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

Звіт
до лабораторної роботи №5
“Алгоритм Флойда”

Підготував:
студент групи ПМІ-31
Процьків Назарій

Львів 2023

Завдання

Для орієнтованого зваженого графа G(V,F), де V={a0, a1,…an} – множина вершин (n –велике число), а F множина орієнтованих ребер (шляхів) між вершинами, використовуючи алгоритм Флойда, знайти найкоротший шлях між заданими вузлами a та в. Для різної розмірності графів та довільних вузлів a та в 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 Флойда призначений для знаходження найкоротшого шляху між усіма парами вершин у заданому зваженому орієнтованому графі. Цей алгоритм використовує підхід динамічного програмування для пошуку найкоротшого шляху.

Найкраща, найгірша та середня швидкодія O(V^3).

Об’єм пам’яті O(V^2).

Хід роботи

Виконав цю лабораторну мовою програмування Python.

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

```python
def test1():
    dimension = 7
    graph = Graph(dimension)
    graph.fill_graph(25)

    print("Input graph:")
    print(graph)

    start = time.time()
    shortest_paths_seq = graph.floyd_algorithm()
    end = time.time()
    print(f"Sequential: {(end - start):.5f}s")
    graph.set_threads_number(4)

    print("Single thread shortest paths:")
    for i in shortest_paths_seq:
        print(i)
```

```python
start = time.time()
shortest_paths_par = graph.floyd_algorithm_parallel()
end = time.time()
print(f"Parallel: {(end - start):.5f}s")


print("Multithread shortest paths:")
for i in shortest_paths_par:
    print(i)
print(f"Single and multi thread answer matrices are equal: "
      f"{equal_matrices(shortest_paths_seq, shortest_paths_par)}")
```

Тут створив граф з 7-ма вершинами і 25-ма ребрами, вивів його на екран, заміряв час виконання послідовного алгоритму, вивів час і граф на екран, побачив, що алгоритм працює правильно і заміряв час виконання паралельного алгоритму, вивів час і граф на екран. Побачив що результуючі графи після і паралельного алгоритмів збігаються.

Результат:

```
Input graph:
[[ 0. inf inf 60. 96.  4. 17.]
 [inf  0. inf 46. inf inf inf]
 [inf inf  0. inf inf inf 32.]
 [inf inf inf  0. 79. 43. 46.]
 [inf 40.  8. 36.  0. inf 38.]
 [82. 86. inf inf inf  0. 79.]
 [76. inf 42. inf 91. 69.  0.]]
Sequential: 0.00000s
Single thread shortest paths:
[ 0. 90. 59. 60. 96.  4. 17.]
[168.   0. 133.  46. 125.  89.  92.]
[108. 163.   0. 159. 123. 101.  32.]
[122. 119.  87.   0.  79.  43.  46.]
[114.  40.   8.  36.   0.  79.  38.]
[ 82.  86. 121. 132. 170.   0.  79.]
[ 76. 131.  42. 127.  91.  69.   0.]
Parallel: 0.01496s
Multithread shortest paths:
[ 0. 90. 59. 60. 96.  4. 17.]
[168.   0. 133.  46. 125.  89.  92.]
[108. 163.   0. 159. 123. 101.  32.]
[122. 119.  87.   0.  79.  43.  46.]
[114.  40.   8.  36.   0.  79.  38.]
[ 82.  86. 121. 132. 170.   0.  79.]
[ 76. 131.  42. 127.  91.  69.   0.]
Single and multi thread answer matrices are equal: True
```

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

```python
def test2():
    dimension = 200
    graph = Graph(dimension)
    edges = 250
    graph.fill_graph(edges)

    start_time = time.time()
    shortest_paths_seq = graph.floyd_algorithm()
    end_time = time.time()
    single_thread_duration = (end_time - start_time)
    print(f"Sequential: {single_thread_duration:.5f}s")
    print(f"Dimension: {dimension}")
    print(f"Edges: {edges}")

    tests_num = 3
    threads_num = list(range(2, 11))  # + list(range(20, 51, 10))
    comparison = []
```

```python
    for i in threads_num:
        graph.set_threads_number(i)
        current_time = []
        for _ in range(tests_num):
            start_ = time.time()
            shortest_paths_par = graph.floyd_algorithm_parallel_2()
            end_ = time.time()

            current_time.append(end_ - start_)
            comparison.append(equal_matrices(shortest_paths_seq, shortest_paths_par))

        current_time = sum(current_time) / tests_num
        acceleration = single_thread_duration / current_time
        efficiency = acceleration / i

        print(f"Threads: {i} \t"
              f"Time: {current_time:.4f}s \t"
              f"Acceleration: {acceleration:.4f} \t"
              f"Efficiency: {efficiency:.4f}\t"
              f"Correct: {comparison[i]}")
```

В цьому тесті створив граф на 200 вершин і заповнив його 250 ребрами. Заміряв час виконання послідовного алгоритму, вивів його на екран, далі в циклі для кількості потоків від 2 до 10 заміряв час виконання паралельного алгоритму, для кожної кількості потоків провів по 3 експерименти і взяв з них

середнє арифметичне та вивів на екран прискорення та ефективність. І для цієї лабораторної роботи вирішив ще для кожного потоку вивести чи збігається його результат з результатом виконання послідовного алгоритму (Correct).

Результат:

```
Sequential: 1.52141s
Dimension: 200
Edges: 250
Threads: 2  Time: 1.3894s   Acceleration: 1.0950    Efficiency: 0.5475  Correct: True
Threads: 3  Time: 1.4213s   Acceleration: 1.0705    Efficiency: 0.3568  Correct: True
Threads: 4  Time: 1.4600s   Acceleration: 1.0420    Efficiency: 0.2605  Correct: True
Threads: 5  Time: 1.5022s   Acceleration: 1.0128    Efficiency: 0.2026  Correct: True
Threads: 6  Time: 1.5707s   Acceleration: 0.9686    Efficiency: 0.1614  Correct: True
Threads: 7  Time: 1.6795s   Acceleration: 0.9059    Efficiency: 0.1294  Correct: True
Threads: 8  Time: 1.5505s   Acceleration: 0.9813    Efficiency: 0.1227  Correct: True
Threads: 9  Time: 1.6195s   Acceleration: 0.9394    Efficiency: 0.1044  Correct: True
Threads: 10     Time: 1.6442s   Acceleration: 0.9253    Efficiency: 0.0925  Correct: True
```

Висновок: під час виконання лабораторної роботи №5 написав програму для знаходження найкоротшого шляху між усіма парами вершин у зваженому орієнтованому графі, використовуючи алгоритм Флойда (послідовний та паралельний), обчислив прискорення та ефективність для різної кількості потоків та навчився аналізувати ці дані.