CMP 2003

Data Structures and Algorithms

TERM PROJECT

Introduction:

In this project we are expected to write an application that reads and counts unique words used in documents: articles, chapters, books about Applied sciences, Mathematics, Information science published from 1900 to 2021 and find top 10 frequent words in these documents.

Data Structures:

To achieve our goal, we have used different data structures to hold the data. We have created a word/count class to hold word with the number of occurrence:

```
// WordCount class holds word and count
struct WordCount {
public:
	std::string word;
	int count;

	WordCount() : word(""), count(0)
	{
	}

	WordCount(const std::string& word, int count) : word(word), count(count)
	{
	}
};
```

This is the data part which we hold word with its count.

We have created a hash table with open addressing (linked list in each cell). The definition of hash table class is as follows:

Hash Table:

```
// Hash table size
#define HASHTABLE_SIZE            32783

// Top 10 most frequently used
#define MAX_COUNT    10

// Hash table class
class HashTable {
public:
	// Add word count to the hash table
	void add(WordCount& wordCount);
	// Check if word exist in the hash table
	bool isExist(const std::string&word);
	// Print top 10 most frequently used word
	void printMostUsed();

private:
	// Create hash value of the word
	unsigned hash(const char* s);
	// Update most frequently used words array
	int updateMostUsed(WordCount& wordCount);

private:
	// Hash table holds words with counts
	LinkedList table[HASHTABLE_SIZE];
	// Holds top 10 mostly used words
	WordCount mostUsedWordCounts[MAX_COUNT];
};
```

We have 32783 cells in the hash table and in each cell we have a linked list which is holding the wordcounts which hash corresponds to that cell.

```
LinkedList table[HASHTABLE_SIZE];
```

Besides we are holding in this class the most frequently used 10 words in an array:

```
WordCount mostUsedWordCounts[MAX_COUNT];
```

The most frequently used word is at the index 0.

Methods in hash table class:

`void add(WordCount& wordCount);` : Adds a word count to the hash table and also updates the most
`bool isExist(const std::string&word);` : Returns true if the word exists in the hash table.
`void printMostUsed();` : Prints the most frequently used top 10 words to the screen
`unsigned hash(const char* s);` : Calculate the hash value the word.
`int updateMostUsed(WordCount& wordCount);` : Update the most frequently used array with the new word/count added recently.

Top 10 most frequently used array is updated according to the words in the array, we find if the word exists and update the correct position in the array, if not exist and the count is bigger than the last element in the array we put the word into the correct position

```
// Update most frequently used words array
int HashTable::updateMostUsed(WordCount& wordCount)
{
	int newIndex = -1;
	int oldIndex = -1;

	for (int i = MAX_COUNT - 1; i >= 0; --i) {
		if (wordCount.word == mostUsedWordCounts[i].word) {
			oldIndex = i;
			break;
		}
	}

	for (int i = MAX_COUNT - 1; i >= 0; --i) {
		if (wordCount.count <= mostUsedWordCounts[i].count) {
			break;
		}

		newIndex = i;
	}

	if (newIndex == -1)
		return newIndex;

	if (oldIndex == -1)
		oldIndex = MAX_COUNT - 1;

	for (int i = oldIndex - 1; i >= newIndex; --i) {
		mostUsedWordCounts[i + 1] = mostUsedWordCounts[i];
	}

	mostUsedWordCounts[newIndex] = wordCount;

	return newIndex;
}
```

Linked List:

In each cell we are holding a doubly linked list, the linked list is holding nodes and inside them the WordCount data. The definition of the doubly linked list is as follows:

```
// Linkedlist node
struct Node {
public:
	WordCount wordCount;
	struct Node* next;
	struct Node* prev;

	Node(const WordCount& wordCount) : wordCount(wordCount), next(NULL), prev(NULL)
	{
	}
};

// Linkedlist class
class LinkedList {
public:
	LinkedList();
	~LinkedList();

	// Add to end of the linked list
	void add(WordCount& wordCount);
	// Remove element from the beginning of the linked list
	void remove();
	// If a string exist in the linked list
	bool isExist(const std::string& word);
	// Number of elements in the linked list
	int count();

private:
	// Head of the linkedlist
	Node* head;
	// Tail of the linked list
	Node* tail;
	// Size of the linked list
	int size;
};
```

Node structure is holding the WordCount data with next and previous nodes.

Methods in hash table class:

`void add(WordCount& wordCount);` : Add WordCount to the end of the linked list, if the word already exists in the linked list, update the count value of the word and also update the total count of the word passed to the function.

`void remove();` : Remove a node from the beginning of the linked list.
`bool isExist(const std::string& word);` : Returns true if the word exists in the linked list.
`int count();` : Return the number of nodes in the linked list

Process:

We used two different hash tables, one for the stop words and the other for the words expects stop words.

First we read the stop words file and put all of them to the stop words hash table.

```
// Hash table holds stop words
HashTable* stopwordsTable = new HashTable();

// Read stop words file
std::ifstream stopwordsfile(stopwordsFileName);

if (stopwordsfile.is_open()) {
	std::string line;

	while (std::getline(stopwordsfile, line)) {
		line[strcspn(line.c_str(), "\r\n")] = 0;

		WordCount wordCount(line, 1);
		stopwordsTable->add(wordCount);
	}

	stopwordsfile.close();
}
```

After that we read the publications data set file and put each word to the data hash table. Before adding to the hash table first we check if the word exists in the stop word hash table or not. If the word is defined in the stop words hash table, we don't add that word to the data hash table.

Results:

Top 10 most frequently used words in the publications printed as a result is the following:

```
. :     46555
= :     42493
1 :     27966
0 :     21823
2 :     21172
, :     19169
+ :     18784
) :     14530
& :     13016
data :  13006
Elapsed time : 33.7319 seconds
```

The total time to find the result is 33.7319 seconds.