МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Кнута-Морриса-Пратта

Студент гр. 8303 _____ Пушпышев А.И.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Реализовать алгоритм Кнута-Морриса-Пратта, найти индексы вхождения подстроки в строку, а также разработать алгоритм проверки двух строк на циклический сдвиг.

Вариант 1.

Подготовка к распараллеливанию: работа по поиску разделяется на k равных частей, пригодных для обработки k потоками (при этом длина образца гораздо меньше длины строки поиска).

Задание

Первая часть:

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P($|P|\leq 15000$) и текста T ($|T|\leq 5000000$) найдите все вхождения P в T.

Вход:
Первая строка - P
Вторая строка - T
Выход:
индексы начал вхождений P в T, разделенных запятой, если P не входит в T, то вывести -1−1

Sample Input:

```
ab
abab
```

Sample Output:

```
0,2
```

Вторая часть:

Заданы две строки A ($|A|\leq 5000000$) и B ($|B|\leq 5000000$).
Определить, является ли A циклическим сдвигом B (это значит, что A и B имеют одинаковую длину и A состоит из суффикса B, склеенного с префиксом B).
Например, defabc является циклическим сдвигом abcdef.
Вход:
Первая строка - A
Вторая строка - B
Выход:

Если A вляется циклическим сдвигом B, индекс начала строки B в A, иначе вывести -1−1. Если возможно несколько сдвигов вывести первый индекс.

Sample Input:

```
defabc
abcdef
```

Sample Output:

```
3
```

Описание алгоритма КМП.

На вход алгоритма передается строка-образец, вхождения которой нужно найти, и строка-текст, в которой нужно найти вхождения.

Оптимизация – строка-текст считывается посимвольно, в памяти хранится текущий символ.

Алгоритм сначала вычисляет префикс-функцию строки-образца.

Далее посимвольно считывается строка-текст. Переменная-счетчик изначально k = 0. При каждом совпадении k-го символа образца и i-го символа текста счетчик увеличивается на 1. Если k = размер образца, значит вхождение найдено. Если очередной символ текста не совпал с k-ым символом образца, то сдвигаем образец, причем точно знаем, что первые k символов образца совпали с символами строки и надо сравнить k+1-й символ образца (его индекс k) с i-м символом строки.

Сложность алгоритма по операциям: O (m + n), m – длина образца, n – длина текста.

Сложность алгоритма по памяти: O (m), m – длина образца.

Префикс-функция - это такая наибольшая длина наибольшего собственного суффикса подстроки, совпадающего с ее префиксом.
Алгоритм вычисления:
-считать значения префикс-функции p[i] по очереди: от i:=1 до i:=n-1(p[0]:= 0)
-для подсчета данного значения p[i] используем переменную j, отображающая длину текущего образца. Изначально - j:=p[i-1]
-Рассматриваем образец длины j, для чего сравниваем символы исходной строки j и i. Если они совпадают, то p[i] := j+1 и переходим к следующему индексу i++; иначе уменьшаем длину j, полагая ее равной p[j-1] и повторяем этот шаг сначала

-Если дошли до j ==0 без совпадения, то конец перебора и полагаем p[i]=0, переходим к следующему индексу i++

Разбиение исходной строки на куски происходит путем формирования подстрок(для оптимизации по памяти будем запоминать начало и конец таких строк, а не сам отрезок). Для начала определимся с количеством кусков: это результат деления всей строки, в которой проводится поиск на количество потоков, для корректировки решения в случае, если с точки зрения математики число получается не целочисленным, мы используем округление вверх. Начала кусков, будут соответствовать номеру потока N с учетом длины кусков NUM, таким образом начальные позиции кусков в исходной строк формируются как N*NUM для соответствующего куска. В случае с формированием координат следует выделить два случая. Первый случай — это последний поток, поэтому координату последнего куска приравниваем к последнему символу строки, по которой производиться поиск. Второй случай — предыдущие потоки. Формируется как сумма начальной позиции, длины куска на 1 поток и размер искомой строки. В таком случае мы избегаем того, что искомая строка может быть наложена на стык, она будет находиться либо в одном, либо в другом потоке.

В случае, когда размер исходной и искомой строки совпадают, то вместо поиска подстроки(это не имеет смысла, так как в этом случае единственный исход – совпадение искомой и исходный строк) мы рассматриваем ситуацию, как то, что искомая строка является сдвигом исходной. Для этого мы формируем строку путем слияния исходной строки с собой. Далее запускаем основной алгоритм, который необходимо завершить в этом случае один раз, так как искомый сдвиг найден, а остальные сдвиги не требуются условием задачи. Случай, когда строки совпадают, рассматривается как сдвиг исходной на 0.

Описание функций и структур данных.

```
char* read_pattern(char *filename)
```

Функция считывания из файла.

```
void* seek_substring_KMP (void *ptr)
```

Функция, реализующая префикс-функцию строки и алгоритм КМП. Принимает на вход указатель массива входных данных для потоков.

```
struct args{};
```

Структура, содержащая информацию об исходной сроке поиска, искомой строки и границах данного куска строки, в которой работает данный поток.

```
bool flag_cycle; //флаг на то, что задача о поиске циклического сдвига
bool* flag_used; //флаг о том, что циклический сдвиг уже найден
const char *pattern;//исходная строка, в которой идет поиск
char *line;//искомое
long int x0;//начало куска
long int x1;//конец куска
int thread_num;//номер данного потока
```

Тестирование.

Так как гарантируется, что искомая строка много меньше строки, в которой производится поиск, поэтому тестирование проводилось на количестве потоков равному трем.

Входные данные	Вывод
abababababbbaaababababab bab 3	Threads count = 3 Part of this thread is Part of this thread is bababab Pos = 14 Pos = 16 Pos = 18 abbbaaaba Part of this thread is ababbabab Pos = 1 Pos = 4 Pos = 6 Work time = 0.014000
qwerqwerqwerqwer werqwerqwerqwerq 2	Threads count = 2 Part of this thread is qwerqwerqwerqwerqwerqwerqwerqwe Pos = 1 Part of this thread is qwerqwerqwerqwer -1

	Work time = 0.006000
qwerqwer qwerqwer 1	Threads count = 1 Part of this thread is qwerqwerqwerqwer Pos = 0 Work time = 0.004000
qwertyuiopuiopuiop uiop 4	Threads count = 4 Part of this thread is qwPart of this thread is tyuiopu Pos = 6 Part of this thread is opuiopu Pos = 10 Part of this thread is opuiop Pos = 14 ertyu Work time = 0.009000
abcd kl 2	Threads count = 2 Part of this thread is abc Part of this thread is cd -1 Work time = 0.004000
qwerqwer klmnoprst 1	Threads count = 1 Part of this thread is qwerqwer -1 Work time = 0.003000
havanagilahavanagilahavanagilahavanagila agil 5	Threads count = 5 Part of this thread is havanagilah Part of this thread is nagilahavan Pos = 25 Part of this thread is vanagila Pos = 35 Pos = 5

	Part of this thread is gilahavanag Part of this thread is lahavanagil Pos = 15 Work time = 0.017000
tyuioptyuioptyuiop yuioptyuioptyuiopt 3	Threads count = 3 Part of this thread is tyuioptyuioptyuioptyuioptyuio Pos = 1 Part of this thread is tyuioptyuiop -1 Part of this thread is tyuioptyuioptyuioptyuiop oleT Work time = 0.014000

Выводы.

В ходе выполнения лабораторной работы был реализован алгоритм КМП и подготовка к распараллеливанию. Для убедительности решения код был написан с использованием POSIX threads. Программа учитывает, что искомая подстрока может находится на стыке двух поток, и работает корректно. Данные полученные в промежуточном выводе позволили провести пошаговую проверку корректности работы алгоритма с контролем значений переменных. При этом мы можем контролировать состояние потока, в случае если в данной подстроке не было найдено решение мы получим -1, что говорит о том, что если каждый поток вывел -1, то решения нет.

ПРИЛОЖЕНИЕ А.
Исходный код

```
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>
#include <math.h>

struct Args
{
    bool flag_cycle;
    bool* flag_used;
    const char *pattern;//исходный поиск
    char *line;//искомое
    long int x0;//начало
    long int x1;//конец
    int thread_num;
};

char* read_pattern(char *filename)
{
    FILE *fp;
    long int lSize;
    char *buffer;
    fp = fopen ( filename , "rb" );
    if( !fp )
        perror(filename),exit(1);
    fseek( fp , 0L , SEEK_END);
    lSize = ftell( fp );
    rewind( fp );
    printf("Size of %s = %ld\n", filename, lSize);
    /* allocate memory for entire content */
    buffer = (char*)calloc( 1, lSize+1 );
    if( !buffer ) fclose(fp),fputs("memory alloc fails",stderr),exit(1);

    /* copy the file into the buffer */
    if( 1 != fread(buffer , lSize, 1 , fp) )
        fclose(fp),free(buffer),fputs("entire read fails",stderr),exit(1);
    fclose(fp);
    return buffer;
}

void* seek_substring_KMP (void *ptr)
{
    struct Args * a = (struct Args*)ptr;
    long int ofst = 0;
    long int M = strlen(a->line);
    long int * pi =(long int *)malloc(M * sizeof(long int)); //динамический массив
длины M

    //Вычисление префикс-функции
    pi[0] = 0;

    int sch = 0;

    for (int p = a->x0, b = printf("Part of this thread is "); p <= a->x1; p++) {
        printf("%c", a->pattern[p]);
```

```
    }
    printf("\n");

    for(long int i = 1; i < M; i++)
    {
        while(ofst > 0 && a->line[ofst] != a->line[i])
            ofst = pi[ofst - 1];
        if(a->line[ofst] == a->line[i])
            ofst++;
        pi[i] = ofst;
    }

    //поиск
    for(long int i = a->x0, j = 0; i <= a->x1; i++)
    {
        while(j > 0 && a->line[j] != a->pattern[i])
            j = pi[j - 1];
        //вывод текущего состояния сдвинутой позиции j
        //контроль позиции в исходной строке
        //отслеживание сравнения символов(ход алгоритма)
        printf("\n>Thread number %d, current j is %ld\n", a->thread_num, j);
        printf("\n>Thread number %d, position in pattern now %ld\n", a->thread_num,
i);
        printf("\n>Thread number %d, checking difference between |%c| in substring on
%ld and |%c| in pattern on %ld\n", a->thread_num, a->line[j], j, a->pattern[i], i);
        if(a->line[j] == a->pattern[i])
            j++;
        printf("\n>Thread number %d, current state of j is %ld\n>Position in pattern
now %ld\n", a->thread_num, j, i);
        if (j == M)
        {
            sch++;
            if(a->flag_cycle) {
                if(i-j+1 < M && !(*a->flag_used)){
                    printf("Pos = %ld\n", i - j + 1);
                    *a->flag_used = true;
                }
            } else printf("Pos = %ld\n", i - j + 1);
        }
        //отслеживания количества верных ответов на текузий момент
        printf("\n>Thread number %d, count of Ok positions is %ld\n", a->thread_num,
sch);
    }
    free (pi); /* освобождение памяти массива pi */
    if(a->x1 == strlen(a->pattern) - 1 && sch == 0)
        printf("-1\n");
    pthread_exit(NULL);
}

int main(int argc, char** argv)
{

    char * pattern = (char*)malloc((sizeof(char) * 5000001));
    char * line = (char*)malloc((sizeof(char) * 5000001));
    scanf("%s", pattern);
    scanf("%s", line);
    bool flag = false;

    //если строки совпадают по размеру, запускаем алгоритм поиска сдвига
    //в случае совпадения строк выдаст начальную позицию
    if (strlen(pattern) == strlen(line)){
```

```
        char * buf = (char*)malloc(sizeof(char)*(strlen(pattern)*2+1));
        for(int i = 0; i < strlen(pattern)*2; i++)
            buf[i] = pattern[i % strlen(pattern)];
        buf[strlen(pattern)*2] = 0;
        free(pattern);
        pattern = buf;
        flag = true;
    }

    int threads_count = 1;
    scanf("%d", &threads_count);
    printf("Threads count = %d\n", threads_count);
    int NUM = ceil((strlen(pattern)/(threads_count)));//количество кусков
    struct Args * a = (struct Args*)malloc(threads_count * sizeof(struct
Args));//объект аргументов
    pthread_t *threads = (pthread_t*)malloc(threads_count*sizeof(pthread_t));//массив
потоков
    int error_code;

    bool check = false;

    for(int i = 0; i < threads_count; i++)
    {
        //соответсвтвие каждого куска потоку
        a[i].flag_used = &check;
        a[i].flag_cycle = flag;
        a[i].pattern = pattern;
        a[i].line = line;
        a[i].x0 = i * NUM;
        a[i].thread_num = i;
        if(i == threads_count - 1)
        {
            a[i].x1 = strlen(pattern) - 1;
        }
        else
        {
            a[i].x1 = i * NUM + strlen(line) + NUM - 2;
        }
    }
    clock_t t = clock();
    //создание потоков
    for(int i = 0 ; i < threads_count; i++)
    {
        error_code = pthread_create( &threads[i], NULL, seek_substring_KMP, (void*)
&a[i]);
        if(error_code)
        {
            fprintf(stderr,"Error - pthread_create() return code: %d\n",error_code);
            exit(0);
        }
        //else printf("Thread %d is created\n",i);

    }
    //ожидание завершения потоков
    for(int i = 0; i < threads_count; i++)
    {
        pthread_join(threads[i],NULL);
    }
    t = clock() - t;
    free(a);
    free(threads);
```

```
    printf("Work time = %f\n",((float)t)/CLOCKS_PER_SEC);
    return 0;
}
```