**LAPORAN TUGAS KECIL 4**

**STRATEGI ALGORITMA 2020**

Oleh: Naufal Arfananda Ghifari/13518096 (K-3)

1. Materi

Pencarian *pattern* merupakan salah satu algoritma penting dalam komputasi. Saat mencari sebuah string pada sebuah file, algoritma string matching akan menampilkan hasil dari pencarian tersebut. Pada persoalan kali ini, penulis mengimplementasikan algoritma pencarian dengan algoritma **Knuth Morris Pratt(KMP)** dan  **Boyer Moore**

1. **Algoritma KMP**

Algoritma KMP merupakan salah satu algoritma yang lebih baik daripada metode *Brute Force.* Algoritma ini memanfaatkan degenerasi(pola memiliki sub-pola yang sama muncul lebih dari sekali dalam pola). Ide dasarnya adalah setiap kali mendeteksi ketidakcocokan setelah beberapa kali kecocokan, tidak dilakukan pengembalian lagi dari awal karena kita sudah mengetahui beberapa karakter di jendela berikutnya.

Sebelum melakukan pencarian, terlebih dahulu akan dicari nilai (*longest proper prefix which is also suffix*) lps dari pattern yang akan memudahkan dalam pencarian.

Contoh beberapa lps :

|  |
| --- |
| pattern “AAAA”,  lps[] = [0, 1, 2, 3]  pattern “ABCDE”,  lps[] = [0, 0, 0, 0, 0]  pattern “AABAACAABAA”,  lps[] = [0, 1, 0, 1, 2, 0, 1, 2, 3, 4, 5] |

**Algoritma pencarian**

1. Kita mulai dengan membandingkann pat [j] dengan j = 0 dengan karakter teks saat ini.
2. Kemudian cocokkan karakter txt [i] dan pat [j] dan terus menambah i dan j apabila pat[j] dan txt [i] terus mengalami kecocokkan
3. Ketika kita melihat ketidakcocokan:

* Kita tahu nilai karakter pat [0..j-1] cocok dengan txt [i-j… i-1] (Perhatikan bahwa j dimulai dengan 0 dan bertambah hanya ketika ada kecocokan).
* Kita juga tahu (dari definisi di atas) bahwa lps [j-1] adalah jumlah karakter pat [0 ... j-1] yang keduanya merupakan awalan dan akhiran yang tepat.
* Dari dua poin di atas, kita dapat menyimpulkan bahwa kita tidak perlu mencocokkan karakter lps [j-1] ini dengan txt [i-j… i-1] karena kita tahu bahwa karakter ini akan tetap cocok.

1. **Algortima Boyer Moore**

Berikut pendekatan yang digunakan pada algoritma ini:

1. Karakter buruk Heuristik
2. Suffix baik Heuristik

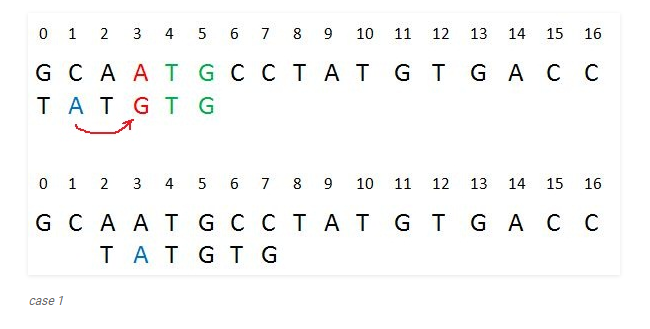
Gagasan heuristik karakter buruk itu sederhana. Karakter teks yang tidak cocok dengan karakter pola saat ini disebut Karakter Buruk. Setelah ketidaksesuaian, kita menggeser polanya hingga -

1) Ketidakcocokan menjadi kecocokan

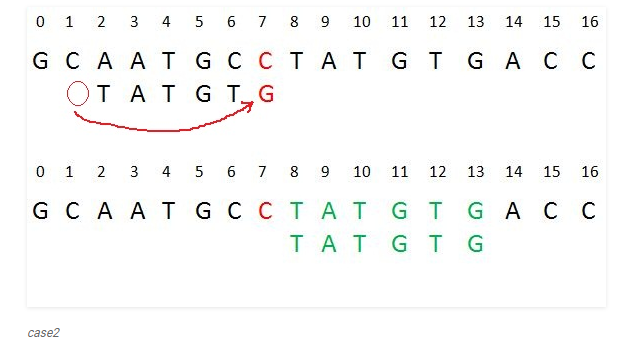
2) Pola P bergerak melewati karakter yang tidak cocok.

Ilustrasi gambar yang menggambarkan :

Case 1 - Kita akan mencari posisi kemunculan terakhir dari karakter yang tidak cocok dalam pola dan jika karakter yang tidak cocok ada dalam pola, maka kami akan menggeser pola tersebut sehingga sejalan dengan karakter yang tidak cocok dalam teks T.



Case 2 – Kita akan mencari posisi kemunculan terakhir karakter yang tidak cocok dalam pola dan jika karakter tidak ada, kita akan menggeser pola melewati karakter yang tidak cocok.



1. **Pengenalan Regular Expression**

Reguar Expression yang biasa disebut Regex merupakan salah satu library yang sering digunakan dalam proses pencarian. Library ini tersedia di berbagai macam Bahasa.

Dalam python library ini dapat digunakan dengan mencantumkan “import re” pada program

1. **Implementasi**
2. **KMP**

|  |
| --- |
| def KMPSearch(pat, txt):      M = len(pat)      N = len(txt)      lps = findArrayLPS(pat) *#cai lpsnya terlebih dahulu*      j = 0 *#index penggeseran pat*      i = 0 *# index penggeseran txt*      while i < N:          if pat[j].lower() == txt[i].lower():              if(j==M-1):*#sudah menemukan yang sesuai*                  return True              i += 1              j += 1          elif(j>0):              j = lps[j-1]          else:              i +=1      return False    def findArrayLPS(pat):  *#res adalah array yang akan menjadi hasil*      length = len(pat)      res = [0 for i in range (length)]      val = 0      i = 1      while( i< length ):          if(pat[i].lower() == pat[val].lower()):*#tidak mempedulinan case*              val += 1              res[i] = val              i += 1          elif val>0:  *#lihat kecocokan yang sebelumnya berdasarkan nilai lps*              val = res[val-1]          else:              i+=1      return res |

1. **Boyer Moore**

|  |
| --- |
| def BMSearch(pat, txt):      M = len(pat)      N = len(txt)      last = build(pat)      i = M-1      j = M-1      while(i<=N-1):          if(pat[j].lower() == txt[i].lower()):*#cek kesesuaian pattern dimulai ekornya*              if(j==0):                  return i              j -=1              i -=1          else :*#jump karakter khas BM*              lo = last[ord(txt[i].lower())]              i = i+M -min(j,i+lo)              j = M-1      return False  def build(pat):    *# Inisialisasi dengan -1*      length = 256      res = [-1 for i in range (length)]    *# Mengisi posisi tiap char yang terakhir ditemukan*      for i in range(len(pat)):          res[ord(pat[i].lower())] = i        return res |

1. **Screen Shot Input Output**

**Daftar Pustaka**

<http://informatika.stei.itb.ac.id/~rinaldi.munir/>

<https://www.geeksforgeeks.org/>