

**Find And Replace**

**Final Report**

Felix Anggara - 2101693851

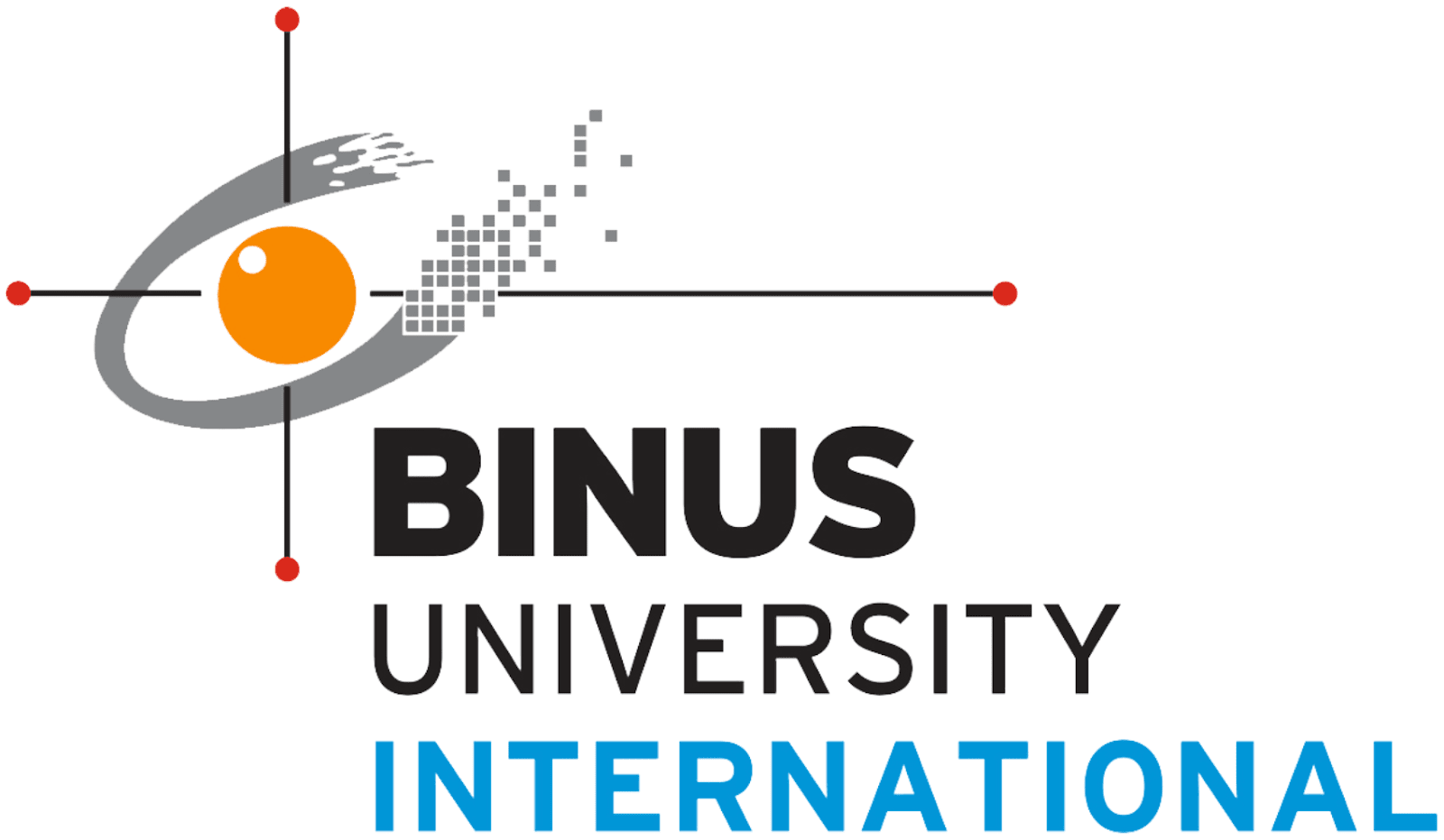


Analysis of Algorithm (Comp6340)

Binus University

School of Computer Science (International Course)

                                                                                                                   Odd Semester (2021)



**Student Information**

**Name:**                  **Student ID:**

1. Felix Anggara                        . 2101693851

**Course Name:** Analysis of Algorithm      **Course Code:** ENTR6094

**Major:** Computer Science                                                           **Lecture(s):** T.A. Budi

**Title of Assignment:** Find and Replace

**Type of Assignment:** Final Project                                             **Due Date:** 15-1-2019

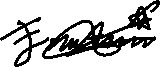
**Plagiarism/Cheating**

Binus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

**Declaration of Originality**

By signing this assignment, I understand, accept and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Correspondent Student



(Felix Anggara)

1. **Introduction**

Algorithm is fundamentally a way to solve a or multiple problems computationally. Algorithm is implemented into the computer program using what we know as programming language. Algorithm is used in almost every activities or problem solving. For example, OS with UI that allow people, even people without any knowledge of either general technology or computer science background to use computer properly. The other example is how people communicate with each other even in very long distance, the Internet.

But, when technology become more and more advance and problem that must to be solved using algorithm become more and more complex, the amount of data that exist inside every single computer become larger and larger that searching every single data using regular iteration become harder because it will take a very long time to finish. Replacing it also become difficult because replacing a pattern of characters requires a good pattern searching algorithm in order to work properly.

The purpose of this project is to solve this problem (Pattern Search Problem and String Search Problem) by using algorithm. By using a correct algorithm, Pattern searching will become faster and pattern replacement become faster. The details of algorithm that is used to solve this problem will be described in

1. **Detail of the Solver Algorithm**

Find and replace requires two important components. One, is the pattern searching mechanism to search the pattern inside the text. Second, replacing mechanism to replace the pattern with the new pattern. In order to do that, this project will use two algorithms. First, KMP algorithm. Second, modified linear probing (Using memoization).

KMP (Knuth-Morris-Pratt) algorithm, is an algorithm that is used to efficiently find a pattern inside string. It is using a lookup table instead of naïve pattern search to optimize the work. To generate the lookup table, dynamic programming is taking a huge role for it. Pattern scanning is strongly relied on dynamic programming, especially memoization.

For replace part, the project only needs memoization part to solve the problem. Other than that, it just using linear probing. Caching may make the work faster.

1. **Algorithm Analysis**

KMP algorithm has complexity with the worst case of length of the pattern plus the length of the text. In notation, it should be:

The worst case may be bigger, but it is quite small.

The following is the table of comparison between Naïve and KMP using "As he crossed toward Sthe pharmacy at the corner he involuntarily turned his head because of a burst of light that had ricocheted from his temple, and saw, with that quick smile with which we greet a rainbow or a rose, a blindingly white parallelogram of sky being unloaded from the van—a dresser with mirrors across which, as across a cinema screen, passed a flawlessly clear reflection of boughs sliding and swaying not arboreally, but with a human vacillation, produced by the nature of those who were carrying this sky, these boughs, this gliding façade." as the text and “corn” as the pattern (This comparison is using python with the unit of nanosecond):

|  |  |
| --- | --- |
| **Naïve** | **KMP** |
| 495500 | 0 |
| 463200 | 498700 |
| 498600 | 0 |
| 497100 | 496000 |
| 495000 | 0 |
| 496600 | 495500 |
| 495600 | 0 |
| 496500 | 496600 |
| 499100 | 0 |
| 989100 | 496100 |

As the data can suggests, KMP is relatively faster than Naïve in terms of performance. That means that it can scan the whole string or text faster than the regular methods.

KMP mechanism is scan the pattern first to generate the table of repetition. Then, using the table of repetition, It will scan the whole text. The repetition table is definitely reduced the workload of pattern searching due to its nature to reduce scanning redundancy. The visualization of the algorithm is available in:

<https://people.ok.ubc.ca/ylucet/DS/KnuthMorrisPratt.html>

For replace mechanism, it will use modified linear probing. The worst case for this would be:

It supposed to quadratic. But why not *O(N2)*? This is the power of memoization that can be generated with the help of KMP. KMP mechanism is useful for scan and store the targets addresses. Then it will only need to check whether the corresponding address is in the index or not (the first and last characters of the pattern). Also, to improve the work, the probing is also equipped with front probing and back probing to make it *O(N)*.

1. **Implementation in Programming**
2. **FindNReplace.hpp**

/\*

\* FindNReplace.hpp

\*

\* Header for FindNReplace

\* Code in FindNReplace.cpp

\*/

#ifndef BIN\_CLASSES\_FindNReplace\_HPP\_

#define BIN\_CLASSES\_FindNReplace\_HPP\_

#include<iostream>

#include<vector>

#include<algorithm>

#include<windows.h>

using namespace std;

class FindNReplace{

private:

    HANDLE hConsole;

    CONSOLE\_SCREEN\_BUFFER\_INFO consoleInfo;

    WORD saved\_attributes;

    int strlength;

    int changed;

    string pattern;

    vector<int> repetlist;

    vector<int> frontmatching;

    vector<int> backmatching;

    vector<bool> tableofchange;

    string texts;

    void patterncheck();

    void replaceallreverse(string replacer, int prop);

    void replaceallfront(string replacer,int prop);

    void replacefront(int target, string replacer, int prop);

    void replaceback(int target, string replacer, int prop);

public:

    FindNReplace();

    void run(string sampl, string texts);

    string returning();

    void printout();

    void printout(int choosing);

    void replaceall(string replacer);

    void replace(int order, string replacer);

    void manualinserttext(string dat);

    void run(string sampl);

    void clearing();

    int patternfoundnumreturn();

    bool isallchanged();

};  
#endif /\* BIN\_CLASSES\_FindNReplace\_HPP\_ \*/

1. **FindNReplace.cpp**

#include "FindNReplace.hpp"

using namespace std;

//Initializer

FindNReplace::FindNReplace(){

    this->hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

    GetConsoleScreenBufferInfo(this->hConsole, &this->consoleInfo);

    this->saved\_attributes = this->consoleInfo.wAttributes;

    this->frontmatching.reserve(3500000);

    this->backmatching.reserve(3500000);

    this->tableofchange.reserve(3500000);

    this->strlength = 0;

    this->changed = 0;

}

//FindNReplace algorithm to scan pattern

void FindNReplace::patterncheck(){

    int patlen = this->pattern.length(), j = 0;

    this->repetlist.clear();

    this->repetlist.resize(patlen);

    for(int i = 1; i < patlen; i++){

        while(j >= 0 && this->pattern[j] != this->pattern[i]){

            if(j-1 < 0){

                j = -1;

                break;

            }

            j = this->repetlist[j - 1];

        }

        j += 1;

        this->repetlist[i] = j;

    }

}

//Insert text manually

void FindNReplace::manualinserttext(string dat){

    if(dat != this->texts){

        this->texts = dat;

        this->strlength=this->texts.length();

        this->frontmatching.clear();

        this->backmatching.clear();

        this->tableofchange.clear();

        this->changed=0;

    }

}

//Run main FindNReplace mechanism

void FindNReplace::run(string sampl){

    this->run(sampl,this->texts);

}

void FindNReplace::run(string sampl, string txts){

    if((sampl == this->pattern || sampl == "" )&& txts == this->texts){

        return;

    }

    if(sampl != this->pattern){

        this->pattern=sampl;

        this->patterncheck();

    }

    if(txts != this->texts){

        this->texts = txts;

        this->strlength=this->texts.length();

    }

    int samplen = this->pattern.length(), txtlen = this->texts.length(), j = 0;

    this->frontmatching.clear();

    this->backmatching.clear();

    this->tableofchange.clear();

    for(int i = 0; i < txtlen; i++){

        while(j >= 0 && this->texts[i] != this->pattern[j]){

            if (j-1 < 0){

                j = -1;

                break;

            }

            j = this->repetlist[j-1];

        }

        j += 1;

        if(j == samplen){

            j = this->repetlist[samplen - 1];

            this->frontmatching.push\_back(i - samplen + 1);

            this->backmatching.push\_back(i);

            this->tableofchange.push\_back(false);

        }

    }

    this->changed = 0;

}

//Print the result of FindNReplace & Find and Replace (ALL RESULT)

void FindNReplace::printout(){

    int runaround = 0;

    for(int k = 0; k < this->strlength;k++){

        if(runaround < this->frontmatching.size()){

            if((k >= this->frontmatching[runaround] && k <= this->backmatching[runaround])){

                if(!this->tableofchange[runaround])SetConsoleTextAttribute(this->hConsole, 10);

                else SetConsoleTextAttribute(this->hConsole, 8);

                runaround = k == this->backmatching[runaround] ? runaround + 1 : runaround;

            }

        }

        cout<<this->texts[k];

        SetConsoleTextAttribute(this->hConsole,this->saved\_attributes);

    }

    cout<<endl;

}

//Print the result of FindNReplace & Find and Replace (CERTAIN PART RESULT)

void FindNReplace::printout(int choosing){

    for(int k = 0; k < this->strlength;k++){

        if((k >= this->frontmatching[choosing-1] && k <= this->backmatching[choosing-1])){

            if(!this->tableofchange[choosing-1])SetConsoleTextAttribute(this->hConsole, 10);

        }

        cout<<this->texts[k];

        SetConsoleTextAttribute(this->hConsole,this->saved\_attributes);

    }

    cout<<endl;

}

//Replace ALL of the result of string finding

void FindNReplace::replaceall(string replacer){

    if(this->tableofchange.size() <= 0 || replacer == this->pattern){

        return;

    }

    if(this->changed < this->tableofchange.size()){

        int workout = replacer.length()-this->pattern.length();

    if(workout == 0){

        for(int i = 0;i < this->frontmatching.size();i++){

            if(!this->tableofchange[i]){

                int tempprober = this->frontmatching[i];

                for(int j = 0;j < replacer.length();j++){

                    this->texts[tempprober]=replacer[j];

                    tempprober++;

                }

                this->changed++;

                this->tableofchange[i] = true;

            }

        }

    }

    else if(workout>0){

        this->replaceallreverse(replacer,workout);

    }

    else{

        this->replaceallfront(replacer,workout);

    }

    }

}

//String 'REPLACE ALL' mechanism from behind

void FindNReplace::replaceallreverse(string replacer, int prop){

    int prevlast=this->strlength-1;

    int occurtoken = this->tableofchange.size();

    this->strlength=this->strlength+(prop \* (occurtoken - this->changed));

    if(this->strlength > this->texts.length()){

        this->texts.resize(this->strlength);

    }

    int prober=this->strlength - 1;

    while(occurtoken > 0 /\*prevlast >= this->frontmatching[0]\*/){

        while(prevlast != this->backmatching[occurtoken - 1]){

            this->texts[prober]=this->texts[prevlast];

            prober--;

            prevlast--;

        }

        if(prevlast == this->backmatching[occurtoken-1] && !this->tableofchange[occurtoken-1]){

            occurtoken--;

            prevlast = this->frontmatching[occurtoken] - 1;

            this->backmatching[occurtoken]=prober;

            for(int i = replacer.length()-1; i >= 0; i--){

                this->texts[prober] = replacer[i];

                prober--;

            }

            this->frontmatching[occurtoken]=prober + 1;

            this->tableofchange[occurtoken]=true;

            this->changed++;

        }

        else{

            occurtoken--;

            this->backmatching[occurtoken]=prober;

            while(prevlast > this->frontmatching[occurtoken]){

                this->texts[prober]=this->texts[prevlast];

                prober--;

                prevlast--;

            }

            this->frontmatching[occurtoken]=prober;

        }

    }

}

//String 'REPLACE ALL' mechanism from front

void FindNReplace::replaceallfront(string replacer, int prop){

    int occurtoken = 0;

    int alreadychanged = this->changed;

    int probpoint=this->frontmatching[0];

    int j = probpoint;

    while(j < this->strlength){

        if(occurtoken < this->tableofchange.size()){

            if(j == this->frontmatching[occurtoken] && !this->tableofchange[occurtoken]){

                j += this->pattern.length();

                this->frontmatching[occurtoken]=probpoint;

                for(int i = 0; i < replacer.length(); i++){

                    this->texts[probpoint]=replacer[i];

                    probpoint++;

                }

                this->backmatching[occurtoken] = probpoint - 1;

                this->tableofchange[occurtoken] = true;

                this->changed++;

                occurtoken++;

            }

            else if(j == this->frontmatching[occurtoken] && this->tableofchange[occurtoken]){

                this->frontmatching[occurtoken]=probpoint;

                while(j < this->backmatching[occurtoken]){

                    this->texts[probpoint] = this->texts[j];

                    probpoint++;

                    j++;

                }

                this->backmatching[occurtoken] = probpoint;

                occurtoken++;

            }

            else{

                this->texts[probpoint] = this->texts[j];

                probpoint++;

                j++;

            }

        }

        else{

            this->texts[probpoint] = this->texts[j];

            probpoint++;

            j++;

        }

    }

    this->strlength=this->strlength + ((this->tableofchange.size() - alreadychanged)\*prop);

}

//Replace CERTAIN PART of the result of string finding

void FindNReplace::replace(int order, string replacer){

    if(this->tableofchange.size() <= 0 || replacer == this->pattern){

        return;

    }

    int target = order < 1 ? 0 : order > this->tableofchange.size() ? this->tableofchange.size() - 1: order - 1;

    int workout = replacer.length()-this->pattern.length();

    if(!this->tableofchange[target]){

        if(workout == 0){

            int tempprober = this->frontmatching[target];

            for(int j = 0;j < replacer.length();j++){

                this->texts[tempprober]=replacer[j];

                tempprober++;

            }

            this->changed++;

            this->tableofchange[target] = true;

        }

        else if(workout > 0){

            this->replaceback(target,replacer,workout);

        }

        else{

            this->replacefront(target,replacer,workout);

        }

    }

}

//String 'REPLACE PARTIAL' mechanism from front

void FindNReplace::replacefront(int target, string replacer, int prop){

    int j = this->frontmatching[target];

    int probpoint=j;

    j += this->pattern.length();

    for(int i = 0; i < replacer.length(); i++){

        this->texts[probpoint]=replacer[i];

        probpoint++;

    }

    this -> backmatching[target] = probpoint - 1;

    this -> tableofchange[target] = true;

    this -> changed++;

    int aftertarget = target + 1;

    while(j < this->strlength){

        if(aftertarget < this->tableofchange.size()){

            if(j == this->frontmatching[aftertarget]){

                this->frontmatching[aftertarget] = probpoint;

                while(j < this->backmatching[aftertarget]){

                    this->texts[probpoint] = this->texts[j];

                    probpoint++;

                    j++;

                }

                this->backmatching[aftertarget] = probpoint;

                aftertarget++;

            }

        }

        this->texts[probpoint] = this->texts[j];

        probpoint++;

        j++;

    }

    this->strlength=this->strlength + prop;

}

//String 'REPLACE PARTIAL' mechanism from behind

void FindNReplace::replaceback(int target, string replacer, int prop){

    int prevlast = this->strlength - 1;

    this->strlength = this->strlength + prop;

    if(this->strlength > this->texts.length()){

        this->texts.resize(this->strlength);

    }

    int prober=this->strlength - 1;

    int helper=this->tableofchange.size() - 1;

    while(prevlast > this->backmatching[target]){

        if(prevlast == this->backmatching[helper]){

            this->backmatching[helper] = prober;

            while(prevlast > this->frontmatching[helper]){

                this->texts[prober] = this->texts[prevlast];

                prober--;

                prevlast--;

            }

            this->frontmatching[helper] = prober;

            helper--;

        }

        this->texts[prober] = this->texts[prevlast];

        prober--;

        prevlast--;

    }

    this->backmatching[target]=prober;

    for(int i = replacer.length()-1; i >= 0; i--){

        this->texts[prober] = replacer[i];

        prober--;

    }

    this->frontmatching[target]=prober + 1;

    this->tableofchange[target]=true;

    this->changed++;

}

//Return Result String

string FindNReplace::returning(){

    this->texts.resize(this->strlength);

    return this->texts;

}

//Clear search results

void FindNReplace::clearing(){

this->pattern="";

    this->frontmatching.clear();

    this->backmatching.clear();

    this->tableofchange.clear();

    this->changed = 0;

}

//Return the number of discovered results

int FindNReplace::patternfoundnumreturn(){

    return this->tableofchange.size();

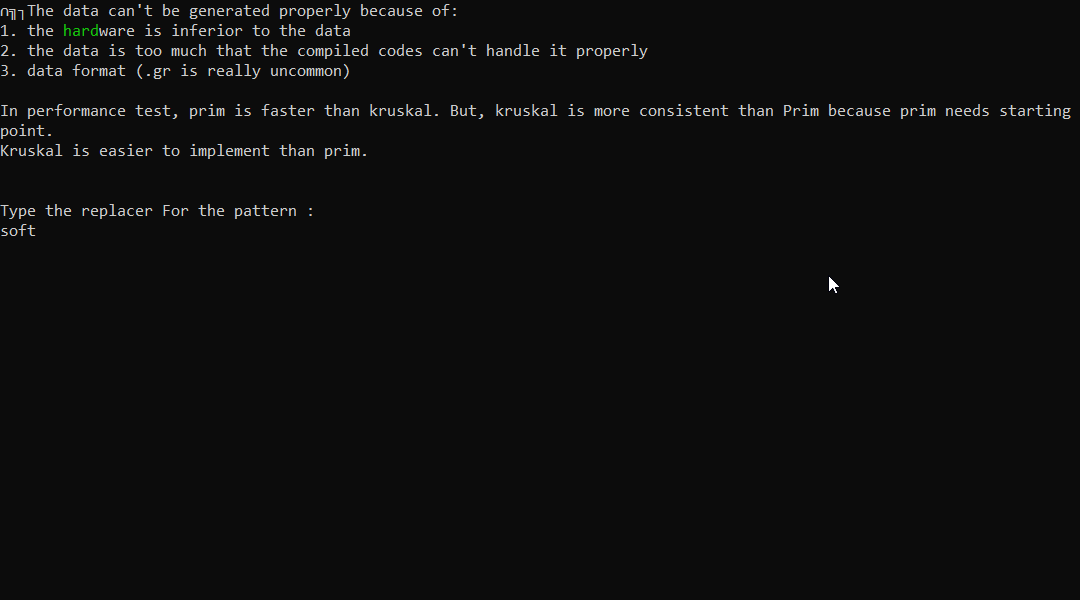
}

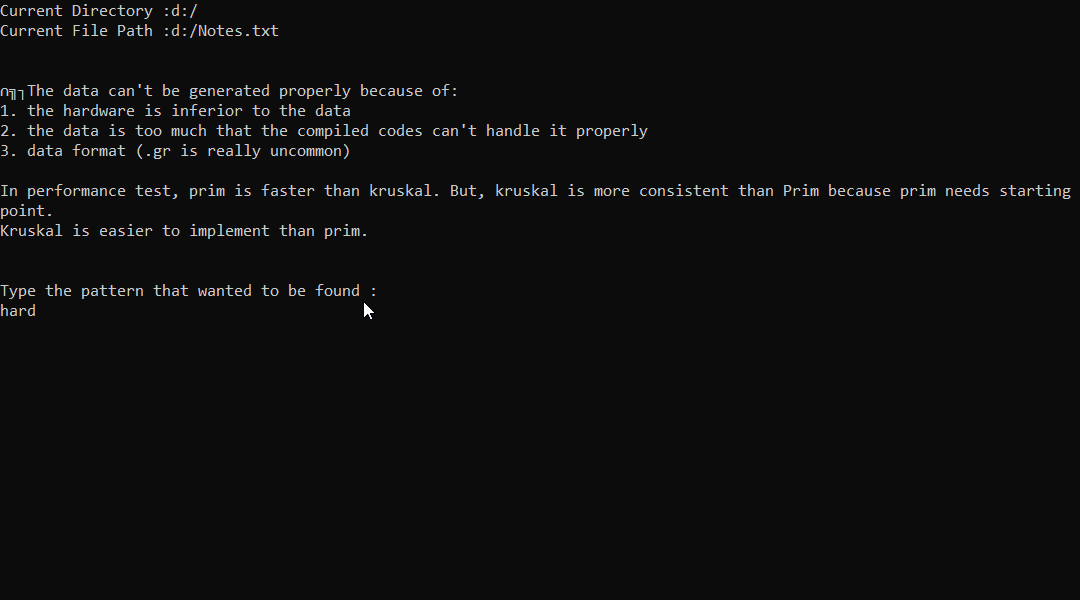
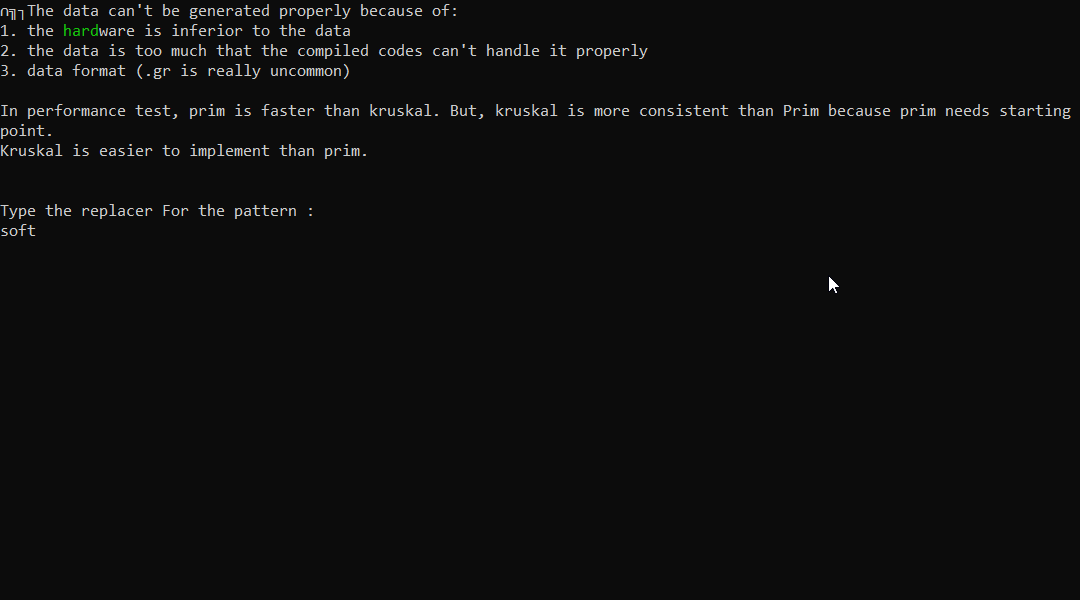
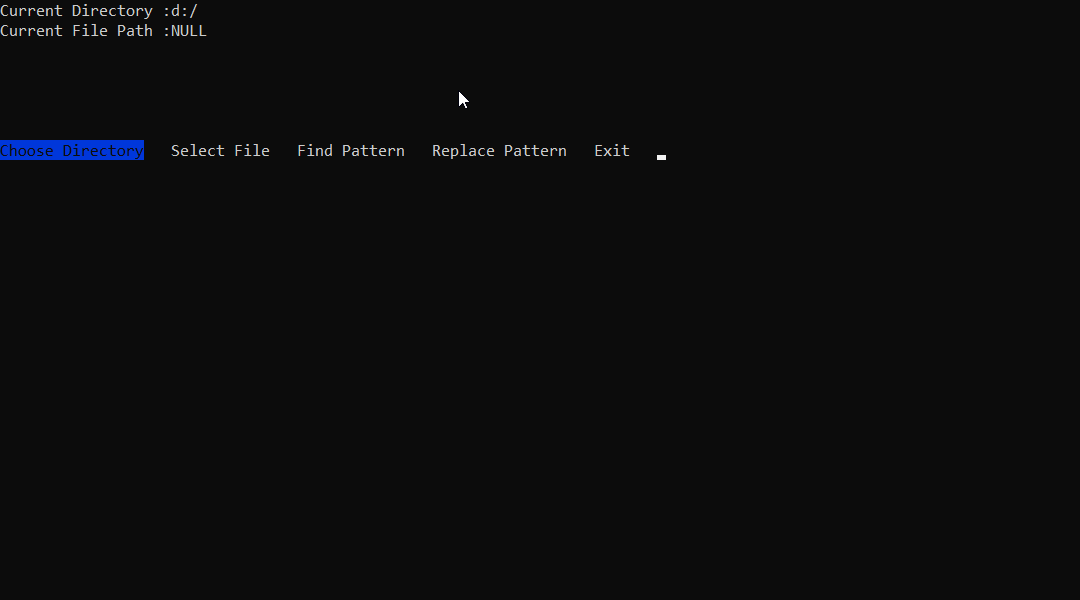
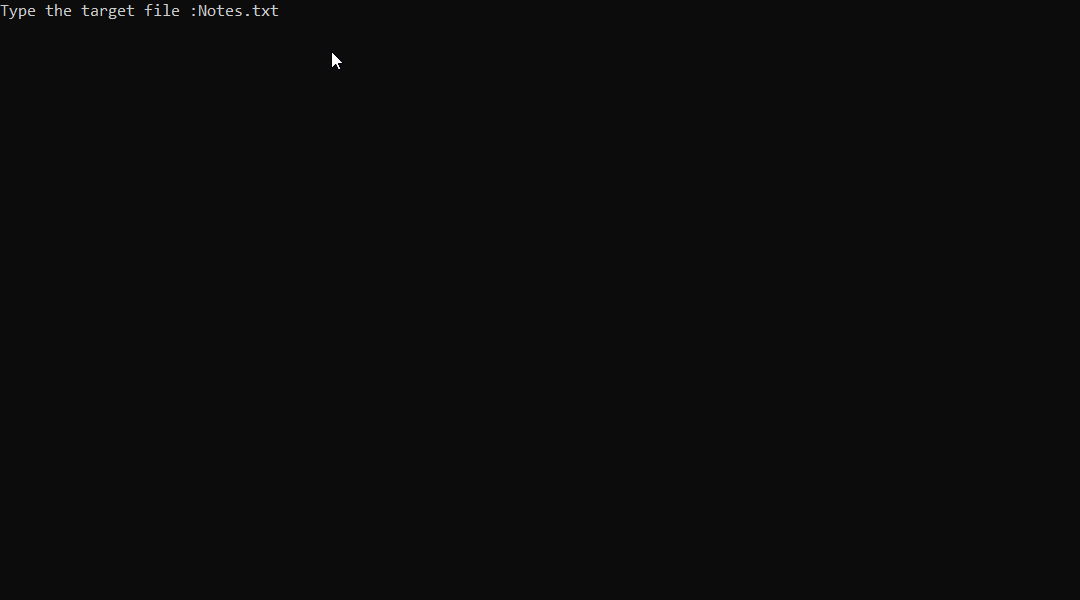
//Return the status whether all of the search results has been replaced

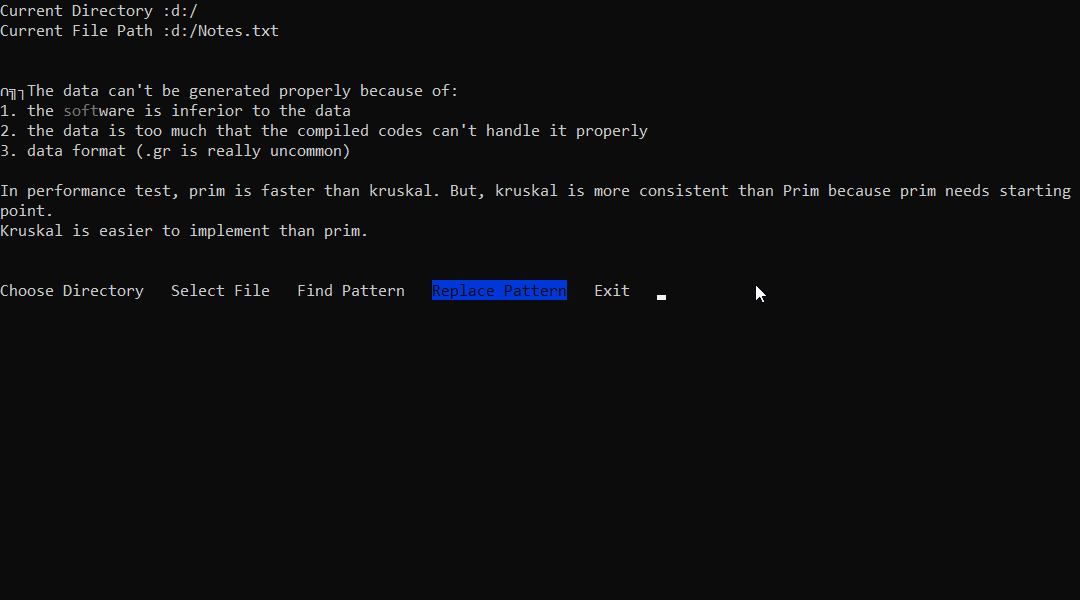
bool FindNReplace::isallchanged(){

    return this->changed == this->tableofchange.size();

}







1. **Conclusion**

There are several conclusions for this project:

1. Dynamic programming, in this project, is essential for optimization in this project.
2. KMP algorithm, by the code structure and mechanics, is definetely an example of dynamic programming-oriented algorithm.
3. Memoization is really useful for find and replace, because it will tag any of correspondence pattern that can be possibly become the one that will be replaced.
4. In order for the pure memoization to work, it will need an efficient pattern-finding algorithm. That’s where KMP comes in. It will scan the whole string or text, then store it to a container (in this program case vector or resizable array). It will form a sort of active caching called memoization.
5. With memoization, the worst case for pattern replacement is reduced into *O(N)*.
6. **Links**

<https://github.com/HelixAngler/FelixAnggara_AoA2018_FinalProject.git>

<https://www.dropbox.com/s/4q1d79kfmos09nt/Final%20project%20aoa.mp4?dl=0>