PERCOBAAN III

ALGORITMA *BOYER MOORE*

I. TUJUAN

a. Praktikan mampu memahami algoritma *boyer moore*.
b. Praktikan mampu mengimplementasikan algoritma *boyer moore*.

II. ALAT DAN BAHAN

a. Laptop
b. Modul
c. *PyCharm*
d. *Python*

III. TEORI DASAR

Algoritma *Boyer-Moore* adalah salah satu algoritma untuk mencari suatu *string* di dalam teks, dibuat oleh R.M Boyer dan J.S Moore. Algoritma *Boyer-Moore* melakukan perbandingan dimulai dari kanan ke kiri, tetapi pergeseran *window* tetap dari kiri ke kanan. Jika terjadi kecocokkan maka dilakukan perbandingan karakter teks dan karakter pola yang sebelumnya, yaitu dengan sama-sama mengurangi indeks teks dan pola masing-masing sebanyak satu. Dengan mengunakan algoritma ini, secara rata-rata proses pencarian akan menjadi lebih cepat jika dibandingkan dengan algoritma lainnya. Alasan melakukan pencocokkan dari kanan (posisi terakhir *string* yang dicari) ditunjukkan dalam contoh berikut:

M	A	K	A	N		T	O	M	A	T
T	O	M	A	T						

Pada contoh di atas, dengan melakukan perbandingan dari posisi paling akhir string dapat dilihat bahwa karakter ‘n’ pada string “makan” tidak cocok dengan karakter “t” pada string “tomat” yang dicari, dan karakter “n” tidak pernah ada dalam string “tomat”

yang dicari sehingga string “tomat” dapat digeser melewati string “makan”, sehingga posisinya seperti berikut.

M	A	K	A	N		T	O	M	A	T
					T	O	M	A	T	

Dalam contoh terlihat bahwa algoritma Boyer-Moore memiliki loncatan karakter yang besar sehingga mempercepat pencarian string karena dengan hanya memeriksa sedikit karakter, dapat langsung diketahui bahwa string yang dicari tidak ditemukan dan dapat digeser ke posisi berikutnya. Ide di balik algoritma ini adalah dengan memulai pencocokan karakter dari kanan dan bukan dari kiri, maka akan lebih banyak informasi yang didapat

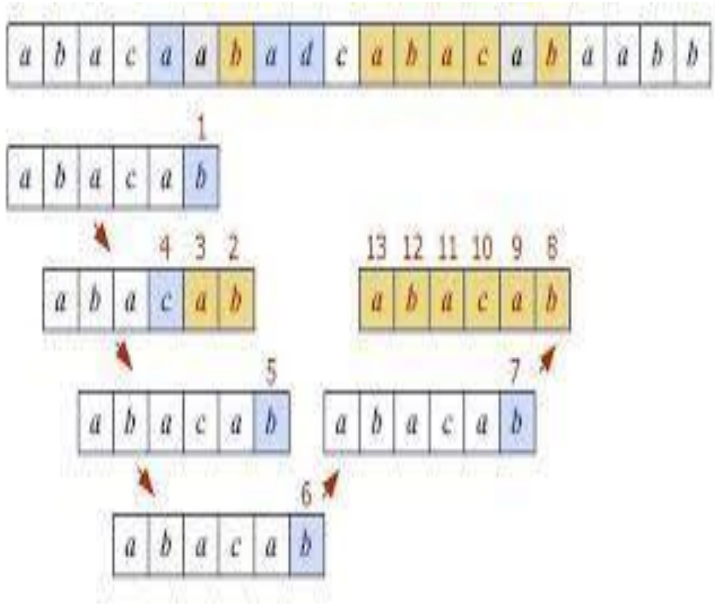
a. Bad-Character Shift (Occurance Heuristic)

Bad-Character Shift (Occurance Heuristic) Tabel Occurrence Heuristic sering disebut juga Bad-Character Shift, dimana pergeserannya dilakukan berdasarkan karakter apa yang menyebabkan tidak cocok dan seberapa jauh karakter tersebut dari karakter paling akhir.

b. Good-Suffix Shift

Good-Suffix Shift (Match Heuristic) Tabel Match Heuristic sering disebut juga Good-Suffix Shift, dimana pergeserannya dilakukan berdasarkan posisi ketidakcocokkan karakter yang terjadi. Maksudnya untuk menghitung tabel Match Heuristic, perlu diketahui pada posisi keberapa terjadi ketidakcocokkan.

c. String Matching

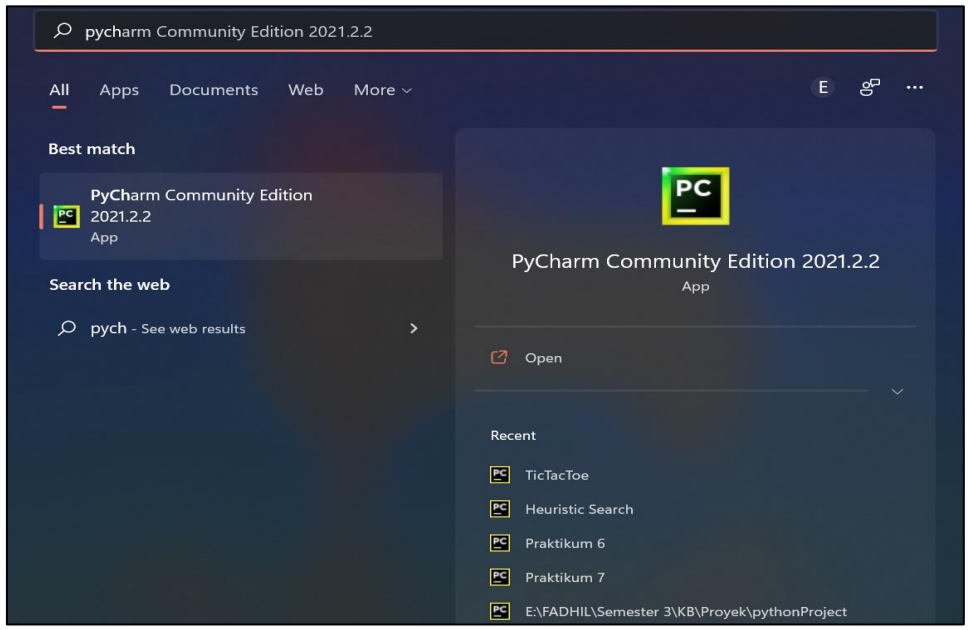


Pencocokan string merupakan bagian penting dari sebuah proses pencarian string (string searching) dalam sebuah dokumen (Saragih, 2013). Hasil dari pencarian sebuah string dalam dokumen tergantung dari teknik pencocokan string yang digunakan. Pencocokan string (string matching) secara garis besar dapat dibedakan menjadi dua yaitu sebagai berikut :

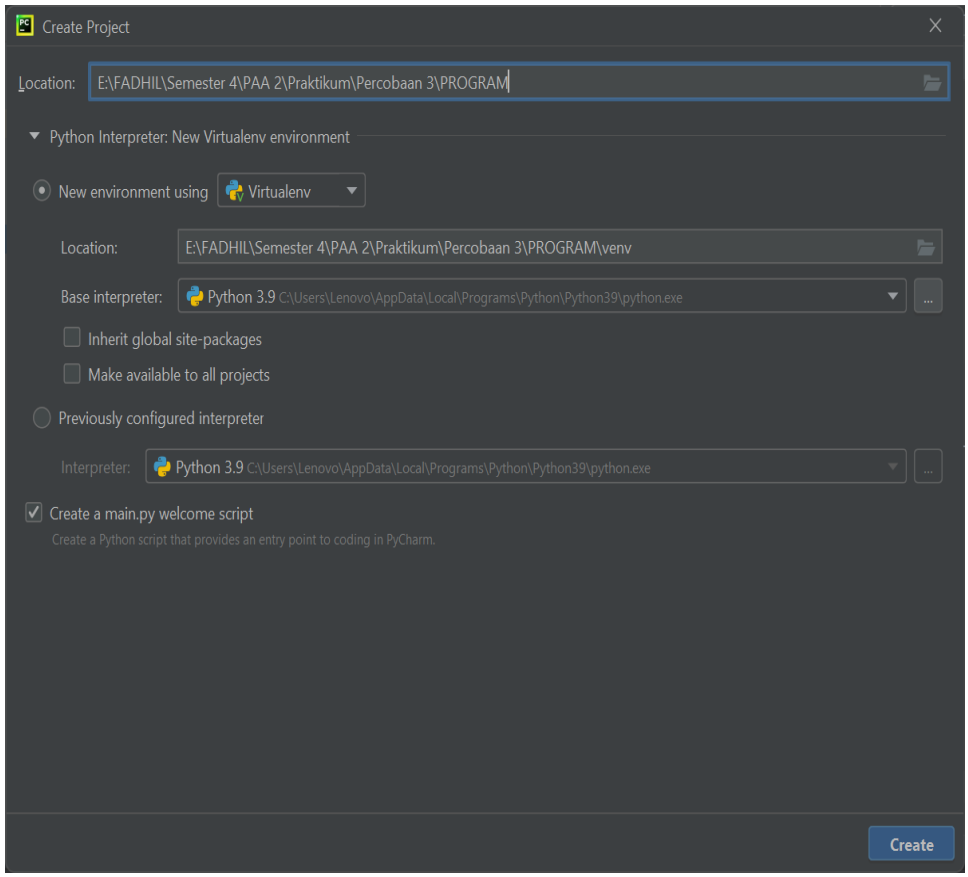
a. Exact String Matching merupakan pencocokan string secara tepat dengan susunan karakter dalam string yang dicocokkan memiliki jumlah maupun urutan karakter dalam string yang sama. Contoh : kata open akan menunjukkan kecocokan hanya dengan kata open.
b. Inexact String Matching atau Fuzzy string matching, merupakan pencocokan string secara samar, dimana string yang dicocokkan memiliki kemiripan dimana keduanya memiliki susunan karakter yang berbeda tetapi string tersebut memiliki kemiripan baik kemiripan tekstual/penulisan (approximate string matching) atau kemiripan ucapan (phonetic string matching).

IV. LANGKAH KERJA

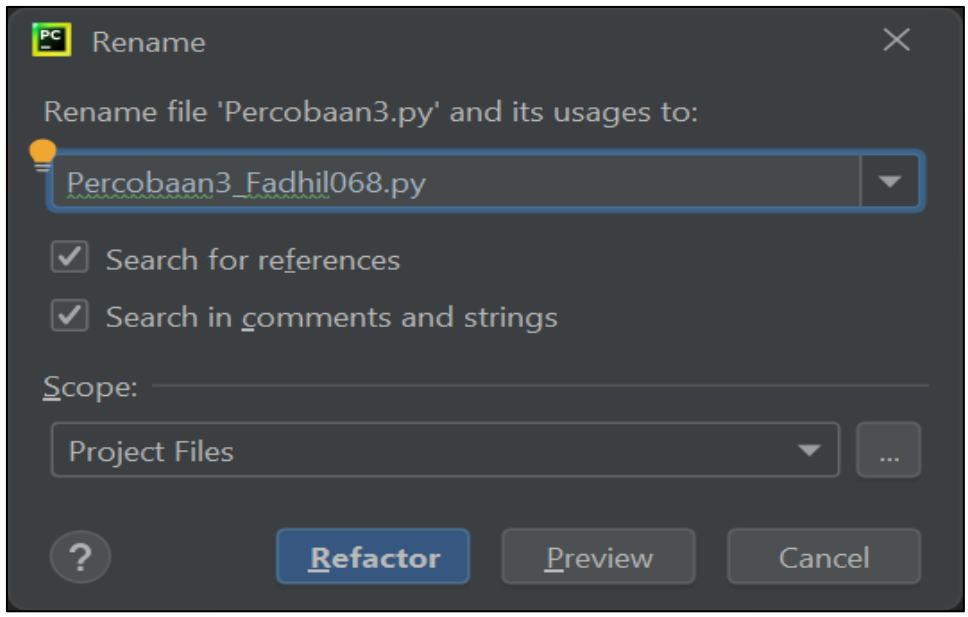
a. Pertama buka aplikasi *pycharm*



b. Buat *project* baru kemudian *save*.



c. Beri nama *file* "*Percobaan3_Fadhil068.py*"



d. masukkan kode program seperti berikut.

```python
# Muhammad Fadhil_F55120068

def get_BMBC(pattern):
    BMBC = dict()
    for i in range (len(pattern) - 1):
        char = pattern[i]
        BMBC[char] = i + 1
    return BMBC

def get_BMGS(pattern):
    BMGS = dict()

    BMGS[''] = 0

    for i in range(len(pattern)):

        GS = pattern[len(pattern) - i - 1:]

        for j in range(len(pattern) - i - 1):

            NGS = pattern[j:j + i + 1]

            if GS == NGS:
                BMGS[GS] = len(pattern) - j - i - 1

    return BMGS

```

```python
def BM(string, pattern, BMBC, BMGS):
    i = 0
    j = len(pattern)
    while i < len(string):
        while (j > 0):
            a = string[i + j - 1:i + len(pattern)]
            b = pattern[j - 1:]

            if a == b:
                j = j - 1
            else:
                i = i + max(BMGS.setdefault(b[1:], len(pattern)), j - BMBC.setdefault(string[i + j - 1], 0))
                j = len(pattern)
            if j == 0:
                return i

    return None

if __name__ == '__main__':
    string = 'Muhammad Fadhil F55120068'
    pattern = 'Fadhil'
    BMBC = get_BMBC(pattern=pattern)
    BMGS = get_BMGS(pattern=pattern)

    x = BM(string=string, pattern=pattern, BMBC=BMBC, BMGS=BMGS)

    print(string[x:])
```

e. Jalankan program, dengan cara menekan tombol *Run* > pada pojok kanan atas.

V. HASIL PERCOBAAN

```
"E:\FADHIL\Semester 4\PAA 2\Praktikum\Percobaan 3\PROGRAM\venv\Scripts\python.exe" "E:\FADHIL\Semester 4\PAA 2\P
Fadhil

Process finished with exit code 0
```

VI. ANALISIS

Pada percobaan diatas, dianalisa bahwa Algoritma *Boyer-Moore* ini dianggap sebagai algoritma yang paling efisien pada aplikasi umum. Algoritma *Boyer-Moore* akan melakukan pencocokan kata kunci dari nilai yang anda yang dimulai dari sebelah kanan (*string matching)* atau disebut pencocokan *string*. Disini terdapat variabel *"def get_BMBC(pattern):"* untuk melakukan pergesaran pada *pattern*, variabel tersebut memiliki fungsi yang sama dengan variabel "*def get_BMGS(pattern)*", variabel *"def BM(string, pattern, BMBC, BMGS): i = 0 j = len(pattern)"* merupakan algoritma yang berfungsi untuk melakukan pencarian *string.* Variabel *" while i < len(string): while (j > 0): a = string[i + j -1:i + len(pattern)] b = pattern[j - 1:]"* sebagai perulangan yang akan melakukan pencocokan *string* pada *pattern*. Variabel *"if a == b: j = j – 1"* merupakan perulangan dengan syarat tertentu untuk mengecek pencocokan *pattern* dan *string*. Jika syarat pencocokan tidak terpenuhi maka fungsi *BMGC* dan *BMGS* akan melakukan kondisi perulangan yang lain dengan variabel *"else: I = i + max(BMGS.setdefault(b[1:], len(pattern)), j - BMBC.setdefault(string[i + j - 1], 0)) j = len(pattern)*. Variabel *"if j == 0: return I"* untuk menyelesaiakn program apabila pencocokan telah selesai. Pada fungsi utama kita akan melakukan pencarian *string* dengan *"string = muhammad fadhil f55120068"* yang akan dilakukan pencocokan kata kunci fadhil, kata kunci dimasukkan pada variabel *pattern* dengan variabel *"pattern = fadhil"*. *Pattern* tersebut akan melakukan *string matching.*

VII. KESIMPULAN

Pada percobaan ini dapat disimpulkan bahwa pada algoritma *Boyer-Moore* merupakan algoritma yang paling efisien pada aplikasi umum. Algoritma *Boyer-Moore* merupakan algoritma yang akan melakukan pencocokan kata kunci pada sebuah nilai yang ditentukan, pencocokan ini disebut dengan (*string matching*) pencocokan *string* oleh *pattern* yang akan dimulai dari arah kanan. Sebelum melakukan *string matching* terdapat bagian yang sangat penting yaitu *string searching*. *String searching* dibedakan menjadi dua yaitu *Exact String Matching* dan *Inexatc String Matching*