребер, n – число вершин. Итоговая сложность алгоритма O(n^2).

В случае если сортировать ребра за время O(m*logm), с помощью Timsort (встроен в Python), то и итоговая сложность алгоритма получится порядка O(m*logm).

Тесты и сравнение производительности алгоритмов.

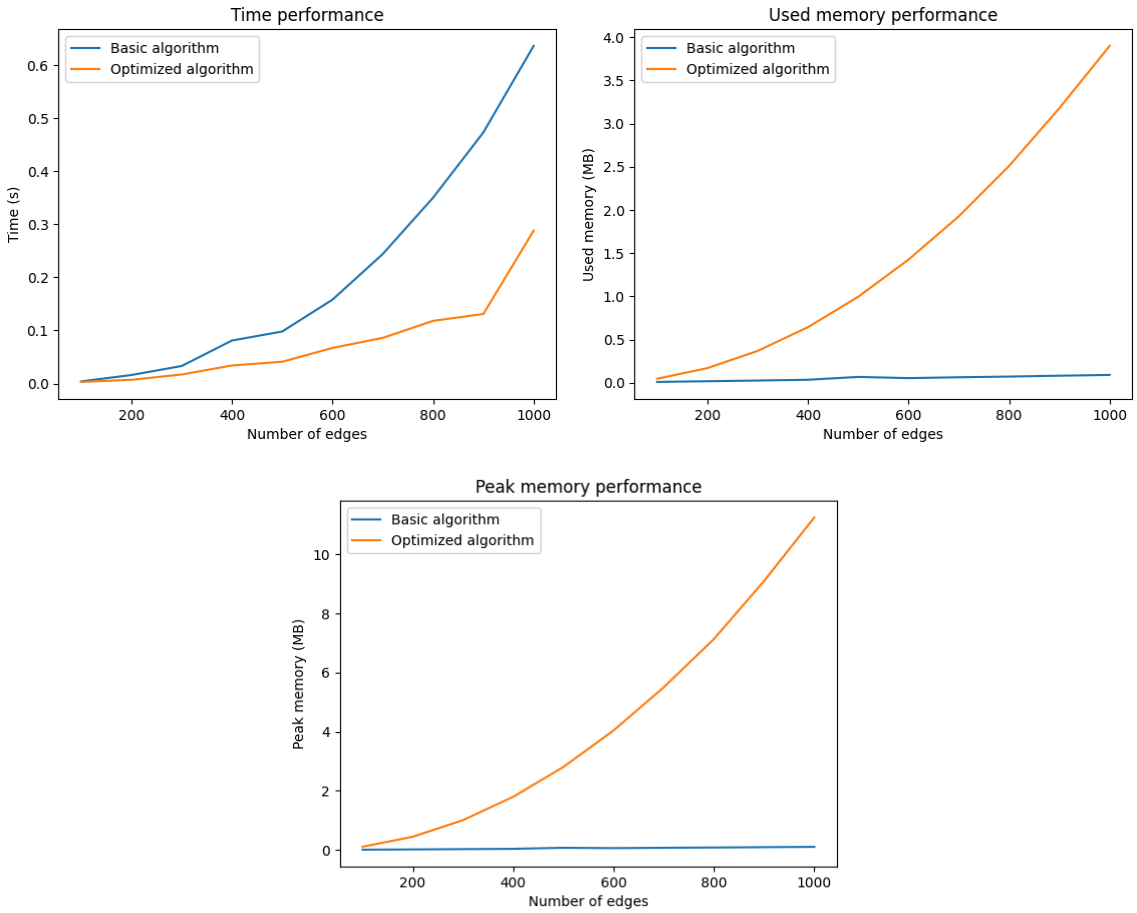
Базовый алгоритм.

Number of edges	Time (s)	Used memory (MB)	Peak memory (MB)
100	0.004	0.009	0.010
200	0.012	0.019	0.022
300	0.030	0.026	0.031
400	0.043	0.034	0.040
500	0.091	0.042	0.050
600	0.151	0.053	0.064
700	0.241	0.064	0.076
800	0.329	0.073	0.086
900	0.460	0.093	0.108
1000	0.594	0.102	0.119

Оптимизированный алгоритм.

Number of edges	Time (s)	Used memory (MB)	Peak memory (MB)
100	0.002	0.047	0.112
200	0.007	0.169	0.447
300	0.018	0.378	1.020
400	0.033	0.644	1.796
500	0.047	0.995	2.807
600	0.072	1.423	4.041
700	0.096	1.927	5.499
800	0.122	2.507	7.135
900	0.157	3.166	9.072
1000	0.193	3.922	11.241

Графики.



Базовый алгоритм квадратичной сложности работает значительно дольше оптимизированного алгоритма линейно-логарифмической сложности, это можно увидеть по тому, как возрастает время в секундах относительно

количества вершин. Оптимизированный алгоритм показывает лучшую произоводительность на большом количестве вершин.

Однако производительность памяти в базовой реализации намного лучше, т. к. в сортировке пузырьком требуется только одна временная переменная (методом свапа она не требуется вообще), что соответствует сложности O(1). Оптимизированный алгоритм (Timsort) должен работать с сложностью по памяти O(N), т. к. не предполагает хранение в памяти более двух дубликатов значения, однако на практике видим, что производительность по памяти близка к линейно-логарифической, т. е. O(n*logn).

Выводы.

Было реализовано два варианта алгоритма Крускала, отличающихся методом сортировки, также были проведены тестовые замеры производительности обоих алгоритмов и построены графики, на которых наглядно видны различия по сложности времени и памяти. Также было показано, что эксперименты по памяти отличаются от теоретических выводов для оптимизированного алгоритма.