Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет прикладної математики та інформатики
Кафедра програмування

Звіт
до лабораторної роботи №6
“Алгоритм Дейкстри”

Підготував:
студент групи ПМІ-31
Процьків Назарій

Львів 2023

Завдання

Для зваженого графа G(V,F), де V={a0, a1,…an} – множина вершин (n –велике число), а F множина ребер між вершинами, використовуючи алгоритм Дейкстри, знайти найкоротший шлях між заданою вершиною a та усіма іншими. Для різної розмірності графів та довільного вузла a порахувати час виконання програми без потоків та при заданих k потоках розпаралелення.

Теоретичні відомості

Граф — це структура, що складається з набору об’єктів, у якому деякі пари об’єктів у певному сенсі «пов’язані». Об’єкти відповідають математичним абстракціям, які називаються вершинами (також називаються вузлами або точками), а кожна з пов’язаних пар вершин називається ребром (також називається ланкою або лінією). Як правило, граф зображується у вигляді діаграми як набір точок або кіл для вершин, з’єднаних лініями або кривими для ребер. Графи є одним з об’єктів вивчення дискретної математики.

Графом G = (V, E) називають сукупність двох множин: скінченної непорожньої множини V вершин і скінченої множини E ребер, які з'єднують пари вершин. Ребра зображуються невпорядкованими парами вершин (u, v).

У графі можуть бути петлі — ребра, що починаються і закінчуються в одній вершині, а також повторювані ребра (кратні, або паралельні). Якщо в графі немає петель і кратних ребер, то такий граф називають простим. Якщо граф містить кратні ребра, то граф називають мультиграфом.

Ребра вважаються неорієнтованими в тому сенсі, що пари (u, v) та (v,u) вважаються одним і тим самим ребром.

Зваженим називають простий граф, кожному ребру e якого приписано дійсне число w(e). Це число називають вагою ребра e.

Алгоритм Дейкстри – знаходження найкоротшого шляху від заданої вершини графа до всіх інших вершин цього графа.

Швидкодія - O(E * logV). Об’єм пам’яті - O(V).

Хід роботи

Виконав цю лабораторну мовами програмування C# і Python.

На код на C# можна навіть не дивитись, бо там все добре працює, а от розпаралелити Дейкстри на Python це для мене було справжнім випробуванням. Що я тільки не пробував робити. В мене вийшло зробити цю лабораторну з 4 (четвертого) разу. Це довга історія, в одному звіті не вийде розказати…

Перейдемо до правильної реалізації Дейкстри на Python.

Створив перший тест test1() щоб перевірити роботу алгоритму на малих графах:

```python
def test1():
    size = 20
    test = Graph(size)

    sequential_result = sequential_dijkstra(test, 0)

    threads = 2
    parallel_result = parallel_dijkstra(test, 0, threads)

    print(f"Sequential Dijkstra Result: {sequential_result}")
    print(f"Parallel Dijkstra Result: {parallel_result}")
    print(f"Parallel and Sequential Dijkstra results are the same: {parallel_result == sequential_result}")
```

Цей тест проведений навіть без заміряння часу, просто для визначення того чи результати послідовного і паралельного алгоритмів збігаються. Це було зробити найважче, але, як можна бачити на скріншоті нижче, все збіглось:

```
Sequential Dijkstra Result: [0, 5, 7, 11, 5, 5, 10, 7, 8, 11, 3, 8, 5, 8, 10, 12, 9, 7, 3, 4]
Parallel Dijkstra Result: [0, 5, 7, 11, 5, 5, 10, 7, 8, 11, 3, 8, 5, 8, 10, 12, 9, 7, 3, 4]
Parallel and Sequential Dijkstra results are the same: True
```

Створив другий тест test2(), щоб виміряти роботу алгоритму на великих графах:

```python
def test2():
    size = 451
    test = Graph(size)

    start_time = time.time()
    sequential_result = sequential_dijkstra(test, 0)
    sequential_time = time.time() - start_time

    print(f"Sequential Time: {sequential_time:.3f} seconds")

    truth = []
    for thread_num in range(2, 11):
        start_time = time.time()
        parallel_result = parallel_dijkstra(test, 0, thread_num)
        end_time = time.time()

        parallel_time = end_time - start_time
        truth.append(parallel_result == sequential_result)

        acceleration = sequential_time / parallel_time
        efficiency = acceleration / thread_num
        print(f"{thread_num} threads. Time: {parallel_time:.3f}s\t"
              f"Acceleration: {acceleration:.3f}\t"
              f"Efficiency: {efficiency:.3f}")
    print(f"Parallel Dijkstra is correct: {all(truth)}")
```

В цьому тесті створив граф на 451 вершину. Виміряв час виконання послідовного алгоритму, вивів його на екран, далі в циклі для кількості потоків від 2 до 10 виміряв час виконання паралельного алгоритму, вивів прискорення та ефективність на екран. І для цієї лабораторної роботи вирішив ще для кожного потоку вивести чи збігається його результат з результатом виконання послідовного алгоритму (останній рядок коду).

Результат:

```
Sequential Time: 0.032 seconds
2 threads. Time: 1.044s Acceleration: 0.030 Efficiency: 0.015
3 threads. Time: 1.057s Acceleration: 0.030 Efficiency: 0.010
4 threads. Time: 1.309s Acceleration: 0.024 Efficiency: 0.006
5 threads. Time: 1.946s Acceleration: 0.016 Efficiency: 0.003
6 threads. Time: 2.092s Acceleration: 0.015 Efficiency: 0.003
7 threads. Time: 2.796s Acceleration: 0.011 Efficiency: 0.002
8 threads. Time: 3.031s Acceleration: 0.010 Efficiency: 0.001
9 threads. Time: 3.605s Acceleration: 0.009 Efficiency: 0.001
10 threads. Time: 3.953s    Acceleration: 0.008 Efficiency: 0.001
Parallel Dijkstra is correct: True
```

На жаль, для алгоритму Дейкстри на пайтоні мені не вдалось отримати адекватні значення прискорення та ефективності, зате завжди правильно виконується. Ще я писав інші варіанти виконання цієї лабораторної і там значення прискорення та ефективності виходили набагато кращими, але правильність роботи паралельного алгоритму не вражала.

Висновок: під час виконання лабораторної роботи №6 написав програму для знаходження найкоротшого шляху між заданою вершиною та всіма іншими вершинами у зваженому орієнтованому графі, використовуючи алгоритм Дейкстри (послідовний та паралельний), обчислив прискорення та ефективність для різної кількості потоків та навчився аналізувати ці дані.