

**DAA UE23CS241B - Orange Problem**

**Implementation and Performance evaluation of Horspool and Boyer-Moore algorithms**

1. **Horspool Algorithm**

Implement the Horspool Algorithm, display the shift table contents and the starting indices of all the occurrences of the pattern in the text. Count the total number of comparisons involved.

1. **Boyer Moore Algorithm**

Implement the Boyer Moore algorithm, display the good suffix shift table and the bad character shift table. Also print the starting indices of all the occurrences of the pattern in the text. Count the total number of comparisons involved.

1. **Performance Evaluation**

Measure and compare execution times and number of comparisons for both algorithms and plot a graph to visualize their performance. (Hint: gnuplot can be used)

**Two graphs to be plotted:**

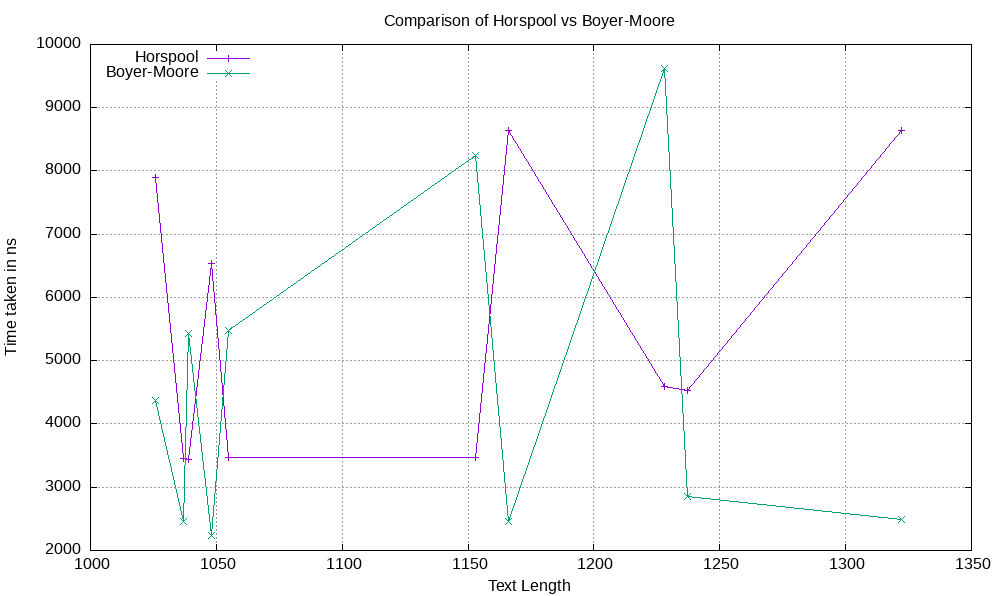
1. Text length vs Execution Time for given test case against both algorithms
2. Text length vs Number of comparisons for both algorithms

**Instructions:**

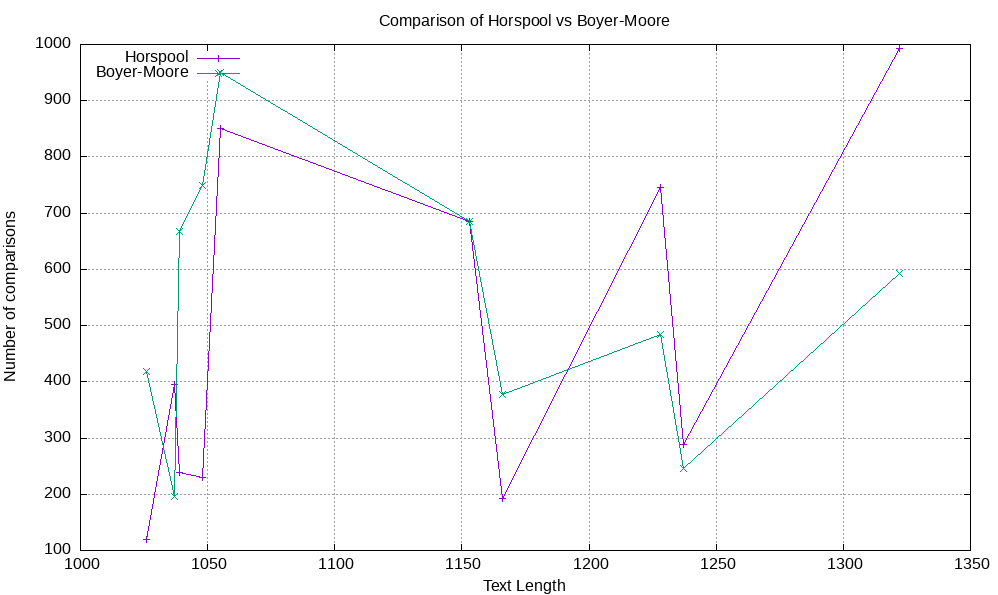
1. Boilerplate codes will handle the input, output and execution time calculation.
2. Input is available in ***input.txt***
3. Fill the code snippets in the TODO sections. The existing ***fprintf*** statements need NOT be touched.
4. The ***horspool\_values.txt*** and ***boyermoore\_values.txt*** files generated by the code can be used for plotting the graphs, sample graphs attached for reference (does not show the true relation between both algorithms).
5. Final files to be submitted - horspool\_srn.c, boyermoore\_srn.c, horspool\_output\_srn.txt, boyermoore\_output\_srn.txt, comparison\_graph\_srn.png and time\_graph\_srn.png (Naming convention is to be strictly followed)

**Sample Graphs:**

**Time Plot:**

****

**Comparison Plot:**

****

**Boilerplate Code:**

**1. Horspool**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<sys/time.h>

#include<time.h>

// Fill in the TODO sections as required and DO NOT TOUCH any of the fprintf statements

void init\_table(int\* shift\_table,int n)

{

//TODO

}

// Construct the Bad Character Shift table

void preprocess(int\* shift\_table,char\* pattern)

{

//TODO

}

int string\_match(int\* shift\_table,char\* pattern,char\* text,FILE\* output\_file)

{

// TODO Variables initialization

int matches;

int star\_pos;

char star\_char;

int cmp;

int occurance;

fprintf(output\_file,"Occurrences:");

while(star\_pos<strlen(text))

{

//TODO find index of occurances

if(matches==strlen(pattern))

{

fprintf(output\_file,"%d,",index);

//TODO

}

}

fprintf(output\_file,"\n");

fprintf(output\_file,"Comparisons:%d\n\n",cmp);

return cmp;

}

void print\_table(int\* shift\_table, FILE\* output\_file)

{

fprintf(output\_file,"BCST:\n");

for(int i=0;i<26;i++)

{

fprintf(output\_file,"%c:%d",(char)(i+97), shift\_table[i]);

}

}

void testcase(FILE\* values\_file, FILE\* input\_file, FILE\* output\_file)

{

char text[2000];

char pattern[100];

fscanf(input\_file,"%s",text);

fscanf(input\_file,"%s",pattern);

int\* shift\_table=calloc(26,sizeof(int));

init\_table(shift\_table,strlen(pattern));

preprocess(shift\_table,pattern);

print\_table(shift\_table, output\_file);

clock\_t start = clock();

int cmp = string\_match(shift\_table,pattern,text,output\_file);

clock\_t end = clock();

int elapse=(int)(((double)(end-start))/CLOCKS\_PER\_SEC\*1000000000); //seconds to nanoseconds

fprintf(values\_file,"%ld,%ld,%d,%d\n",strlen(pattern),strlen(text),cmp,elapse);

}

int main()

{

FILE \*input\_file = fopen("input.txt", "r");

FILE \*output\_file = fopen("horspool\_output.txt", "w");

FILE \*values\_file = fopen("horspool\_values.txt", "w");

if (!input\_file || !output\_file || !values\_file) {

printf("Error opening file!\n");

return 1;

}

int testcases;

fscanf(input\_file,"%d",&testcases);

int count = 0;

fprintf(values\_file, "patternlen,textlen,cmp,timetaken\n");

while(count < testcases)

{

testcase(values\_file, input\_file, output\_file);

count += 1;

}

fclose(input\_file);

fclose(output\_file);

fclose(values\_file);

return 0;

}

**2. Boyer Moore**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include<sys/time.h>

#include<time.h>

// Fill in the TODO sections as required and DO NOT TOUCH any of the fprintf statements

// Function to create the Bad Character Shift Table

int\* bcst\_create(const char\* pattern, int pattern\_len) {

// TODO BCST creation

return bcst;

}

// Function to create the Good Suffix Shift Table

int\* gsst\_create(const char\* pattern, int pattern\_len) {

int\* gsst = (int\*)malloc((pattern\_len + 1) \* sizeof(int));

if (gsst == NULL) {

perror("Memory allocation failed");

exit(EXIT\_FAILURE);

}

for (int x = 1; x <= pattern\_len; x++) {

int r2 = 0;

char\* suffix = (char\*)malloc((x + 1) \* sizeof(char));

if (suffix == NULL) {

perror("Memory allocation failed");

exit(EXIT\_FAILURE);

}

for(int i=0;i<x;i++)

{

suffix[x-i-1] = pattern[pattern\_len-i-1];

}

suffix[x] = '\0';

char mis\_char = (pattern\_len - x - 1 >= 0) ? pattern[pattern\_len - x - 1] : '~';

char\* rev\_pattern = (char\*)malloc(pattern\_len \* sizeof(char));

if (rev\_pattern == NULL) {

perror("Memory allocation failed");

exit(EXIT\_FAILURE);

}

for(int i=0;i<pattern\_len-1;i++)

rev\_pattern[i] = pattern[pattern\_len-i-2];

rev\_pattern[pattern\_len-1] = '\0';

char\* rev\_suffix = (char\*)malloc((x + 1) \* sizeof(char));

if (rev\_suffix == NULL) {

perror("Memory allocation failed");

exit(EXIT\_FAILURE);

}

for(int i=0;i<x;i++)

rev\_suffix[i] = suffix[x-i-1];

rev\_suffix[x] = '\0';

int count = 0;

while (1) {

char\* pos\_ptr = strstr(rev\_pattern, rev\_suffix);

if (pos\_ptr == NULL) {

r2 = 1;

break;

}

int pos = pos\_ptr - rev\_pattern;

char check\_char = (pos + x < pattern\_len - 1) ? rev\_pattern[pos + x] : '`';

if (check\_char != mis\_char) {

gsst[x] = pos + count + 1;

break;

}

rev\_pattern += pos + 1;

count += pos + 1;

}

char\* suffix\_ptr = suffix;

if (r2) {

char\* prefix = (char\*)malloc(pattern\_len \* sizeof(char));

if (prefix == NULL) {

perror("Memory allocation failed");

exit(EXIT\_FAILURE);

}

for (int i = 0; i < pattern\_len; i++) {

prefix[i] = pattern[i];

}

prefix[pattern\_len] = '\0';

while (1) {

if (strlen(suffix) == 1) {

gsst[x] = pattern\_len;

break;

}

suffix++;

if (strncmp(prefix, suffix, strlen(suffix)) == 0) {

gsst[x] = pattern\_len - strlen(suffix);

break;

}

}

//free(prefix);

}

//free(suffix\_ptr);

//free(rev\_suffix);

//free(rev\_pattern);

}

return gsst;

}

// Boyer-Moore search function

int boyer\_moore(const char\* text, const char\* pattern, int\* bcst, int\* gsst, FILE\* output\_file)

{

// TODO variables initializations

fprintf(output\_file,"Occurrences:");

while (pos < text\_len) {

// TODO find indices of occurances

if (match == pattern\_len) {

fprintf(output\_file,"%d,", end - pattern\_len + 1);

//TODO

}

}

fprintf(output\_file,"\n");

return comparisons;

}

void testcase(FILE\* values\_file, FILE\* input\_file, FILE\* output\_file)

{

char text[2000];

char pattern[500];

fscanf(input\_file,"%s",text);

fscanf(input\_file,"%s",pattern);

int pattern\_len = strlen(pattern);

int\* bcst = bcst\_create(pattern, pattern\_len);

fprintf(output\_file,"BCST:\n");

for(int i = 0; i < 26; i++){

fprintf(output\_file,"%c:%d, ", (char)(i+'a'), bcst[i]);

}

fprintf(output\_file,"\n");

int\* gsst = gsst\_create(pattern, pattern\_len);

fprintf(output\_file,"GSST:\n");

for(int i = 1; i <= pattern\_len; i++){

fprintf(output\_file,"%d:%d, ", i, gsst[i]);

}

fprintf(output\_file,"\n");

clock\_t start = clock();

int comparisons = boyer\_moore(text, pattern, bcst, gsst, output\_file);

fprintf(output\_file,"Comparisons:%d\n\n", comparisons);

free(bcst);

free(gsst);

clock\_t end = clock();

int elapse=(int)(((double)(end-start))/CLOCKS\_PER\_SEC\*1000000000); //seconds to nanoseconds

fprintf(values\_file,"%d,%ld,%d,%d\n",pattern\_len,strlen(text),comparisons,elapse);

}

int main() {

FILE \*input\_file = fopen("input.txt", "r");

FILE \*output\_file = fopen("boyermoore\_output.txt", "w");

FILE \*values\_file = fopen("boyermoore\_values.txt", "w");

if (!input\_file || !output\_file || !values\_file) {

printf("Error opening file!\n");

return 1;

}

int testcases;

fscanf(input\_file,"%d",&testcases);

int count = 0;

fprintf(values\_file, "patternlen,textlen,cmp,timetaken\n");

while(count < testcases)

{

testcase(values\_file,input\_file, output\_file);

count += 1;

}

fclose(input\_file);

fclose(output\_file);

fclose(values\_file);

return 0;

}

Marks Distribution

1. Horspool implementation 3M
2. Boyer-Moore implementation 5M
3. Output file and graph generation 2M