Advanced Data Structures(COP 5536)
Fall 2018

KEYWORD COUNTER

Programming Project Report

SHAIFIL MAKNOJIA
UFID: 7805-9466
shaifilmaknojia@ufl.edu

PROJECT DESCRIPTION

The project consists of finding the n most popular keywords in the input file. The keywords will be given from an input file. The implementation needs to be done using a max priority structure. We use a maximum Fibonacci Heap to keep the frequencies of keywords. Also, a hashtable is used. The key for the hashtable is the keywords and value is the corresponding node in the Fibonacci heap. The idea is to remove the max element from the top, store it in a list and reinsert it again.

A Fibonacci heap is a data structure for priority queue operations, consisting of a collection of heap-ordered trees. It has a better amortized running time than many other priority queue data structures including the binary heap and binomial heap. Michael L. Fredman and Robert E. Tarjan developed Fibonacci heaps in 1984 and published them in a scientific journal in 1987. They named Fibonacci heaps after the Fibonacci numbers, which are used in their running time analysis.

Fibonacci Max Heap	
Amortised Complexity	
Space	O(1)
Search	O(1)
Insert	O(log n)
Delete	O(1)
Find Max	O(1)
Delete Max	O(log n)
Increase Key	O(1)
Merge	O(1)

WORKING ENVIRONMENT

HARDWARE REQUIREMENT

Hard Disk space: 2 GB minimum

Memory: 512 MB

CPU: x86

OPERATING SYSTEM

LINUX/UNIX/MAC OS

COMPILER

Javac

COMPILING & RUNNING INSTRUCTIONS

The project has been compiled and tested on both local machine and thunder server.

To execute the program on thunder server follow below steps

You can remotely access the server using ssh
username@thunder.cise.ufl.edu

For running the KeywordCounter

1) **Extract the contents of the zip file**
2) **Type 'make' without the quotes.**
3) **Type java KeywordCounter "file path/input_file_name.txt" and add the file path and name**

   Eg. If input is in same folder → java KeywordCounter input_file.txt

   If input is in some other folder →

   java KeywordCounter "path/input_file.txt"

Note: K and C are capital in KeywordCounter

STRUCTURE OF THE PROJECT

The program consists of 3 classes.

1) **KeywordCounter -** The main class that reads the input file, parses the input and writes to output file. Each line that starts with $ is split into keyword and frequency and a node is added to the Fibonacci heap if the keyword is encountered for the first time using insert(). Otherwise the frequency is incremented using increaseKey() and cut() and cascadeCut() functions are invoked if needed. If line starts with digit we pass that digit to find the max node that many times and add it to the output file. If we encounter stop/STOP then we terminate and return

2) **Node** – This class is used to instantiate an object of Fibonacci node structure in memory.

3) **FibonacciHeap** – This class is used for the actual working and operations of Fibonacci heap

FUNCTIONS and VARIABLES

CLASS: Node

```
public Node(String keyword, int key)
```

➔ **Constructor to instatntiate an object of node class, takes keyword and frequency and sets it to instance variables**

```
    boolean childCut;
    Node childNode;
    Node leftNeighbor;
    Node parent;
    Node rightNeighbor;
    String keyword;
    int key;
    int degreeOfNode;
```

➔ **Instance variables in Node class which are required fields of Fibonacci Structure like child, degree etc**

```
public int getKey()
```

➔ **Returns the frequency/key associated with a given node**

```
public String getkeyword()
```

➔ **Returns the keyword associated with the given node**

CLASS: KeywordCounter

```
Hashtable<String, Node> trackerHashtable = new Hashtable<String, Node>();
```

➔ **Hashtable to keep track of the Node corresponding to the keyword**

```
FibonacciHeap fiboHeap = new FibonacciHeap();
```

➔ **Max Fibonnaci Heap**

```
ArrayList<Node> trackRemovedNodes = new ArrayList<Node>();
```

➔ **ArrayList to keep track of removed nodes**

```
public void parseInputFile(String InputFilePath)
```

➔ **Takes input file from main function, reads the file then perform necessary actions accordingly, If "stop/STOP" terminate the program and return, if line starts with $ either create new entry or perform increase key, if line starts with integer then call handleQuery**

```
void handleQueries(int totalQueries)
```

➔ **Takes the integer as input, performs remove max operation that many times, stores the removed nodes in above mentioned trackRemovedNodes arraylist, once all queries are processed adds back the removed nodes**

```
void updateOutputFile(String outputString) throws IOException
```

➔ **Takes the string as input and writes it to the output file**

```
void OverwriteOutputFile() throws IOException
```

➔ **This function is empty the content of output_text file, else new output would be appended to the output_text file which is not desired**

```
public static void main(String args[]) throws IOException
```

➔ **Driver program, we pass the input file name/path as the args0, then it kick starts the project**

CLASS: Fibonacci Heap

```
    int trackNumberOfNodes;
    Node maxNodePointer;
```

- ➔ **total number of nodes present**
- ➔ **pointer to the max node**

```
public void insert(Node newNode)
```

- ➔ **If it's the first element make it the max node, else combine it with other elements in the rootlist, if this new element is greater than previous max make this the new maxPointer**

```
void combineSiblings(Node maxNodePointer, Node joiningNode)
```

- ➔ **joins siblings list of joining node with that of maxNodePointer, i.e readjust neighboring nodes**
- ➔ **Takes parameter as max node and the new joining node**

```
public Node getMax()
```

- ➔ **Returns the max element from the Fibonacci heap and calls subsequent methods which are required for the implementation of Fibonacci Heap**

```
private void setParentPointerNull()
```

- ➔ **Sets parent pointer to null when we perform getMax operation and the child of max node are added to the root list**

```
private void pairwiseCombine()
```

- ➔ **Find the nodes with same degrees and join them to create a 1 higher degree node**

```
void cleanupRemovedNode(Node removedNode)
```

- ➔ **Setups the default values when we remove a node after removemax operation**

```
private void joinSameDegreeNodes(Node min, Node maxNodePointer)
```

- ➔ **This function does the job of making the node with higher root key as the parent of the other node. i.e when we combine two same degree nodes the node with smaller key at root becomes the child of node with higher key at root**

```
public void increaseKey(Node node, int key)
```

- ➔ **If the node/keyword already exists in the heap/hashtable increase its frequency**

```
private void cutNode(Node childNode, Node parentNode)
```

- ➔ **Remove child from parent and insert child in root level list of Fibonacci heap**

```
private void cascading(Node m)
```

- ➔ **Do a cascade cut upwards towards the root until a node whose childCut field is false is encoutered**

DEMO

```
thunder:39% ls -lart
total 27
drwxr-xr-x+ 17 maknojia grad   25 Nov 14 01:04 ../
-rw-------+  1 maknojia grad 6500 Nov 14 01:05 FibonacciHeap.java
-rw-------+  1 maknojia grad 5468 Nov 14 01:05 KeywordCounter.java
-rw-------+  1 maknojia grad   78 Nov 14 01:05 makefile
-rw-------+  1 maknojia grad  524 Nov 14 01:05 Node.java
-rw-------+  1 maknojia grad  262 Nov 14 01:05 test.txt
drwxrwxr-x   2 maknojia grad    7 Nov 14 01:07 ./
thunder:40% make

javac KeywordCounter.java
javac Node.java
javac FibonacciHeap.java
thunder:41% ls -lart
total 39
-rw-------+  1 maknojia grad 6500 Nov 14 01:05 FibonacciHeap.java
-rw-------+  1 maknojia grad 5468 Nov 14 01:05 KeywordCounter.java
-rw-------+  1 maknojia grad   78 Nov 14 01:05 makefile
-rw-------+  1 maknojia grad  524 Nov 14 01:05 Node.java
-rw-------+  1 maknojia grad  262 Nov 14 01:05 test.txt
drwxr-xr-x+ 18 maknojia grad   26 Nov 14 01:10 ../
-rw-------+  1 maknojia grad 3731 Nov 14 01:10 KeywordCounter.class
drwxrwxr-x   2 maknojia grad   10 Nov 14 01:10 ./
-rw-------+  1 maknojia grad  620 Nov 14 01:10 Node.class
-rw-------+  1 maknojia grad 2480 Nov 14 01:10 FibonacciHeap.class
thunder:42% java KeywordCounter test.txt
******* Completed ********
thunder:43% cat output_file.txt
facebook,youtube,amazon
facebook,youtube,gmail,ebay,amazon
thunder:44%
```

CONCLUSION

The objective of this project has been met. The program successfully creates a implementation of Fibonacci Heap. While correctly performing the removeMax and Increase Key operation on a Max Fibonacci Heap.