def Naiwny(text, pattern):

Start

Zainicjalizuj zmienne pomocnicze:
n = len(text)
m = len(pattern)
i = 0
output = [" "] * n
index = []

Czy i <= n-m

Nie

Tak

i++

Czy text[i:i+m] = pattern?

Nie

Tak

output[i] = "^"
index.append(i)

Wypisz output i
"Indeksy poczatkowe
wzorcow:" + index

Koniec

def Karpa-Rabina(text, pattern):

Start

Zainicjalizuj zmienne pomocnicze:
n = len(text)
m = len(pattern)
d = 256
p = 101
hash_pattern = 0
hash_text = 0
i, j = 0

Czy i < m?

Nie / Tak

i++

hash_pattern = (hash_pattern * d + ord(pattern[i])) % p

Czy j < m?

Nie / Tak

j++

hash_text = (hash_text * d + ord(text[i])) % p

index = []
output = [" "] * n
i = 0

Czy i < (n-m + 1)?

Nie / Tak

Wypisz output i
"Indeksy poczatkowe
wzorcow:" + index

Koniec

i++

Czy hash_text = hash_pattern?

Nie / Tak

Czy text[i:i+m] = pattern?

Nie / Tak

index.append(i)
output[i] = "^"

Czy i < n - m?

Nie / Tak

hash_text = (d * (hash_text - ord(text[i]) * pow(d, m - 1, p)) + ord(text[i + m])) % p

Czy hash_text < 0?

Nie / Tak

hash_text += p

def Boyer-Moore(text, pattern):

```
Start
```

Zainicjalizuj zmienne pomocnicze:
n = len(text)
m = len(pattern)
i = m - 1
index = []
output = [" "] * n

Czy i < n?

Nie → Tak

Wypisz output i
"Indeksy poczatkowe
wzorcow:" + index

Koniec

j = m - 1

Czy text[i] =
pattern[j]?

Nie → Tak

i += m - min(j, 1+
in_last(text[i]))

j = j - 1
i = i - 1

Czy j = -1?

Nie → Tak

index.append(i+1)
output[i+1] = "^"
i += m - min(j, 1+
in_last(text[i]))