Algorithms and Data Structures Lab 06 Preparation

Authors: Karim Haidar
Charithma Perera

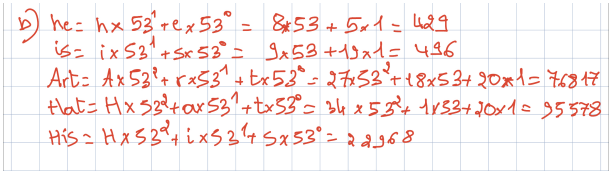
June 13, 2023

Contents

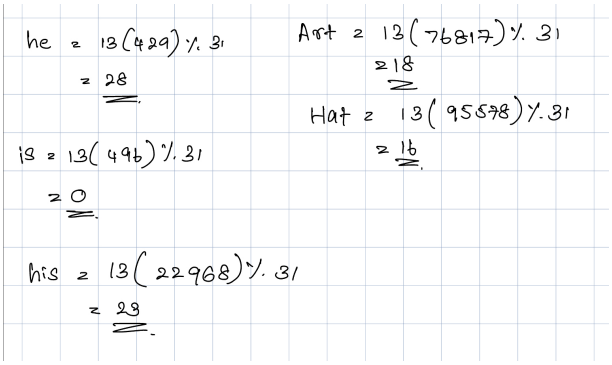
1 Problem 16 (Common Preparation)

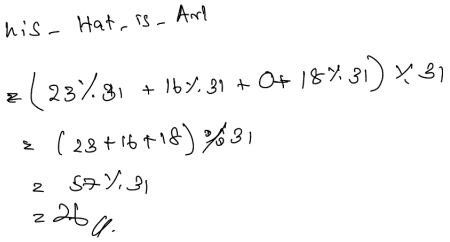
1.1 String hash calculation exercise:

Calculate the integer value of x for the following strings:



hash function h(x) for the following Strings : "he", "is", "Art", "Hat" and his_Hat_is_Art".





1.1.1 d

```
StringCode.main({ });
he -> 429
429 -> he
is -> 496
496 -> is
Art -> 76817
76817 -> Art
Hat -> 95579
95579 -> Hat
```

```
HashCoding.main({ });
he -> 28
is -> 0
Art -> 18
Hat -> 16
his_Hat_is_Art -> 1
```

1.2 Lambdas and Streams

1.2.1 a

Functional Interface	SAM	Example Lambda Expressions
Predicate<T>	boolean test(T t)	s -> s.length() > 3 [1] (Integer n) -> n % 2 == 0 (Double d) -> d > 0
Consumer<T>	void accept(T t)	(String s) -> System.out.println(s) (Integer n) -> System.out.println(n * 2) (Double d) -> System.out.println(Math.sqrt(d))
Function<T, R>	R apply(T t)	s -> Integer.parseInt(s)[1] (Integer n) -> n * 2 (Double d) -> Math.sqrt(d)
Supplier<T>	T get()	() -> "Hello, world!" () -> 42 () -> Math.random()
UnaryOperator<T>	T apply(T t)	(String s) -> s.toUpperCase() (Integer n) -> n * n (Double d) -> Math.sin(d)

1.2.2 b

Functional Interface	SAM	Lambda Expressions	Stream Methods
Predicate<T>	boolean test(T t)	s -> s.length() > 3 (Integer n) -> n % 2 == 0 (Double d) -> d > 0	filter(Predicate<T> predicate) anyMatch(Predicate<T> predicate) allMatch(Predicate<T> predicate) noneMatch(Predicate<T> predicate)
Consumer<T>	void accept(T t)	(String s) -> System.out.println(s) (Integer n) -> System.out.println(n * 2) (Double d) -> System.out.println(Math.sqrt(d))	forEach(Consumer<T> action) peek(Consumer<T> action)
Function<T, R>	R apply(T t)	s -> Integer.parseInt(s)[1] (Integer n) -> n * 2 (Double d) -> Math.sqrt(d)	map(Function<T, R> mapper) flatMap(Function<T, Stream<R» mapper)
Supplier<T>	T get()	() -> "Hello, world!" () -> 42 () -> Math.random()	generate(Supplier<T> s) iterate(T seed, UnaryOperator<T> f)
UnaryOperator<T>	T apply(T t)	(String s) -> s.toUpperCase() (Integer n) -> n * n (Double d) -> Math.sin(d)	map(UnaryOperator<T> operator) flatMap(UnaryOperator<T> operator)

1.2.3 c

Three ways to generate a stream of Integer objects filled with values 0 to N-1[3]

```
    int N = 10;

int sum = IntStream.range(0, N)
                   .sum();
```

In the first approach, IntStream.range(0, N) generates a stream of integers from 0 to N-1, and sum() calculates the sum of the elements in the stream. [3]

```
    int N = 10;

int sum = Stream.iterate(0, i -> i + 1)
                .limit(N)
                .reduce(0, Integer::sum);
```

In the second approach, Stream.iterate(0, i -> i + 1) generates an infinite stream starting from 0, and limit(N) restricts the stream to the first N elements. The reduce(0, Integer::sum) operation accumulates the sum of the stream elements. [3]

```
    int N = 10;

int sum = Stream.generate(new AtomicInteger(0)::getAndIncrement)
                .limit(N)
                .collect(Collectors.summingInt(Integer::intValue));
```

In the third approach, Stream.generate(new AtomicInteger(0)::getAndIncrement) generates an infinite stream using the AtomicInteger to increment the value. limit(N) limits the stream to N elements, and collect(Collectors.summingInt(Integer::intValue collects the elements and calculates their sum using the summingInt() collector.

2 Problem 17 (Lambdas and streams, Member A)

Printed list of persons in the terminal window can be seen in the below figure.

```
PersonList personLi1 = new PersonList();
personLi1.init();
personLi1.printList();
Sharif Almasri          MALE    25
Ritika Bansal           FEMALE  30
Shayan Khan             MALE    40
Sandani Perera          FEMALE  35
Karim Haidar            MALE    28
Esther Ojinji           FEMALE  32
Sharif Almasri          MALE    25
Ritika Bansal           FEMALE  30
Shayan Khan             MALE    40
Sandani Perera          FEMALE  35
Karim Haidar            MALE    28
Esther Ojinji           FEMALE  32
```

Implemented a method to getAverage of the Lisi of people.

```
personLi1.getAverageOfAges()
Processing Age: 25
Processing Age: 30
Processing Age: 40
Processing Age: 35
Processing Age: 28
    returned double 31.666666666666668
Processing Age: 32
Processing Age: 25
Processing Age: 30
Processing Age: 40
Processing Age: 35
Processing Age: 28
Processing Age: 32
```

results of getPerson by condition and getAverageOfAges methods can be seen below.

```
PersonList personLi1 = new PersonList();
personLi1.init();
personLi1.getPersonByConditon()
Age: 40 Gender MALE Name: Shayan Khan
    returned Object <object reference>
Age: 34 Gender MALE Name: Swarna singh
Age: 40 Gender MALE Name: Shayan Khan
Age: 34 Gender MALE Name: Swarna singh
personLi1.getAverageOfAgeByCondition()
Age: 40 Gender MALE Name: Shayan Khan
Age: 34 Gender MALE Name: Swarna singh
Age: 40 Gender MALE Name: Shayan Khan
Age: 34 Gender MALE Name: Swarna singh
Processing Age: 40
Processing Age: 34
Processing Age: 40
Processing Age: 34
    returned double 37.0
```

3 Problem 18: (Parallel streams and performance, Member B)

A series of experiments were conducted to analyze the performance of sequential and parallel streams for different values of k ranging from 3 to 20. In each experiment, M sets of random arrays, with each array having a length of N = 2 ∧ k - 1, were generated. These arrays were then converted into trees, and the average depth of each tree was calculated. The CPU consumption of both sequential and parallel processes was measured within a time frame of 60 seconds. The experiments were carried out for the binary search tree (BST) length N = 2∧20 - 1. The results provide insights into the efficiency and resource utilization of sequential and parallel streams in handling various BST sizes.

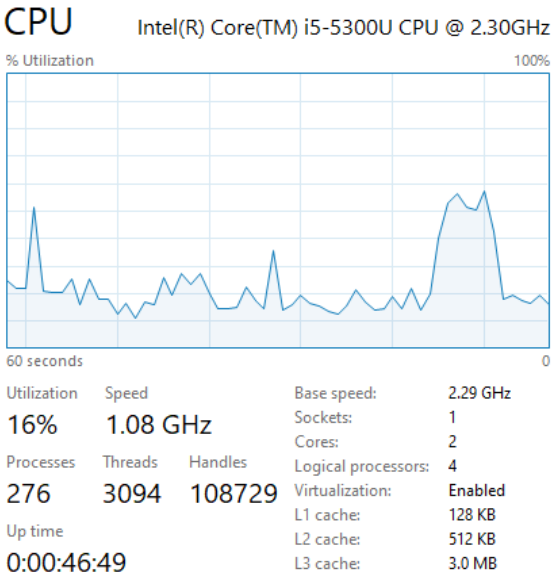


Figure 1: CPU Utilization for Sequential Process for Array up to $N = 2 \wedge 20 - 1$

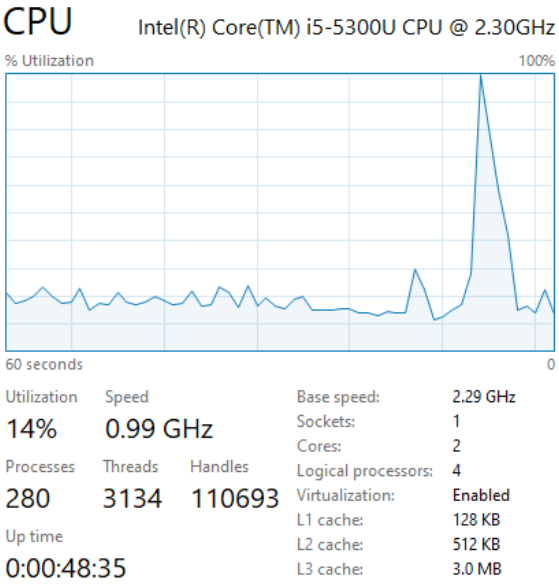


Figure 2: CPU Utilization for Parallel Process for Array up to $N = 2 \wedge 20 - 1$

Based on the depicted charts, we can infer the following conclusions:

- In the sequential process, the generation of binary search trees (BSTs) happens sequentially, resulting in a consistent and steady average process.
- On the other hand, in the parallel process, the BST generation occurs

concurrently, but the sharp decline indicates that certain arrays require more time to complete. Consequently, larger arrays tend to remain in the process until the later stages.[5]

References

[1] Sedgewick Slides. (n.d.). In R. Sedgewick and K. Wayne, Algorithms, 4th Edition. Addison-Wesley Professional.

[2] StackOfDoubles class. (n.d.). In Princeton University, Algorithms, Part I. Retrieved from https://algs4.cs.princeton.edu/13stacks/StackOfDoubles.java.html

[3] Stack overflow https://stackoverflow.com/

[4] Algorithms-Datastructures IE3 (Renz) SoSe 2023 https://e-assessment.haw-hamburg.de/course/view.php?id=391#section-0

[5] Lab report 6 - Pelumi Soyombo