Project Description

The aim of the project is to find the ’n’ most popular keywords used in a search engine using a max fibonacci heap and a hash table. Once a query appears, we need to write down the most popular keywords to the output file in descending order. Output for a query should be comma separated list without any new lines. Once the output for a query is finished we have to put a new line to write the output for the next query. All outputs should be written to an output file named “output_file.txt”.

Program Structure

-Every node in the fibonacci heap is represented as a pointer object whose class has the following public data members:

Count(int): Represents the number of times the search string has been searched.
searchString(string): The searched query in the search engine.
childCut: Represents the child cut value of the node. Initialized to false.
right(node*): Pointer pointing to the right sibling of the node. Points to itself if doesn’t exist.
left(node*): Pointer pointing to the left sibling of the node. Points to itself if doesn’t exist.
child(node*): Pointer pointing to a child node. NULL if it doesn’t exist.
parent(node*): Pointer pointing to parent of the node. NULL if it doesn’t exist.
degree(unsigned long int): Represents the degree of the node. Initialized to 0.

-A constructor is used to initialise these members. It takes searchString and count values as arguments.

-A pointer object is created for every search string encountered in the input file. A hash table is maintained which consists of unique search strings and a corresponding pointer pointing to its node in the heap.

Some global variables that are used are as follows:

maxNode(node*): Pointer pointing to the max node in the heap. Initialized to NULL.
nodeCount: Keeping track of the total number of nodes in the heap.

Key Functions

void insertNode(node* nodeToBeInserted, bool existingNode)	
Description	Inserts a new/updated node into the top level circular list and updates the maxNode and sibling pointers at the root accordingly.
Parameters	nodeToBeInserted -> Node to be inserted existingNode -> Indicates whether the node to be inserted is a new node being added or if it already existed in the heap.
Return Value	No return value.

void increaseKey(node* nodeToBeUpdated, bool hasParent)	
Description	Increases the count value of the nodeToBeUpdated. If it is greater than its parent it is removed from the tree using removeNode() followed by the cascading cut operation. Also, if it is greater than the maxNode, maxNode will now point to this node.
Parameters	nodeToBeUpdated -> Node which whose count has been increased. hasParent -> Indicates whether the node to be updated is a child of a parent or not.
Return Value	No return value.

node* removeMaxNode()	
Description	Removes the maxNode from the top circular list. Updates the sibling pointers if it has siblings. If this maxNode has children, it iteratively inserts them in the the top circular list and pair-wise combines them. If this maxNode does not have any siblings or children meaning that it is the only node in the heap, it sets the maxNode pointer to NULL.
Parameters	No parameters.
Return Value	No return value.

void pairwiseCombine()	
Description	Iteratively counts the number of nodes at the root level by starting from the maxNode and going through all it's siblings as they are at the root level. Iteratively stores the degree of nodes in a vector table and combines nodes if a new node in an iteration has the same degree as an existing node in the table. Parent-child relationships are formed according to the count values of the same degree nodes. Calls combineParentChild() passing the child and parent it determined.
Parameters	No parameters.
Return Value	No return value.

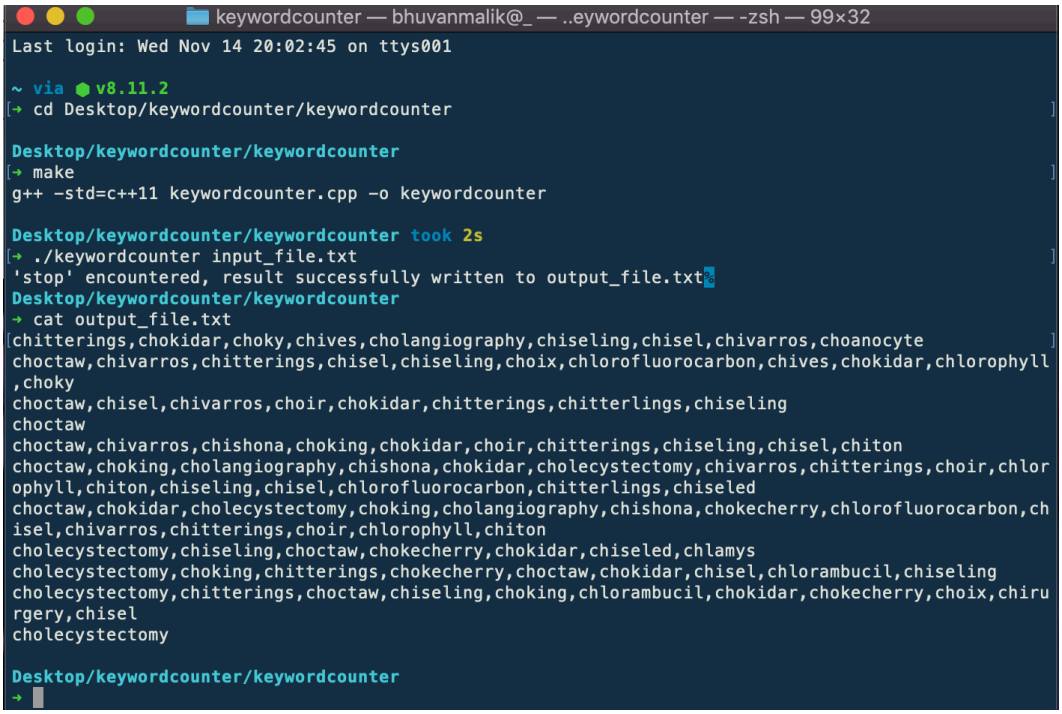
void combineParentChild(node* child, node* parent)	
Description	Called inside pairwiseCombine() where the relationship between parent and child were determined based on their count values. This function adds a child to a parent and adjusts the sibling pointers if the parent already has existing children and also updates the child pointer accordingly.
Parameters	updatedNode -> A node whose count was increased by an increase key operation or if it is being removed as its childCut was 'true' and it lost a child. parentOfUpdatedNode -> Parent of the node being removed.
Return Value	No return value.

void cascadingCut(node* updatedNode)	
Description	A recursive function which keeps removing a parent if its childCut value was ‘true’ and it lost a child either due to an increase key operation or due to cascading cut itself. The recursion halts when the stopping condition is met in which a parent is encountered whose childCut value is ‘false’ in which case the value is changed to ‘true’ or if the parent of the updatedNode is NULL i.e the node being processed after many cascading cuts is now a root node.
Parameters	updatedNode -> A node whose count was increased by an increase key operation.
Return Value	No return value.
void removeNode(node* updatedNode, node* parentOfUpdatedNode)	
Description	Removes a node whose count has been increased. Adjusts sibling pointers and updates child pointer of its parent to point to a child that is to the right of this node being removed(updatedNode). It is called inside the increaseKey() and cascadingCut() functions.
Parameters	updatedNode -> A node whose count was increased by an increase key operation or if it is being removed as its childCut was ‘true’ and it lost a child. parentOfUpdatedNode -> Parent of the node being removed.
Return Value	No return value.
int main(int argc, const char * argv[])	
Description	Program execution starts here. Opens input and output streams to read and write to a file. Handles all the different input cases, converts count/frequency of a search string from string to int. It iterates over the lines of input initialising node pointer objects and inserting them into the heap or increasing the count if a node exists in the hash table. Output formatting is also done here.
Parameters	argc -> Number of arguments passed. argv -> Array containing all the arguments.
Return Value	Any integer return value.

-Some helper functions are also used for code reusability.

Working Cases

The project has been successfully run on 1000, 10000, 1000000 and 5000000 input search strings. A sample screenshot of output for 1000 input search strings is given below:



Conclusion

Successfully implemented a max fibonacci heap on an input file to retrieve the top ’n’ searches and write them to an output file.