0.1 Требования к программному обеспечению

Минимальные системные требования: PC с операционной системой Windows версии XP/Vista/7/8/10. Требуются устройства ввода: клавиатура, мышь.

0.2 Средства реализации

Для выполнения работы был выбран язык программирования Python ввиду его простоты.

0.3 Интерфейс

Интерфейс из себя представляет простую консоль, где пользователю ничего делать ненужно.

0.4 Листинг

В данном подразделе приведены листинги программ.

1. алгоритм Кнута - Мориса - Прата
2. алгоритм Бойера - Мура

lstname

```
\label{listing:kmp}
\caption{
                -                -                }
  def prefix(s):
      v = [0] * len(s)
      for i in range(1, len(s)):
          k = v[i - 1]
          while k > 0 and s[k] != s[i]:
              k = v[k - 1]
          if s[k] == s[i]:
              k = k + 1
          v[i] = k
      return v

  def kmp(s, t):
      index = -1
      f = prefix(s)
      k = 0
      for i in range(len(t)):
          while k > 0 and s[k] != t[i]:
              k = f[k - 1]
          if s[k] == t[i]:
              k = k + 1
          if k == len(s):
              index = i - len(s) + 1
```

```
25                  break
26          return index
```

lstname

```
1       def badCharHeuristic(string, size):
2           badChar = [-1] * 256
3
4           for i in range(size):
5               badChar[ord(string[i])] = i
6
7           return badChar
8
9
10      def search(txt, pat):
11          """
12          A pattern searching function that uses Bad
                Character
13          Heuristic of Boyer Moore Algorithm
14          """
15          m = len(pat)
16          n = len(txt)
17
18          # create the bad character list by calling
19          # the preprocessing function badCharHeuristic()
20          # for given pattern
21          badChar = badCharHeuristic(pat, m)
22
23          # s is shift of the pattern with respect to text
24          s = 0
25          while s <= n - m:
26              j = m - 1
27
28              # Keep reducing index j of pattern while
29              # characters of pattern and text are matching
30              # at this shift s
31              while j >= 0 and pat[j] == txt[s + j]:
32                  j -= 1
33
34              # If the pattern is present at current shift,
35              # then index j will become -1 after the
                    above loop
36              if j < 0:
37                  # print("Pattern occur at shift =
                        {}".format(s))
38                  return s  # Return only first entry
39
40                  '''
41                  Shift the pattern so that the next
                        character in text
```

```
42                  aligns with the last occurrence of it in
                        pattern.
43                  The condition s+m < n is necessary for
                        the case when
44                  pattern occurs at the end of text
45
46                  s += (m - badChar[ord(txt[s + m])] if s
                        + m < n else 1)
47                  '''
48              else:
49                  '''
50                  Shift the pattern so that the bad
                        character in text
51                  aligns with the last occurrence of it in
                        pattern. The
52                  max function is used to make sure that
                        we get a positive
53                  shift. We may get a negative shift if
                        the last occurrence
54                  of bad character in pattern is on the
                        right side of the
55                  current character.
56                  '''
57                  s += max(1, j - badChar[ord(txt[s + j])])
58
59          return -1
```