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

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Построение и анализ алгоритмов»
Тема: Алгоритм Кнута-Мориса-Пратта

Студент гр.8382 _____ Синельников М.Р

Преподаватель _____ Фирсов М.А

Санкт-Петербург
2020

Цель работы.

Научиться реализовывать алгоритм нахождение подстроки в строке.

Задание.

Задание № 1

Реализуйте алгоритм КМП и с его помощью для заданных шаблона *P (|P* <= 15000) и текста *T(|T|* <= 5000000) найдите все вхождения *P* в *T.*

Вход:

Первая строка — *P*

Вторая строка — *T*

Выход: индексы начало вхождений *P* в *T* , разделённых запятой, если *P* не входит в *T* , то вывести -1

Sample Input:

ab

abab

Sample Output:

0, 2

Задание № 2

Заданы две строки *A(|A|* <= *5000000)* и *B(B* <= *5000000).*

Опрелить, является ли *A* циклическим сдвигом *B*(это значит, что имеют A и *B* одинаковую длину и *A* состоит из *B* суффикса , склееного с префиксом *B*). Например, является циклическим сдвигом.

Вход:

Первая строка — *A*

Вторая строка — *B*

Выход: Если является циклическим сдвигом , индекс начало строки в , иначе вывести -1. Если возможно несколько сдвигов, вывести первый индекс.

Sample Input:

defabc

abcdef

Sample Output:

3

вариант 1 - подготовка к распараллеливанию.

Описание алгоритма.

Задание № 1

Вначале вводятся шаблон, текст и количество потоков. После этого вычисляется символ, который не входит ни в шаблон ни в текст. Для этого вначале в функции *fillHashTable* хеш-таблица заполняется всеми символами, которые встретились либо в тексте либо в шаблоне. Затем в функции *findSymbol* происходит поиск до первого элемента, который не встретился ни в шаблоне ни в тексте. Если очередной символ не лежит в таблице, на этом цикл останавливается, а найденный символ возвращается функцией.

После этого в функции *constructPrefixTable* вычисляется значение префикс функции для каждого элемента строки, состоящей из склеивания строки шаблона, специального символа и строки текста. При этом сама строка в памяти не хранится, а очередной её индекс вычисляется с помощью функции *getIndex*. Это сделано для экономии памяти. Значение префикс функции характеризует длину максимального суффикса строки до текущего символа, совпадающего с префиксом.

После завершения вычисления префикс таблицы вызывается фунция *FindMatching,* которая обходит «склеиную» строку за заданное число потоков. Каждый поток обрабатывает отдельный фрагмент строки. Если значение префикс функции очередного символа равняется длине шаблона, значит мы нашли вхождение. В этом случае из текущего индекса вычитается удвоенная длина шаблона, чтобы найти нужное смещение. Если шаблон ни разу не встретился, после завершения обхода печатается -1.

Задание № 2

Алгоритма практически совпадает с тем, который был реализован для задания № 1, за исключением того, что подстрока *A* ищется не в тексте *B,* а в тексте *B + B,* чтобы не проспустить циклический сдвиг.

Так как построение префикс таблицы происходит за время O(m + n), где m — длина шаблоне, а n — длина текста, а поиск вхождений за время O(n), то сложность по времени обоих алгоритмов — O(m + n).

Хранение префиск таблицы для обоих алгоритмов занимает по памяти O(m + n), так что сложность по памяти обоих алгоритмов также составляет O(m + n).

Описание функций и структур данных.

void findMatching(string &text,string &pattern, vector<int> &prefix_array,int flow) — функция поиска вхождений шаблона в текст. *string &text* — ссылка на строку текста, *string &pattern* — ссылка на строку шаблона, *vector<int> &prefix_array* — ссылка на массив значений префикс функции, *int flow* — количество потоков.

void constructPrefixTable(string &text,string &pattern,vector<int> &prefix_array,char symbol) — функция для формирования таблицы из значений префикс-функции. s*tring &text* —ссылка на строку текста, *string &pattern* — ссылка на строку шаблона, *vector<int> &prefix_array* — ссылка на массив значений префикс функции, *char symbol* — специальный символ.

char findSymbol(map<char,bool > &hash_table) — функция для нахождения специального символа. Функция возвращает символ. *map<char,bool > &hash_table* — ссылка на хэш-таблицу, состоящюю из символов, которые встретились в тексте или в шаблоне.

void fillHashTable(map<char,bool> &hash_table,string &text) — функция для заполнения хэш-таблицы. *map<char,bool> &hash_table* — ссылка на хэш-таблицу, *string &text* — ссылка на текст.

char getIndex(string &text,string &pattern,int index,char symbol) — функция, возвращающая по индексу элемент из склеинной строки. *tring &text* —ссылка на строку текста, *string &pattern* — ссылка на строку шаблона, *int index* — значение индекса, *char symbol* — специальный символ.

string text — строка текста

string pattern — строка шаблона

vector<int> &prefix_array — массив значений префикс-функции

map<char,bool > &hash_table — хэш-таблица, которая хранит все символы, которые встретились в тексте или в шаблоне.

Тестирование.

Программа1 была протестирована на следующих исходных данных:

Входные данные	Результат
ab abab 2	Обработка 1 ого потока Текущая последовательность: a Текущая последовательность: ab Найденно новое вхождение с позиции 0 Обработка 2 ого потока Текущая последовательность: a Текущая последовательность: ab Найденно новое вхождение с позиции 2 0,2
TACG GT 1	-1
ATAT GATATATGCATATACTT 3	Текущая последовательность: Текущая последовательность: A Текущая последовательность: AT Текущая последовательность: ATA Текущая последовательность: ATAT Найденно новое вхождение с позиции 1 Обработка 2 ого потока Текущая последовательность: ATA Текущая последовательность: ATAT Найденно новое вхождение с позиции 3 Текущая последовательность: Текущая последовательность: Текущая последовательность: A Обработка 3 ого потока Текущая последовательность: AT Текущая последовательность: ATA Текущая последовательность: ATAT Найденно новое вхождение с позиции 9 Текущая последовательность: ATA Текущая последовательность: 1,3,9
abdefab a 4	Обработка 1 ого потока Текущая последовательность: a Найденно новое вхождение с позиции 0 Обработка 2 ого потока Текущая последовательность:

	Текущая последовательность: Обработка 3 ого потока Текущая последовательность: Текущая последовательность: Обработка 4 ого потока Текущая последовательность: a Найденно новое вхождение с позиции 5 Текущая последовательность: 0,5
ab abdabbaab	Текущая последовательность: a Текущая последовательность: ab Найденно новое вхождение с позиции 0 Текущая последовательность: Текущая последовательность: a Обработка 2 ого потока Текущая последовательность: ab Найденно новое вхождение с позиции 3 Текущая последовательность: Текущая последовательность: a Текущая последовательность: a Текущая последовательность: ab Найденно новое вхождение с позиции 7 0,3,7

Программа 2 была протестирована на следующих входных данных:

Входные данные	Результат
defabc abcdef 2	Обработка 1 ого потока Текущая последовательность: Текущая последовательность: Текущая последовательность: Текущая последовательность: d Текущая последовательность: de Текущая последовательность: def Обработка 2 ого потока Текущая последовательность: defa Текущая последовательность: defab Текущая последовательность: defabc Нашли циклический сдвиг с индексом 3 На данный момент он минимальный Текущая последовательность: d Текущая последовательность: de Текущая последовательность: def 3
abde ebdb 1	Текущая последовательность: Текущая последовательность: Текущая последовательность: Текущая последовательность:

	Текущая последовательность: Текущая последовательность: Текущая последовательность: Текущая последовательность: -1
AECDE ECDEA 3	Обработка 1 ого потока Текущая последовательность: Текущая последовательность: Текущая последовательность: Обработка 2 ого потока Текущая последовательность: Текущая последовательность: A Текущая последовательность: AE Обработка 3 ого потока Текущая последовательность: AEC Текущая последовательность: AECD Текущая последовательность: AECDE Нашли циклический сдвиг с индексом 1 На данный момент он минимальный Текущая последовательность: A 1
abcdea abcdea 3	0

Выводы.

Были получены навыки работы с алгоритмами поиска подстроки в строке. В частности, был реализован алгоритм Кнута-Мориса-Пратта, для нахождения всех вхождений шаблона в текст, а также алгоритм, проверяющий, является ли строка циклическим сдвигом другой.

Приложения А. Исходный код программы 1

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <math.h>

using std::string;
using std::cout;
using std::cin;
using std::vector;
using std::map;

char getIndex(string &text,string &pattern,int index,char symbol){//функция для получения элемента "строки" по индексу
   if(index < pattern.length())//если элемента строки pattern
      return pattern[index];
   else
   if(index == pattern.length())//если нейтральный символ
      return symbol;
   else
   if(index < pattern.length() + text.length() + 1)//если элемента строки text
      return text[index - (pattern.length() + 1)];
}

void fillHashTable(map<char,bool> &hash_table,string &text){//заполнение хэш таблицы
   for(int i = 0;i < text.length();i++)
      hash_table[text[i]] = true;
}

char findSymbol(map<char,bool > &hash_table){//нахождение подходящего нейтрального символа
   int counter = 1;
   while (1){//смотрим все возможные символы
      if(!hash_table[counter]){
         return (char) counter;//когда нашли, возвращаем его
      } else
         counter++;
   }
}
```

```
void constructPrefixTable(string &text,string &pattern,vector<int> &prefix_array,char symbol){//построение таблицы со значениями префикс функции
   prefix_array[0] = 0;
   int longest_prefix;
    for(int i = 1;i < pattern.length() + text.length() + 1;i++){//проходим по каждмоу элементу строки, склеинной из шаблона, нейтрального элемента и текста
      longest_prefix = prefix_array[i - 1];
      while (1) {
            if (getIndex(text,pattern,i,symbol) == getIndex(text,pattern,longest_prefix,symbol)) {//если совпадает с элементом из префикса, увеличиваем значение на единицу
          prefix_array[i] = longest_prefix + 1;
          break;
        } else {
          if(!longest_prefix)//если не соответсвует никакому префику, переходим к следующему
            break;
          longest_prefix = prefix_array[longest_prefix - 1];
          continue;
        }
      }
   }
}

void findMatching(string &text,string &pattern, vector<int> &prefix_array,int flow) {//функция для нахождения всех вхождений шаблона в текст
   unsigned long  start,end;
   vector<vector<int>> indeces(flow);
   bool flag = false;
   string curr = "";
   for(int k = 0;k < flow;k++) {//разделяем поиск по "потокам"
     cout << "Обработка " << k + 1 << " ого потока\n";
       start = pattern.length() + 1 + k * (ceil(text.length()) / flow);//выделяем начальную позицию для текущего потока
      end = pattern.length() + 1 + (k + 1) * (ceil(text.length()) / flow);//выделяем конечную позицию для текущего потока
      end = end < pattern.length() + text.length() + 1 ? end : pattern.length() + text.length() + 1;//проверка, не выходим ли за границы текста
      for (int i = start; i < end; i++) {
        curr = text.substr(i - pattern.length() - prefix_array[i], prefix_array[i]);
        cout << "Текущая последовательность: " << curr << "\n";
        if (prefix_array[i] == pattern.length()) {//проверяем значение префикс функциис размером шаблона
          cout << "Найденно новое вхождение с позиции " << i - 2 * pattern.length() << "\n";
          if (!flag)
```

```
                flag = true;
            indeces[k].push_back(i - 2 * pattern.length());//добавляем в результирующий вектор
        }
      }
    }
    if(!flag)
      cout << -1;//выодим -1, если не встретили шаблон
    else{
      flag = false;
      for(int i = 0;i < flow;i++){//выводим все вхождения
        for(int j = 0;j < indeces[i].size();j++)
          if(flag)
            cout << "," << indeces[i][j];
          else {
            cout << indeces[i][j];
            flag = true;
          }
      }
    }
}

int main() {
    string pattern;
    string text;
    int flow;
    char symbol;
    cout << "Введите шаблон: ";
    cin >> pattern;
    cout << "Введите текст: ";
    cin >> text;
    cout << "Введите количество потоков: ";
    cin >> flow;
    if(flow > text.size()) {//проверка на корректность количества потоков
      cout << "Потоков не может быть больше, чем длина текста!!!";
      return 0;
    }
    if(flow <= 0){
      cout << "Нельзя вводить отрицательные или нулевые потоки!!!";
      return 0;
    }
    vector<int> prefix_array(pattern.length() + text.length() + 1);
    map<char,bool> hash_table;
```

```
    fillHashTable(hash_table,text);
    symbol = findSymbol(hash_table);
    if(pattern.length() > text.length()) {//если размер шаблона больше размера текста, не обрабатываем
        cout << -1;
        return 0;
    }
    constructPrefixTable(text,pattern,prefix_array,symbol);//вызываем построение префикс таблицы
     findMatching(text,pattern,prefix_array,flow);//вызываем функцию для нахождения всех вхождений щаблона в текст
    return 0;
}
```

Приложение Б. Исходный код программы 2

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <math.h>

using std::string;
using std::cout;
using std::cin;
using std::vector;
using std::map;

char getIndex(string &text,string &pattern,int index,char symbol){///функция для получения элемента "строки" по индексу
   if(index < pattern.length())//если элемента строки pattern
      return pattern[index];
   else
   if(index == pattern.length())//если нейтральный символ
      return symbol;
   else
      if(index < pattern.length() + text.length() + 1)//если элемента строки text
         return text[index - (pattern.length() + 1)];
   else
      return text[index - (2 * pattern.length() + 1)];
}

void fillHashTable(map<char,bool> &hash_table,string &text){//заполнение хэш таблицы
   for(int i = 0;i < text.length();i++)
      hash_table[text[i]] = true;
}

char findSymbol(map<char,bool > &hash_table){//нахождение подходящего нейтрального символа
   int counter = 1;
   while (1){//смотрим все возможные символы
      if(!hash_table[counter]){
         return (char) counter;//когда нашли, возвращаем его
      } else
         counter++;
   }
```

```
}

void constructPrefixTable(string &text,string &pattern,vector<int> &prefix_array,char symbol){//построение таблицы со значениями префикс функции
   prefix_array[0] = 0;
   int longest_prefix;
    for(int i = 1;i < pattern.length() + 2 * text.length() + 1;i++){//проходим по каждмоу элементу строки, склеинной из шаблона, нейтрального элемента и текста
      longest_prefix = prefix_array[i - 1];
      while (1) {
            if (getIndex(text,pattern,i,symbol) == getIndex(text,pattern,longest_prefix,symbol)) {//если совпадает с элементом из префикса, увеличиваем значение на единицу
          prefix_array[i] = longest_prefix + 1;
          break;
        } else {
          if(!longest_prefix)//если не соответсвует никакому префику, переходим к следующему
            break;
          longest_prefix = prefix_array[longest_prefix - 1];
          continue;
        }
      }
   }
}

void findMatching(string &text,string &pattern, vector<int> &prefix_array,int flow) {
   bool flag = false;
   int min = pattern.length() + 1;
   unsigned long start, end;
   string curr;
   for (int k = 0; k < flow; k++) {//разделяем поиск по "потокам"
      cout << "Обработка " << k + 1 << " ого потока\n";
       start = pattern.length() + 1 + k * (ceil(2 * text.length()) / flow);//выделяем начальную позицию для текущего потока
        end = pattern.length() + 1 + (k + 1) * (ceil(2 * text.length()) / flow);//выделяем конечную позицию для текущего потока
       end = end < pattern.length() + 2 * text.length() + 1 ? end : pattern.length() + 2 * text.length() + 1;//проверка, не выходим ли за границы текста
      for (int i = start; i < end; i++) {
        if((i - pattern.length() - 1 - prefix_array[i]) % text.length() + prefix_array[i] < text.length())
          curr = text.substr((i -  pattern.length()  - prefix_array[i]) % text.length(), prefix_array[i]);
        else{
          curr = text.substr(((i - pattern.length() - prefix_array[i]) % text.length()), text.length());
```

```
            curr += text.substr(0,((i - pattern.length() - 1) - text.length() + 1) % text.length());
        }
        cout << "Текущая последовательность: " << curr << "\n";
        if (prefix_array[i] == pattern.length()) {//проверка значения префикс функции с длинной шаблона
            cout << "Нашли циклический сдвиг с индексом " << pattern.length() - (i - 2 * pattern.length()) << "\n";
            if (pattern.length() - (i - 2 * (pattern.length())) < min) {
                min = pattern.length() - (i - 2 * (pattern.length()));
                cout << "На данный момент он минимальный\n";
            }
            if (!flag)
                flag = true;
        }
    }
  }
  if (!flag)
    cout << -1;//если не нашли циклического сдвига, выводим -1
  else
    cout << min;//если нашли. выводим минимальный индекс
}

int main() {
  string pattern;
  string text;
  int flow;
  char symbol;
  cout << "Введите строку 1: ";
  cin >> pattern;
  cout << "Введите строку 2: ";
  cin >> text;
  cout << "Введите количество потоков: ";
  cin >> flow;
  if(flow > text.size()) {//проверка на корректность количества потоков
    cout << "Потоков не может быть больше, чем длина текста!!!";
    return 0;
  }
  if(flow <= 0){
    cout << "Нельзя вводить отрицательные или нулевые потоки!!!";
    return 0;
  }
  if(pattern == text){
    cout << 0;
    return 0;
```

```
    }
    vector<int> prefix_array(pattern.length() + 2 * text.length() + 1);
    map<char,bool> hash_table;
    fillHashTable(hash_table,text);
    symbol = findSymbol(hash_table);
    if(pattern.length() > text.length()) {//если размер шаблона больше размера текста, не обрабатываем
        cout << -1;
        return 0;
    }
    constructPrefixTable(text,pattern,prefix_array,symbol);//вызываем построение префикс таблицы
     findMatching(text,pattern,prefix_array,flow);//вызываем функцию для нахождения всех вхождений щаблона в текст
    return 0;
}
```