COP 5536 - Advanced Data Structures

Programming Project

Fibonacci Heap Implementation

Name - Vaibhav Sahay
UFID - 54541830
Email - vaibhav.sahay@ufl.edu

Problem Statement:

The goal of this project is to implement a system to find the n most popular hashtags appeared on social media such as Facebook or Twitter. For the scope of this project, hashtags are taken as an input file. The basic idea for the implementation is to use a max priority structure to find out the most popular hashtags.

The project uses the following data structure.
1. Max Fibonacci heap: Use to keep track of the frequencies of Hashtags.
2. Hash table(Hash Map in java): Key for the hash table is hashtag and value is a pointer to the corresponding node in the Fibonacci heap.

Implementation:

The project has been implemented in Java by implementing the maximum Fibonacci heap from scratch and making use of hashmap for fast access to the required nodes. The common functionalities for a maximum Fibonacci heap are implemented which are:

1) Insert
2) RemoveMax
3) Cut
4) IncreaseKey
5) CascadeCut
6) Consolidate

The implementation also handles the input and output constraints.
If the program is run without any arguments, it will display an error message and terminate, since it expects at least one command-line argument.
If only a single command-line argument is passed, it assumes it to be the input file name and starts reading from the file if it exists and writes to the console since no output file has been specified.
If 2 command-line arguments are passed, it would read from the first file and then create an output file with the name of the second argument and write to it.

Steps to Run:

1) Unzip the file
   $unzip Sahay_Vaibhav.zip

2) Move to the directory
   $cd Sahay_Vaibhav

3) Build the project
   $make

4) Run the project
   $java hashtagcounter <inputfile> [outputfile]

Note: $ represents the command line start. The commands start from next character. [outputfile] represents that is not mandatory.
<inputfile> represents that it is mandatory and should be a valid filename in the directory.

```
[thunder:14% unzip Sahay_Vaibhav.zip
 Archive:  Sahay_Vaibhav.zip
[replace Sahay_Vaibhav/sampleoutput.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
   inflating: Sahay_Vaibhav/sampleoutput.txt
   inflating: __MACOSX/Sahay_Vaibhav/._sampleoutput.txt
   inflating: Sahay_Vaibhav/Node.java
   inflating: __MACOSX/Sahay_Vaibhav/._Node.java
   inflating: Sahay_Vaibhav/sampleInput_million.txt
   inflating: Sahay_Vaibhav/.DS_Store
   inflating: __MACOSX/Sahay_Vaibhav/._.DS_Store
   inflating: Sahay_Vaibhav/makefile
   inflating: __MACOSX/Sahay_Vaibhav/._makefile
   inflating: Sahay_Vaibhav/FibonacciHeap.java
   inflating: __MACOSX/Sahay_Vaibhav/._FibonacciHeap.java
   inflating: Sahay_Vaibhav/input.txt
   inflating: __MACOSX/Sahay_Vaibhav/._input.txt
   inflating: Sahay_Vaibhav/IFibonacciHeap.java
   inflating: __MACOSX/Sahay_Vaibhav/._IFibonacciHeap.java
   inflating: Sahay_Vaibhav/hashtagcounter.java
   inflating: __MACOSX/Sahay_Vaibhav/._hashtagcounter.java
   inflating: Sahay_Vaibhav/output.txt
   inflating: __MACOSX/Sahay_Vaibhav/._output.txt
[thunder:15% cd Saha
 Sahay_Vaibhav/     Sahay_Vaibhav.zip
[thunder:15% cd Sahay_Vaibhav
[thunder:16% make
 javac -g hashtagcounter.java
[thunder:17% java hashtagcounter input.txt output.txt
[thunder:18% java hashtagcounter input.txt
 cholelithotomy,chlorococcum,chloramine,chon,chivarras
 chloramine,chivarras,chloroprene,chloral,chlorococcum,cholelithotomy,chlorothiazide
 chloramine,chirurgy,chivarras,chloroprene,chisel,chocolate,chloral,chloroquine,chlorococcum
 choke,chokidar
 choke,chishona,chloroquine,chokidar,chloroprene,chloramphenicol,chirurgy,chlorothiazide,choleraic,chokra,chloramine,chivarras,chon,chlorophyll,choir,chirurgery,cholecystectomy,chlorura
 chlorococcum,chishona,choke,chirurgery,cholelithotomy,chitterings,chloroprene,chokra,chisel,choleraic,cholecystectomy,chirurgy,chloramphenicol,chlorophyceae,chloroquine
 chishona,cholelithotomy,chlorococcum,choke,choleraic,chloramphenicol,chivarras
 choke,chlorura,chisel,cholelithotomy,chishona,choleraic,chlorophyll,chivarras
 choke,chisel,cholelithotomy
 chlorura,cholecystectomy,chlorophyll,choleraic,cholelithotomy,chisel,choke
 chlorophyll,chlorura,choleraic,cholelithotomy,cholecystectomy,choke,chlorella,chlorococcum,chisel
[thunder:19% java hashtagcounter
 Enter the input filename
 thunder:20%
```

Fig 1: Steps to run the project.

Project Modules and Structure:

The project has the following set of files:

1) Node.java
2) FibonacciHeap.java
3) hashtagcounter.java
4) IFibonacciHeap.java

Node.java: This class represents a node in the Fibonacci heap. The data members are private and hence public accessors are provided to change the data from outside of the class.

Each node contains:

a) The frequency of hashtag
b) Hashtag
c) Parent node
d) Left sibling
e) Right sibling
f) Degree
g) isMarked
h) Child node

The generated JavaDoc with the function prototypes can be seen below:

Class Node

java.lang.Object
    com.fibonacciheap.application.Node

```
public class Node
extends java.lang.Object
```

Author:

vaibhav

Constructor Summary

Constructors

Constructor and Description
`Node(int frequency, java.lang.String hashtag)` Parameterized constructor for initializing the node

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`Node`	`getChild()` Returns the value of child.
`int`	`getDegree()` Returns the degree of the node.
`int`	`getFrequency()` returns the frequency of the hashtags.
`java.lang.String`	`getHashtag()` Returns the hashtag stored in the node.
`Node`	`getLeft()` Returns the left sibling.
`Node`	`getParent()` Returns the parent of the node
`Node`	`getRight()` Returns the right sibling.

Modifier and Type	Method and Description
Node	getChild() Returns the value of child.
int	getDegree() Returns the degree of the node.
int	getFrequency() returns the frequency of the hashtags.
java.lang.String	getHashtag() Returns the hashtag stored in the node.
Node	getLeft() Returns the left sibling.
Node	getParent() Returns the parent of the node
Node	getRight() Returns the right sibling.
boolean	isMarked() returns the value of the cut field.
void	setChild(Node child) Sets the value of child to the passed param.
void	setDegree(int degree) Sets the degree.
void	setFrequency(int frequency) Sets the frequency.
void	setHashtag(java.lang.String hashtag) sets the hashtag.
void	setLeft(Node left) Sets the left sibling.
void	setMarked(boolean isMarked) sets the value of cut field.
void	setParent(Node parent) Sets the parent in node to the passed param
void	setRight(Node right) Sets the right sibling.

IFibonacciHeap.java

It is the interface with the method prototypes.
The implementation is provided in the FibonacciHeap class which implements the interface.

FibonacciHeap.java

It implements the IFibonacciHeap interface and provides the implementation of the methods such as insert,increaseKey and removeMax.
The generated Javadoc are attached below:

Class FibonacciHeap

java.lang.Object
    com.fibonacciheap.application.FibonacciHeap

All Implemented Interfaces:

`com.fibonacciheap.application.IFibonacciHeap`

```
public class FibonacciHeap
extends java.lang.Object
implements com.fibonacciheap.application.IFibonacciHeap
```

Basic max fibonacci heap implementation

Author:

`vaibhav`

Constructor Summary

Constructors

Constructor and Description
`FibonacciHeap()`

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`void`	`addToRootList(com.fibonacciheap.application.Node node)` Adds the root of the singleton tree to the right of max and changes the existing links to achieve it.
`void`	`cascadeCut(com.fibonacciheap.application.Node node)` Recursively cuts the heap till it doesn't find a node marked F
`void`	`consolidate()` Combines the trees degreewise after removeMax
`void`	`cut(com.fibonacciheap.application.Node x, com.fibonacciheap.application.Node y)` Cuts the link between node and parent.
`int`	`getHeapSize()` Returns the current heap size.
`void`	`increaseKey(com.fibonacciheap.application.Node node, int value)` Increases the key of the node.
`com.fibonacciheap.application.Node`	`insert(int hashtagFrequency, java.lang.String hashtag)` Creates the new node from the hashtagFrequency and inserts it as a single tree in the heap, Adds the singleton tree to the right of the existing max.
`boolean`	`isEmpty()` Returns true if heap is empty, otherwise false
`void`	`link(com.fibonacciheap.application.Node y, com.fibonacciheap.application.Node x)` Links node y with x
`com.fibonacciheap.application.Node`	`removeMax()` Returns the maximum node from the heap.
`com.fibonacciheap.application.Node`	`returnMax()` Returns the current maximum node in heap.

Method Detail

insert

```
public com.fibonacciheap.application.Node insert(int hashtagFrequency,
                                                 java.lang.String hashtag)
```

Creates the new node from the hashtagFrequency and inserts it as a single tree in the heap, Adds the singleton tree to the right of the existing max. Updates the max pointer if required.

Specified by:
`insert in interface com.fibonacciheap.application.IFibonacciHeap`

Parameters:
`hashtagFrequency -`

`hashtag -`

Returns:
`node`

addToRootList

```
public void addToRootList(com.fibonacciheap.application.Node node)
```

Adds the root of the singleton tree to the right of max and changes the existing links to achieve it.

Parameters:
`node - Takes in the node of the new singleton tree`

increaseKey

```
public void increaseKey(com.fibonacciheap.application.Node node,
                        int value)
```

increaseKey

```
public void increaseKey(com.fibonacciheap.application.Node node,
                        int value)
```

Increases the key of the node.

Specified by:
`increaseKey in interface com.fibonacciheap.application.IFibonacciHeap`

Parameters:
`node - node for which increaseKey has to be executed.`

`value - value by which the existing value has to be increased`

cascadeCut

```
public void cascadeCut(com.fibonacciheap.application.Node node)
```

Recursively cuts the heap till it doesn't find a node marked F

Specified by:
`cascadeCut in interface com.fibonacciheap.application.IFibonacciHeap`

Parameters:
`node - the node being cut`

cut

```
public void cut(com.fibonacciheap.application.Node x,
                com.fibonacciheap.application.Node y)
```

Cuts the link between node and parent.

cut

```
public void cut(com.fibonacciheap.application.Node x,
                com.fibonacciheap.application.Node y)
```

Cuts the link between node and parent.

Specified by:

`cut in interface com.fibonacciheap.application.IFibonacciHeap`

Parameters:

`x - Node`

`y - Parent`

removeMax

```
public com.fibonacciheap.application.Node removeMax()
```

Returns the maximum node from the heap by removing it. The operation is followed by a degreewise merge of the trees.

Specified by:

`removeMax in interface com.fibonacciheap.application.IFibonacciHeap`

Returns:

`maximum node`

consolidate

```
public void consolidate()
```

Combines the trees degreewise after removeMax.

link

```
public void link(com.fibonacciheap.application.Node y,
                 com.fibonacciheap.application.Node x)
```

Links node y with x

Parameters:

`y - The root to be linked`

`x - The root that y is linked to.`

returnMax

```
public com.fibonacciheap.application.Node returnMax()
```

Returns the current maximum node in heap.

Returns:

getHeapSize

```
public int getHeapSize()
```

Returns the current heap size.

Returns:

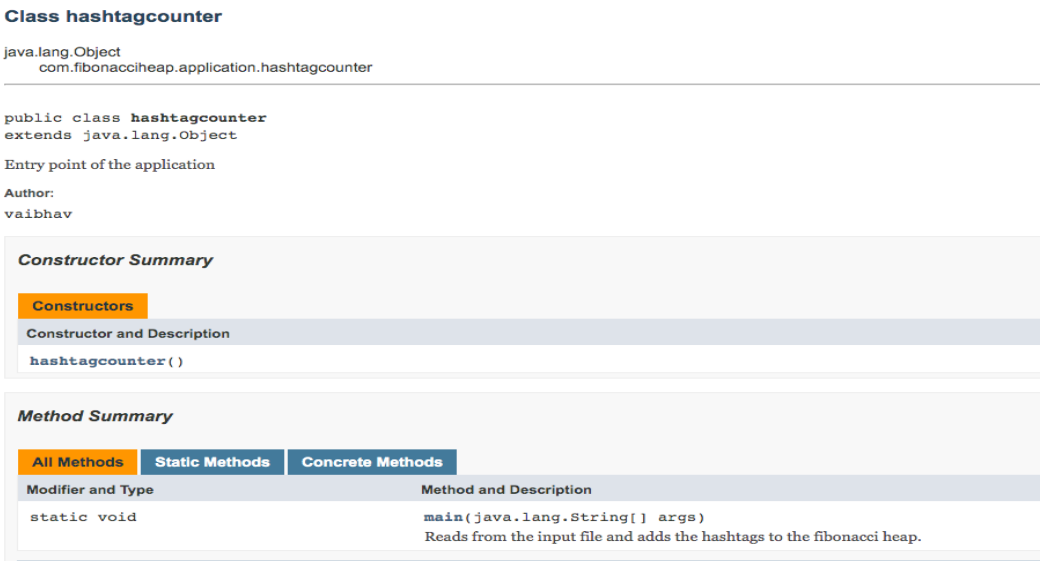
isEmpty

```
public boolean isEmpty()
```

Returns true if heap is empty, otherwise false

Hashtagcounter.java

It has the main method which gets executed when the project is run.



Class hashtagcounter

java.lang.Object
com.fibonacciheap.application.hashtagcounter

public class **hashtagcounter**
extends java.lang.Object

Entry point of the application

Author:
vaibhav

Constructor Summary

Constructors
Constructor and Description
`hashtagcounter()`

Method Summary

All Methods	Static Methods	Concrete Methods
Modifier and Type	**Method and Description**	
`static void`	`main(java.lang.String[] args)` Reads from the input file and adds the hashtags to the fibonacci heap.	

It reads in the input file, stores the hashtags in the heap and writes to the output file or the console on query.
It uses BufferedReader for reading the input file and BufferedWriter for writing to the output file.
The buffered reader object reads in the input file line by line which is in the format: #hashtag 10, where 10 is the frequency.
For each line, we split the line on space and store the result in an array.
We then check for the array size, if the size is 2, we store the hashtag details in the hashmap and the heap.
In here, we have 2 cases:

a) Hashmap contains the hashtag as the key
b) It does not contain the hashtag

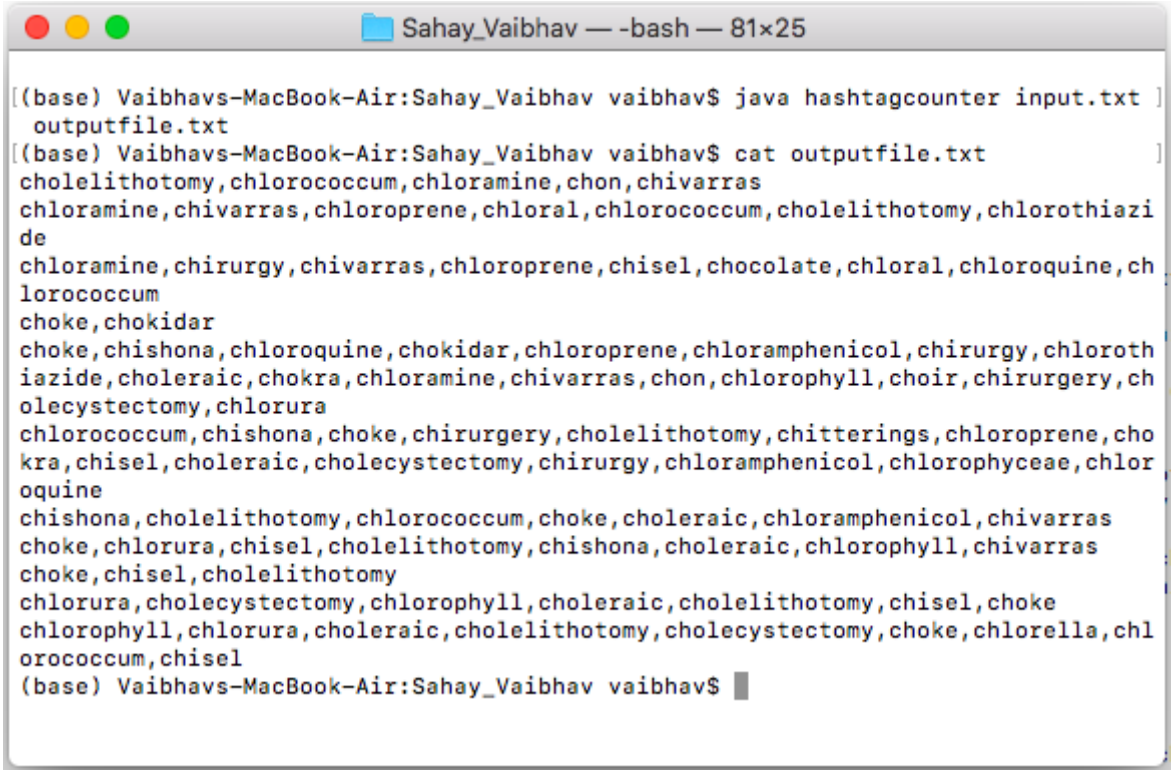
In the first case, since we already have the hashtag and the corresponding node in the map, we get the node using the key and call increaseKey on the heap, since we have found the occurrence of an already present hashtag.
In the 2nd case, it would be the first occurence of the hashtag, so we simply write it to the hashmap and call insert method on the heap.
If the array size is 1, we know that it is a query of the form n, where n is the number of hashtags with the maximum frequencies.
In this case, we iterate from 1 to n and call removeMax everytime on the heap which would return the hashtag with the maximum frequency.
It is then written to an output file, if it was provided from the command-line, else it would be printed on the console. The removed node is then added back to the heap and the map, to make sure that the frequencies are cumulatively calculated.

Sample Output

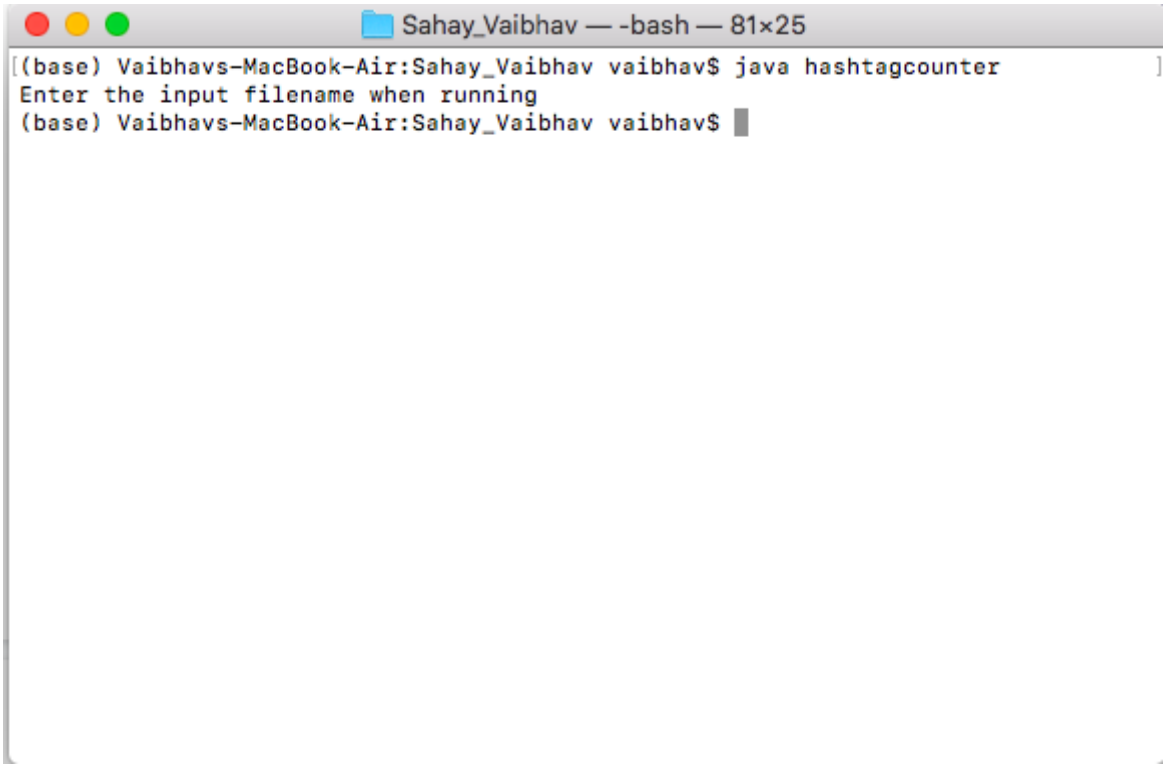
1) Without output file

```
(base) Vaibhavs-MacBook-Air:Sahay_Vaibhav vaibhav$ make
javac -g hashtagcounter.java
(base) Vaibhavs-MacBook-Air:Sahay_Vaibhav vaibhav$ java hashtagcounter input.txt
cholelithotomy,chlorococcum,chloramine,chon,chivarras
chloramine,chivarras,chloroprene,chloral,chlorococcum,cholelithotomy,chlorothiaz
ide
chloramine,chirurgy,chivarras,chloroprene,chisel,chocolate,chloral,chloroquine,c
hlorococcum
choke,chokidar
choke,chishona,chloroquine,chokidar,chloroprene,chloramphenicol,chirurgy,chlorot
hiazide,choleraic,chokra,chloramine,chivarras,chon,chlorophyll,choir,chirurgery,
cholecystectomy,chlorura
chlorococcum,chishona,choke,chirurgery,cholelithotomy,chitterings,chloroprene,ch
okra,chisel,choleraic,cholecystectomy,chirurgy,chloramphenicol,chlorophyceae,chl
oroquine
chishona,cholelithotomy,chlorococcum,choke,choleraic,chloramphenicol,chivarras
choke,chlorura,chisel,cholelithotomy,chishona,choleraic,chlorophyll,chivarras
choke,chisel,cholelithotomy
chlorura,cholecystectomy,chlorophyll,choleraic,cholelithotomy,chisel,choke
chlorophyll,chlorura,choleraic,cholelithotomy,cholecystectomy,choke,chlorella,ch
lorococcum,chisel
(base) Vaibhavs-MacBook-Air:Sahay_Vaibhav vaibhav$
```

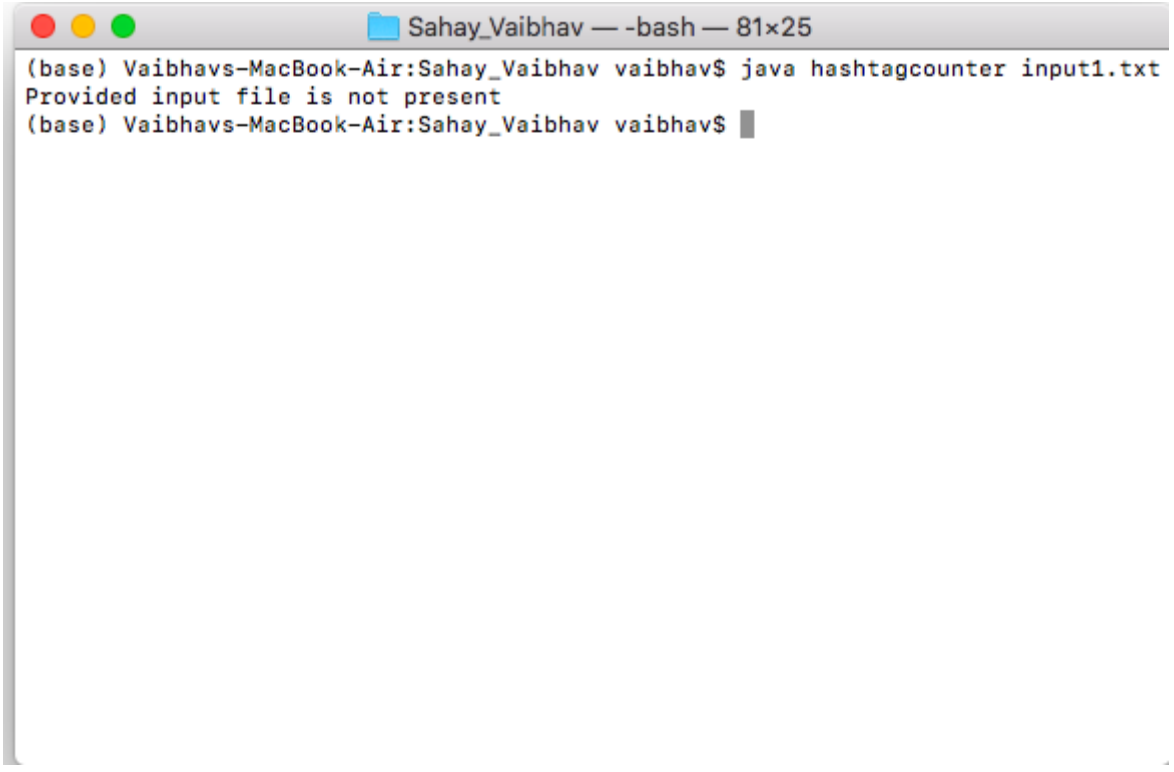
2) With output file



3) **Without command line arguments**



4) **When inputfile is not present in the Directory**



Conclusion:

The project solves the given problem statement by making use of an efficient data structure, maximum Fibonacci heap along with hashmap.
The tests have given outputs very efficiently which justifies the use of the Fibonacci heap data structure for the given problem.
It also handles all the possible scenarios and constraints mentioned in the description.

References:

[1] Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.
Chapter 19 - Fibonacci Heaps.

[2] https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/FibonacciHeaps.pdf