Program Structures and Algorithms Spring 2024

Name : Shaofan Wei

NUID: 002815198

GITHUB LINK:https://github.com/Arthurccone123/INFO6205

Task:

This Task requires us to solve the 3-SUM problem by using the Quadrithmic, Quadratic, and quadraticWithCalipers methods in the sample code stored in the repository, and using a spreadsheet, Shows that we make at least five timed observations of N values for each algorithm using the multiplication method. And an explanation of why the quadratic method work.

Relationship Conclusion:

The quadratic method is suitable for 3-SUM because it combines sorting and double pointer technology, effectively utilizes the sorting nature of arrays, and significantly reduces the number of necessary comparisons. In the case of large data sets, this algorithm can skip many combinations that do not meet the summation conditions, and avoid calculating the same triples multiple times.

Evidence to support that conclusion:

Dataset Size (N)	Quadrithmic Algorithm Time (sec)	Quadratic Algorithm Time (sec)	Quadratic with Calipers Time (sec)	Cubic Algorithm Time (sec)
250	0.02878	0.03335	0.10014	0.10454
500	0.05408	0.02319	0.41792	0.39392
1000	0.09653	0.04235	1.20327	1.21911
2000	0.25051	0.06957	5.08340	5.04197
4000	0.69018	0.20508	22.79352	22.95829

Here is the code:

ThreeSumBenchmark

```
package edu.neu.coe.info6205.threesum;

import edu.neu.coe.info6205.util.Benchmark_Timer;
import edu.neu.coe.info6205.util.TimeLogger;
import edu.neu.coe.info6205.util.Utilities;
```

```
import java.util.function.Consumer;
import java.util.function.Supplier;
import java.util.function.UnaryOperator;

public class ThreeSumBenchmark {
    public ThreeSumBenchmark(int runs, int n, int m) {
        this.runs = runs;
        this.supplier = new Source(n, m).intsSupplier(10);
        this.n = n;
    }

    public void runBenchmarks() {
        System.out.println("ThreeSumBenchmark: N=" + n);
        benchmarkThreeSum("ThreeSumQuadratic", (xs) -> new
ThreeSumQuadratic(xs).getTriples(), n, timeLoggersQuadratic);
        benchmarkThreeSum("ThreeSumQuadrithmic", (xs) -> new
ThreeSumQuadrithmic(xs).getTriples(), n, timeLoggersQuadrithmic);
        benchmarkThreeSum("ThreeSumCubic", (xs) -> new
ThreeSumCubic(xs).getTriples(), n, timeLoggersCubic);
        benchmarkThreeSum("ThreeSumQuadraticWithCalipers", (xs) -> new
ThreeSumCubic(xs).getTriples(), n, timeLoggersCalipers);
    }

    public static void main(String[] args) {
        new ThreeSumBenchmark(100, 250, 250).runBenchmarks();
        new ThreeSumBenchmark(50, 500, 500).runBenchmarks();
        new ThreeSumBenchmark(20, 1000, 1000).runBenchmarks();
        new ThreeSumBenchmark(10, 2000, 2000).runBenchmarks();
        new ThreeSumBenchmark(5, 4000, 4000).runBenchmarks();
        new ThreeSumBenchmark(3, 8000, 8000).runBenchmarks();
        new ThreeSumBenchmark(2, 16000, 16000).runBenchmarks();
    }

    private void benchmarkThreeSum(final String description, final
Consumer<int[]> function, int n, final TimeLogger[] timeLoggers) {
        if (description.equals("ThreeSumCubic") && n > 4000) return;

        Benchmark_Timer<int[]> timer = new Benchmark_Timer<>(
                description,
                null,
                function,
                null
        );

        int[] array = supplier.get();

        double time = timer.runFromSupplier(() -> array, runs);
        for (TimeLogger timeLogger : timeLoggers) {
            timeLogger.log(time, n);
        }
    }


    private final static TimeLogger[] timeLoggersCubic = {
            new TimeLogger("Raw time per run (mSec): ", (time, n) -> time),
```

```
            new TimeLogger("Normalized time per run (n^3): ", (time, n) -> time /
n / n / n * 1e6)
    };
    private final static TimeLogger[] timeLoggersQuadrithmic = {
            new TimeLogger("Raw time per run (mSec): ", (time, n) -> time),
            new TimeLogger("Normalized time per run (n^2 log n): ", (time, n) ->
time / n / n / Utilities.lg(n) * 1e6)
    };
    private final static TimeLogger[] timeLoggersQuadratic = {
            new TimeLogger("Raw time per run (mSec): ", (time, n) -> time),
            new TimeLogger("Normalized time per run (n^2): ", (time, n) -> time /
n / n * 1e6)
    };
    private final static TimeLogger[] timeLoggersCalipers = {
            new TimeLogger("Raw time per run (mSec): ", (time, n) -> time),
            new TimeLogger("Normalized time per run (n^2): ", (time, n) -> time /
n / n * 1e6)
    };

    private final int runs;
    private final Supplier<int[]> supplier;
    private final int n;
}
```

ThreeSumQuadratic

```
 public List<Triple> getTriples(int j) {
        List<Triple> triples = new ArrayList<>();
        int left = 0;
        int right = length - 1;
        while (left < j && right > j) {
            int sum = a[left] + a[j] + a[right];
            if (sum == 0) {
                triples.add(new Triple(a[left], a[j], a[right]));
                left++;
                right--;
                while (left < j && a[left] == a[left - 1]) left++;
                while (right > j && a[right] == a[right + 1]) right--;
            } else if (sum < 0) {
                left++;
            } else {
                right--;
            }
        }
        return triples;
    }
```

ThreeSumQuadraticWithCalipers

```
```

```
public static List<Triple> calipers(int[] a, int i, Function<Triple, Integer>
function) {
        List<Triple> triples = new ArrayList<>();
        int left = i + 1;
        int right = a.length - 1;
        while (left < right) {
            Triple triple = new Triple(a[i], a[left], a[right]);
            if (function.apply(triple) == 0) {
                triples.add(triple);
                left++;
                right--;
                while (left < right && a[left] == a[left - 1]) left++;
                while (left < right && a[right] == a[right + 1]) right--;
            } else if (function.apply(triple) < 0) {
                left++;
            } else {
                right--;
            }
        }
        return triples;
    }

    private final int[] a;
    private final int length;
}
```

Unit Test Screenshots: