Федеральное государственное бюджетное образовательное учреждение

высшего образования «Сибирский государственный университет телекоммуникаций и информатики»

Факультет ИВТ

Кафедра вычислительных систем

**Курсовая работа**

на тему «Полнотекстовый поиск по шаблону»

Вариант 3.4 «Алгоритм Бойера Мура»

Выполнил:студент гр. ИС-341Корытин М. А.

Проверил:Старший преподаватель кафедры ВС

Ефимов А. В.

Новосибирск, 2024

Оглавление

**Тема курсовой работы**3

**Задание на курсовую работу** 3

**Критерии оценки** 3

**Указания к выполнению задания** 3

**Анализ задачи** 3

**Тестовые данные** 8

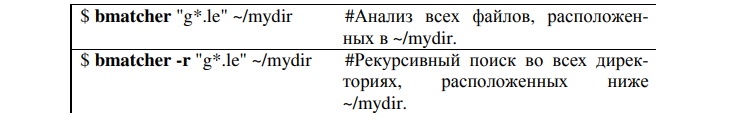
**Листинг программы** 9

**Тема курсовой работы**

Полнотекстовый поиск по шаблону. Алгоритм Бойера-Мура.

**Задание на курсовую работу**

Реализовать программу bmatcher (Boyer–Moore string mATCHER) полнотекстового поиска по шаблону. Шаблон и имя файла (директории), в которой осуществляется поиск, передаются через аргументы командной строки в следующем порядке:



**Критерии оценки**

Оценка «отлично»: разработанная программа обеспечивает поиск текста по шаблону рекурсивно в заданной директории. Под рекурсивным поиском понимается анализ всех текстовых файлов в текущей директории, а также во всех вложенных директориях.

Оценка «хорошо»: разработанная программа не предусматривает поиск по шаблону ИЛИ не способна выполнять рекурсивный поиск в дереве каталогов (поиск только в одном файле).

Оценка «удовлетворительно»: реализован только алгоритм Бойера-Мура с эвристикой стоп-символа.

**Указания к выполнению задания**

Алгоритм Бойера-Мура (БМ) основан на применении эвристики стоп- 38 символа и эвристики безопасного суффикса, которые были подробно рассмотрены в общей информации к данному разделу. В листинге 6 приведен псевдокод алгоритма БМ.

**Анализ задачи**

1. В данной работе нужно реализовать алгоритм Бойера-Мура с применением эвристики стоп-символа и хорошего суффикса. Запуск программы производится с 3 аргументами, сам запуск программы, строка, которую будем искать(pattern), путь к директории. В директории берутся файлы и ищется pattern.
2. Эвристика стоп – символа.   
   Инициализируем массив StopSymbol нулями.   
   Заполняем массив сдвигами, как предполагает алгоритм Бойера – Мура.   
   В случае, если последний элемент встречался и в нем лежит какое – то значение, оставляем его, а если этот элемент массива равен 0, тогда присваиваем максимальный сдвиг элементов.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | void stop\_symbol\_heuristic(char \*pattern, int m, int StopSymbol[ASCII\_TABLE\_SYMBOLS])  {  int i;  int count = 1;  for (i = 0; i < ASCII\_TABLE\_SYMBOLS; i++)  {  StopSymbol[i] = 0;  }  for (i = m - 2; i >= 0; i--)  {  if ((StopSymbol[(int)pattern[i]]) == 0)  {  StopSymbol[(int)pattern[i]] = count;  }  count++;  }  if ((StopSymbol[(int)pattern[m - 1]]) == 0)  {  StopSymbol[(int)pattern[m - 1]] = m - 1;  } |

1. Данная функция ищет итоговые индексы pattern в файлах.   
   Для начала, в m и n сохраняется длина подстроки и текста.   
   Массив сдвига инициализируются нулями.   
   После формируются массив стоп символов, позиций суффикса и сдвига суффикса.   
   Далее, за s обозначим позицию в тексте, то есть индекс.   
   Можно считать это за сдвиг.   
   Позиция, где сейчас находится наш pattern.   
   В while обязательно условие, чтоб не вылезти за границы массива – то есть текста.   
   Дальше проверяем нашу подстроку до того момента, j не станет 0, либо элементы не сойдутся.   
   Если j = 0, то выводим индекс строки и сдвигаем pattern на его длину.   
   Если же, j !=0, тогда выбираем наилучший сдвиг для pattern с помощью двух эвристик.   
   С помощью функции max выбираем наибольшее значение сдвига.

Text: Персональные данные

Pattern: данные

По алгоритму Бойера- Мура таблица сдвигов выглядит так

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Д | А | Н | Н | Ы | | Е | \* |
| 5 | 4 | 2 | 2 | 1 | 6 | | 6 |

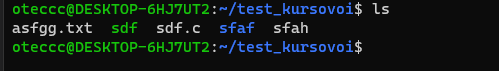
Несходство сразу же в символе н, по таблице символ н = 2, следовательно, в StopSymbolShift кладем эту двойку, и после, будем выбирать между двух эвристик.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | void BM\_MATCHER(char \*text, char \*pattern)  {  int n = strlen(text);  int m = strlen(pattern);  int count = 0;  int StopSymbol[ASCII\_TABLE\_SYMBOLS];  int s = 0;  stop\_symbol\_heuristic(pattern, m, StopSymbol);  while (s <= n - m)  {  int j = m - 1;  while (j >= 0 && pattern[j] == text[s + j])  {  j--;  StopSymbolShift = m - **1**;  }  **else**  {  StopSymbolShift = StopSymbol[(**int**)text[s + j]];  }  **int** suffixShift = shift[j + **1**];  s += max(StopSymbolShift, suffixShift);  }  }  **if** (count == **0**)  printf("No patterns found**\n**");  }  if (j < 0)  {  printf("Pattern found by index %d\n", s);  s++;  count++;  }  else  {  int StopSymbolShift;  if ((StopSymbol[(int)text[s + j]]) == 0)  {  StopSymbolShift = m - 1;  }  else  {  StopSymbolShift = StopSymbol[(int)text[s + j]];  }  s += StopSymbolShift;  }  }  if (count == 0)  printf("No patterns found\n");  }  } |

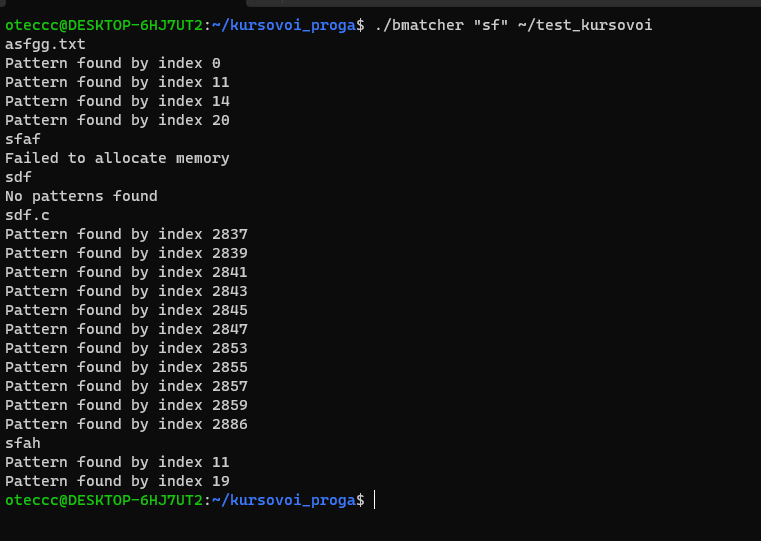
1. Реализация анализа всех файлов, расположенных в директории.   
   Работа с директориями осуществляется через библиотеку #include <dirent.h>.   
   Данная библиотека помогает удобно и быстро работать с ними.   
   Для начала, убедимся, что пользователь подал верную строку, а иначе выведем ошибку и предоставим правильный формат ввода.   
   Сохраним все аргументы и откроем директорию, в цикле while будем идти, пока есть файлы в этой директории.   
   В условии ветвления проверяем на точки имя файла, так как в linux эти два файла создаются автоматичекси с созданием директории.   
   Откроем файл и выделим память для буфера динамически.   
   Если end\_file будем равняться 0, то скорее всего мы зашли в директорию, и память не выделится, что нам и нужно, для того, чтоб учесть этот момент.   
   После считываем все данные в буфер и используя функцию BM\_MATCHER выведем все вхождения pattern в файл.

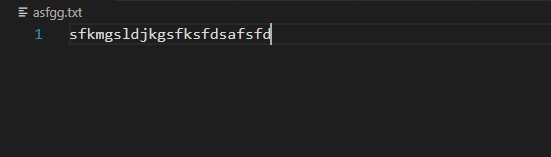
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65 | int main(int argc, char \*argv[])  {  if (argc == 3)  {  char \*string = argv[1];  char \*p = argv[2];  char \*p\_new = strcat(p, "/");  DIR \*dp = opendir(p);  if (!dp)  {  printf("Failed to open directory\n");  return 1;  }  struct dirent \*de;  while ((de = readdir(dp)))  {  if ((strcmp(de->d\_name, ".") != 0) && ((strcmp(de->d\_name, "..")) != 0))  {  char str\_temp[100];  strcpy(str\_temp, p\_new);  char \*str\_path = strcat(str\_temp, de->d\_name);  printf("%s\n", de->d\_name);  FILE \*sf = fopen(str\_path, "r");  strcpy(str\_temp, p\_new);  if (!sf)  {  printf("Failed to open file: %s\n", str\_path);  continue;  }  fseek(sf, 0, SEEK\_END);  int end\_file = ftell(sf);  char \*buf = malloc(end\_file \* sizeof(char));  if (!buf)  {  printf("Failed to allocate memory\n");  fclose(sf);  continue;  }  fseek(sf, 0, SEEK\_SET);  fread(buf, sizeof(char), end\_file, sf);  BM\_MATCHER(buf, string);  fclose(sf);  free(buf);  }  else  {  continue;  }  }  closedir(dp);  }  else  {  printf("Usage:\n./bmatcher \"<str>\" <pwd>\n");  }  return 0;  } |
|  |  |

**Тестовые данные**



Тестовая директория для проверки работа программы.





Проверка нескольких pattern из файла asfgg.txt. По 0 и 11 индексу и лежит подстрока “sf'.

**Листинг программы**

bmatcher.c

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146 | #include <stdio.h>  #include <string.h>  #include <stdlib.h>  #include <dirent.h>  #define ASCII\_TABLE\_SYMBOLS 256  int max(int a, int b)  {  if (a > b)  return a;  else  return b;  }  void stop\_symbol\_heuristic(char \*pattern, int m, int StopSymbol[ASCII\_TABLE\_SYMBOLS])  {  int i;  int count = 1;  for (i = 0; i < ASCII\_TABLE\_SYMBOLS; i++)  {  StopSymbol[i] = 0;  }  for (i = m - 2; i >= 0; i--)  {  if ((StopSymbol[(int)pattern[i]]) == 0)  {  StopSymbol[(int)pattern[i]] = count;  }  count++;  }  if ((StopSymbol[(int)pattern[m - 1]]) == 0)  {  StopSymbol[(int)pattern[m - 1]] = m - 1;  }  }  void BM\_MATCHER(char \*text, char \*pattern)  {  int n = strlen(text);  int m = strlen(pattern);  int count = 0;  int StopSymbol[ASCII\_TABLE\_SYMBOLS];  int s = 0;  stop\_symbol\_heuristic(pattern, m, StopSymbol);  while (s <= n - m)  {  int j = m - 1;  while (j >= 0 && pattern[j] == text[s + j])  {  j--;  }  if (j < 0)  {  printf("Pattern found by index %d\n", s);  s++;  count++;  }  else  {  int StopSymbolShift;  if ((StopSymbol[(int)text[s + j]]) == 0)  {  StopSymbolShift = m - 1;  }  else  {  StopSymbolShift = StopSymbol[(int)text[s + j]];  }  s += StopSymbolShift;  }  }  if (count == 0)  printf("No patterns found\n");  }  int main(int argc, char \*argv[])  {  if (argc == 3)  {  char \*string = argv[1];  char \*p = argv[2];  char \*p\_new = strcat(p, "/");  DIR \*dp = opendir(p);  if (!dp)  {  printf("Failed to open directory\n");  return 1;  }  struct dirent \*de;  while ((de = readdir(dp)))  {  if ((strcmp(de->d\_name, ".") != 0) && ((strcmp(de->d\_name, "..")) != 0))  {  char str\_temp[100];  strcpy(str\_temp, p\_new);  char \*str\_path = strcat(str\_temp, de->d\_name);  printf("%s\n", de->d\_name);  FILE \*sf = fopen(str\_path, "r");  if (!sf)  {  printf("Failed to open file: %s\n", str\_path);  continue;  }  fseek(sf, 0, SEEK\_END);  int end\_file = ftell(sf);  char \*buf = malloc(end\_file \* sizeof(char));  if (!buf)  {  printf("Failed to allocate memory\n");  fclose(sf);  continue;  }  fseek(sf, 0, SEEK\_SET);  fread(buf, sizeof(char), end\_file, sf);  BM\_MATCHER(buf, string);  fclose(sf);  free(buf);  }  else  {  continue;  }  }  closedir(dp);  }  else  {  printf("Usage:\n./bmatcher \"<str>\" <pwd>\n");  }  return 0;  } |