CS 2413 – Data Structures – Spring 2022 – Project 2
Due 11:59 PM, April 17, 2022
Dr Katia Papakonstantinopoulou

DESCRIPTION OF THE PROJECT

Every web site on the WorldWide Web (WWW) consists of one or more **webpages**. Each webpage (or resource in the WWW, in general) is identified by a unique **URL**. Each webpage contains the text that appears in it, as well as some links to other webpages, which we use to navigate in the WWW. These links are referred to as **Hyperlinks**; each hyperlink essentially specifies a URL and each URL is represented by a string. You view the source code of a webpage, and verify that it has the above features, if you right-click on the page and choose "View page source". In this project we deal with creating two data structures that are used for storing and processing:

1. Sets of URLs (see part A and implementation steps 1 and 2 below),
2. The structure among URLs (or hyperlinks) in the WWW (see part B and implementation step 3 below),

respectively.

A. STORING AND PROCESSING SETS OF URLs

Consider the document below that is included in a Wikipedia page:

> "Each web page is identified by a distinct Uniform Resource Locator (URL). When the user inputs a URL into their browser, that page's elements are downloaded from web servers. The browser then transforms all of the elements into an interactive visual representation on the user's device.[4]
>
> If the user clicks or taps a link to another page, the browser repeats this process to display the new page, which could be part of the current website or a different one. The browser has user interface features that indicate which page is displayed."

In this example, all the text marked in blue are hyperlinks, i.e., links to other webpages (they specify URLs that identify other webpages). For example, as we can see by viewing the HTML code of the above page, the hyperlink Uniform Resource Locator is a link to the webpage/URL https://en.wikipedia.org/wiki/URL while the hyperlink web servers is a link to the webpage/URL https://en.wikipedia.org/wiki/Web_server .

In the first part of the project you will work with strings (arrays of characters), vectors, binary search, and sorting. You will **read** from an input file some text similar to the one quoted above (i.e., containing text and URLs), you will **identify the URLs** it contains (considering each URL as string, or equivalently, an array of characters), you will **store** the URLs in an array (a field in the class *setOfURLs* that you will create), and you will count the **frequency** of each URL (i.e., how many times it appears in the input text). Then you will **filter** this array, removing from it a set of

URLs that you will be provided. The size of the input file and the number of URLs it contains are unknown, so you will need to expand the size of the array you will use whenever it is required (making it to behave like a vector). You will also create the auxiliary class *myString* to help you process strings.

After you create a *setOfURLs* data structure, containing all the URLs included in the text part of the input file, you will have to perform the following tasks on it:

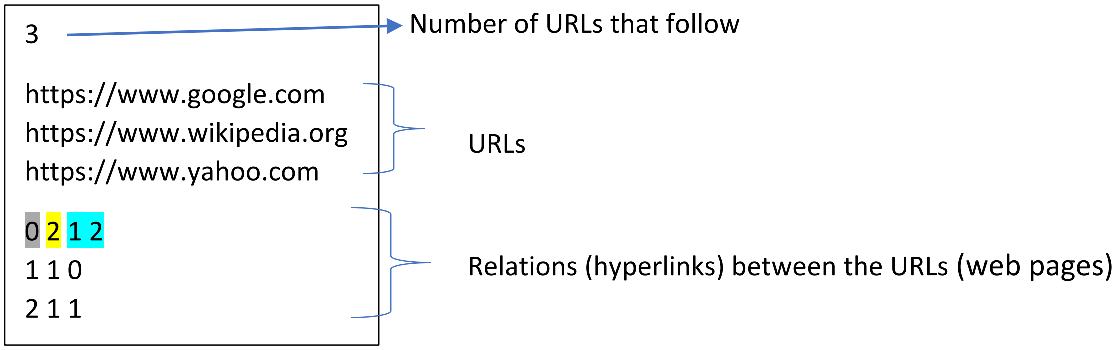
- Print the URLs it contains, sorted by alphabetical order (assuming that a<z<_)
- Print the URLs it contains, sorted by frequency (in increasing order)
- Remove a set of 'dangerous' URLs from it (this list of URLs is provided in the beginning of the input file)
- Print the remaining URLs, sorted by alphabetical order
- Print the remaining URLs, sorted by frequency
- Print the remaining URLs, sorted by alphabetical order

You can find details for the creation of the classes *myString* and *setOfURLs* below, in the implementation steps 1 and 2 respectively. We have given you part of the code for these classes (in canvas), and you are asked to **fix** any errors and write the **code** that is missing. YOU ARE NOT ALLOWED TO USE ANY LIBRARIES OTHER THAN #include <iostream>.

B. STORING THE HYPERLINK STRUCTURE

In the second part of the project you will read a set of URLs from the input file as *myString* class objects (this class has been implemented in step 1) and store them in an array of *URLLinks* objects.

This part of the input is separated by the previous one using the mark '$$$$', and it starts with the number of URLs that follow. Then we have the set of URLs. Remember that each URL corresponds to a web page. For the purposes of the project, we assume that each URL is assigned an integer number, starting from 0 (in what follows we will refer to these numbers as identifiers/IDs). Then, for each URL, in the order they appear above, we have a line that consists of the following: its ID, the number of URLs (i.e., hyperlinks to other web pages) this web page contains, and the IDs of these URLs. You can see an example of the input below.



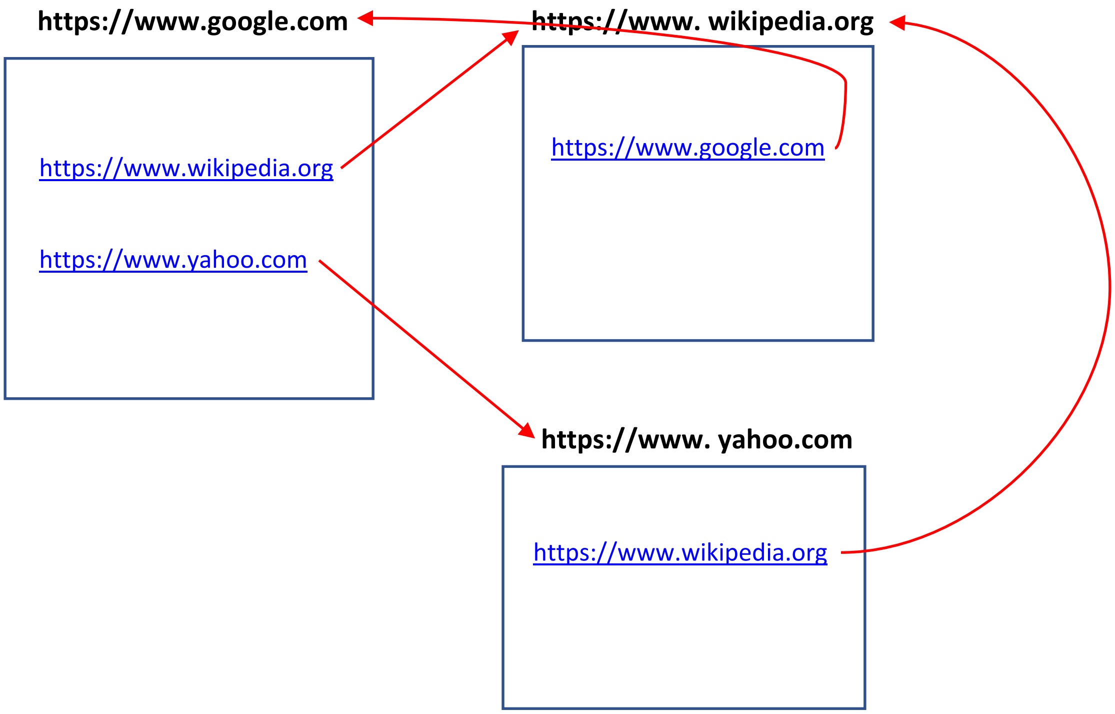
```
3

https://www.google.com
https://www.wikipedia.org
https://www.yahoo.com

0 2 1 2
1 1 0
2 1 1
```

Number of URLs that follow
URLs
Relations (hyperlinks) between the URLs (web pages)

Consider the line **0** **2** **1 2** above. Identifier **0** represents the first URL from the aforementioned list, i.e., https://www.google.com . Number **2** that follows is the number of links to other URLs contained in the resource (web page) https://www.google.com . The rest sequence of numbers is a list of the IDs of the URLs contained in https://www.google.com , i.e., the URLs to which this web page has a link. Therefore, **1 2** denotes that https://www.google.com has a link to https://www.wikipedia.org (URL **1**) and a link to https://www.yahoo.com (URL **2**). You can see a pictorial representation of this structure in the figure below.



You will read a file in the above format (i.e., containing some URLs and the information about their connections), and you will store these data in a *myLinkStructure* data structure, i.e., an array of pointers to *URLLinks* objects.

In canvas you can find a sketch that includes all classes mentioned in this project. You can use to verify your understanding.

IMPLEMENTATION STEPS

STEP 1: Implement the class *myString*

myString Class: The definition of this class is given below. **The Methods for which you need to write code are written in bold font**.

```
class myString;

class myString {
        friend ostream& operator << (ostream& s, myString& A);

protected:
        char* strArray;
        int size;
        void strCpy (char* A, char* B, int n);

public:
        myString ();
        myString(char* inpStr);
        myString (myString& B);
        int Size();
        char* getWord();

        bool operator == (myString& B);

        bool operator > (myString& B);

        bool operator < (myString& B);

        myString& operator = (myString& B);

        myString& operator = (char* B);

};
```

On canvas we will provide some code to read words from a file. We will also provide some code to empty a string, find the length of a string, etc. You can use that code. You are responsible for fixing any issues with that code. Note that the input file contains words and URLs that consist of alphanumeric characters, numbers and/or the symbols “:”, “/” and “.”. The words and URLs are **case-sensitive**.

STEP 2: Implement the class *setOfURLs*

setOfURLs Class: The definition of this class is given below. **<u>The Methods for which you need to write code are written in bold font</u>**.

You can add any other methods that you find necessary. You must initialize the arrays _URLs and _frequencies as empty. As you add new URLs, you have to increase the size of the above arrays by one. If, while reading the input file, you encounter a URL that already exists in the array _URLs, you should just increase the frequency counter for this URL by one. Also, note that you must keep the URLs in the array _URLs sorted, by decreasing order of their frequency. For this reason you will employ the binarySearchAndInsert method. Your textbook contains code that you can use (you need to modify it slightly). You may use any sorting algorithm that you have learnt in the past (Bubble Sort for example). PLEASE SEE THE COMMENTS on the methods in the class structure below, that describe their functionality.

```
class setOfURLs {

private:
        int binarySearchAndInsert (myString& u);

protected:
        myString* _URLs;
        int* _frequencies;
        int _size;

public:
        setOfURLs ();

        setOfURLs (int numOfWords);


        myString* get_Words ();

        int* get_Freq();

        int get_size();

        void setSize(int i);

        void addURL (myString& u); //insert URL u into the array _URLs - keep the array sorted
                                   //alphabetically

        void sortFreq(); //sort the URLs array based on the URLs frequency

        void sortURLs(); //sort the URLs array alphabetically

        void display(); //print URL followed by a colon, a single space and the corresponding frequency

        setOfURLs* removeURLs(myString* URLsToFilterOut, int numURLsToFilterOut);
        //remove all the URLs in the array URLsToFilterOut from _URLs array

        ~setOfURLs();

};
```

STEP 3: Implement the class *URLLinks*

URLLinks Class*:* The definition of this class is given below. You will need to write **all** the methods.

You may use additional methods if necessary. You need to initialize the size of hyperLinks pointer array depending upon the numLinks for each website. Each link has to be stored as a pointer to another URLLinks object (using addNeighbor). PLEASE SEE THE COMMENTS on the methods in the class structure below.

```
class URLLinks {

friend ostream& operator << (ostream& s, URLLinks& A);

protected:
        myString URL;
        int numLinks;
        URLLinks** hyperLinks;

public:
        URLLinks ();
        URLLinks (myString& x, int n);
        ~URLLinks ();

        int getNumLinks();
        URLLinks* getHyperLink(int i);
        myString& getURL();

        void addSite(myString& t);
        void addNeighbor(URLLinks& link);
        void setNeighbors(int nei);

};
```

There are two display methods that you have to implement in the main file:

1. Display all the hyperlinks present for a webpage (outgoing links for each website) - using the overloaded << operator.
2. Display all the webpages contained as hyperlinks of other webpages along with the number of times they occur (incoming links to each website)

Please see the sample input and output files for an indicative example of the 2 displays above.

STEP 4: ***Main* Function**

main Function: You have to use the following main function. You can modify it, but the display must follow the operations above them.

```
int main () {
    int numURLsToFilterOut;
    char* url;
    myString* urlString;
    int numPages;
    int pageNo;
    int numNeighbors;
    int neighbor;


    //read the first number from the file that contains the number of URLs that have to be filtered out
    cin >> numURLsToFilterOut;
    cout << “Number of URLs to filter out: ” << numURLsToFilterOut << endl;
    myString* URLsToFilterOutList = new myString[numURLsToFilterOut];
    //read the URLs that have to be filtered out
    for (int i=0; i < numURLsToFilterOut; i++) {
        url = getNextURL ();
        URLsToFilterOutList [i] = url; //calls the overloaded = operator on myString class
    }
    //read the given inlut and put the URLs read in the setOfURLs instance
    setOfURLs* mySetOfURLs = new setOfURLs ();
    url = getNextURL (); //first read the next URL as an array of characters
    while (url != NULL) {
        urlString = new myString (url);//create a myString object with the URL read
        (*mySetOfURLs).addURL(*urlString); //add url to mySetOfURLs
        url = getNextURL ();
    }
    // this should display each URL along with its frequency;
```

```
        // note that because you are using binary search and insert, the URLs will
        // be sorted alphabetically
        cout << endl;
        cout << “Input display:” << endl;
        (*mySetOfURLs).display ();


        (*mySetOfURLs).sortFreq ();
        cout << endl;
        cout << “mySetOfURLs – Sorted based on frequency:” << endl;
        (*mySetOfURLs).display ();

        (*mySetOfURLs).sortURLs();
        cout << endl;
        cout << “mySetOfURLs – Sorted alphabetically:” << endl;
        (*mySetOfURLs).display ();

        setOfURLs* newSetOfURLs = (*mySetOfURLs).removeURLs(URLsToFilterOutList,
numURLsToFilterOut);
        cout << endl;
        cout << “newSetOfURLs – Sorted alphabetically:” << endl;
        (*newSetOfURLs).display();

        (*newSetOfURLs).sortFreq ();
        cout << endl;
        cout << “newSetOfURLs – Sorted based on frequency:” << endl;
        (*newSetOfURLs).display ();



        //read the first number from the file that contains the number of URLs that have to be removed
        cin >> numPages;
        cout << “Number of webpages: ” << numPages << endl;
        URLLinks* myLinkStructure = new URLLinks [numPages];

        for (int i=0; i < numPages; i++) {
                //read the URLs and store them it in myLinkStructure array of URLLink objects
        }


        //store the neighbours/hyperlinks
        for (int i=0; i < numPages; i++) {

                cin >> PageNo >> numNeighbors;

                myLinkStructure[i].setNeighbors(numNeighbors);

                for (int j=0; j < numNeighbors; j++) {
                        cin >> neighbor;
                        myLinkStructure [PageNo].addNeighbor (&(myLinkStructure [neighbor]));
                }

        }

        // display all URLLink objects using the overloaded << operator

        cout << "Webpages contained as hyperlinks:" << endl;
        // display all the incoming nodes here



        //TODO: implement the destructors
        //delete [] URLsTFilterOutList;

        //delete mySetOfULRs;

        //delete newSetOfURLs;

        delete[] myLinkStructure;

        return 0;

}
```

You must ensure that your program outputs the correct results.

INSTRUCTIONS AND GUIDELINES

Programming Objectives:

1. All code must be written in **standard C++** (so that it can be compiled by a g++ compiler).
2. **You have to submit**: A file named **HW2_CS2413.cpp** that contains all the code of this project.
3. **You are provided**: A file named template2.cpp including the template your submission should follow, as well as sample input and output files of your program. These files will be available in Module "Projects" in Canvas.
4. You will create the classes described above.

5. The only header you will use is #include <iostream> and using namespace std.
6. All input will be read via **redirected input**. That is, you should not open a file inside the program.
7. Please refer to the sample input and output files given, and make sure to follow these formats exactly.
8. The **classes structure** should be as shown in the provided template file (you are responsible for filling-in the code that is missing where "___" is placed, as well as for fixing any syntax errors!). We have provided code for a few methods, and you need to make sure that they work correctly. You will also write code for other methods needed.
9. The structure of your **main** program is also provided, and you should use that in your project.
10. The submission will be done through GradeScope's autograder.

Redirected Input:

Redirected input provides you a way to send a file to the standard input of a program without typing it using the keyboard. Please check out the **Visual Studio Installation and Setup guidelines for C++.doc** provided to you on canvas. Section 3 in the document has instructions on setting up the redirected input to Visual Studio.

Remarks:

1. None of the projects is a group project. Consulting with other members of this class or seeking coding solutions from other sources including the web on programming projects is strictly not allowed and plagiarism charges will be imposed on students who do not follow this.
2. You can post questions about the projects in the respective thread in the discussion board in Canvas. Please do not include code there.
3. In case you have some specific coding question, please upload it to codePost and we will review it there.

This file will be being updated, if necessary, to reflect any further clarifications that may be given to the students by the instructor or the TAs.