**LAPORAN TUGAS KECIL 4**

**EKSTRAKSI INFORMASI DARI ARTIKEL BERITA DENGAN ALGORITMA PENCOCOKAN STRING**

**IF2211 Strategi Algoritma**

**­­**

Oleh:

**Daffa Pratama Putra 13518033**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2020**

# **BAB I ALGORITMA PENCOCOKAN STRING**

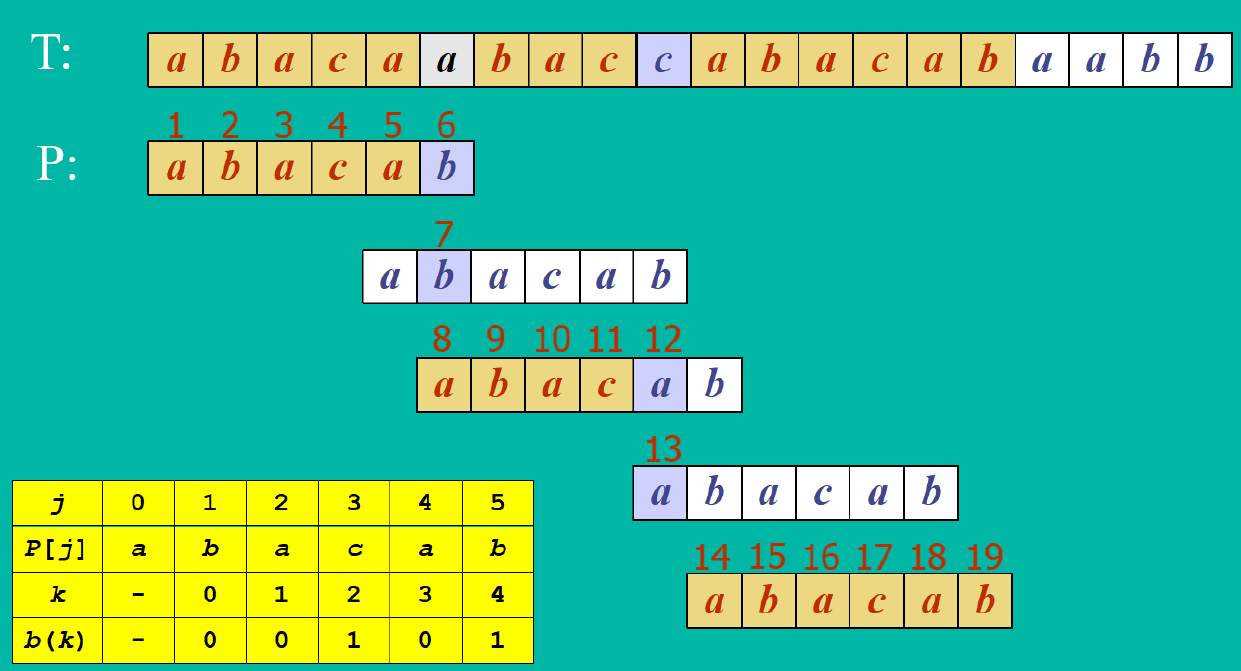
1. **Algoritma Knuth-Morris-Pratt (KMP)**

Algoritma Knuth-Morris-Pratt adalah salah satu algoritma pencocokan *string* yang dinilai lebih baik dari algoritma Brute Force. Algoritma KMP ini dikembangkan oleh salah satu *computer scientist* dari Standford University, Donald E. Knuth pada tahun 1967. Setahun sebelumnya, James H. Morris dan Vaughan R. Pratt juga mengembangkan algoritma yang sama, oleh karena itu nama dari algoritma ini diambil dari nama ketiga penciptanya, yakni Knuth, Morris, dan Pratt.

Pada dasarnya, algoritma ini mirip seperti algoritma Brute Force dalam pencocokan *string*, yaitu melakukan pencocokan dari kiri ke kanan *string*. Namun ada hal yang berbeda. Pada algoritma Brute Force, pergeseran dalam pencocokan *string* dilakukan satu persatu dari kiri ke kanan, baik karakter yang dicocokkan sama atau berbeda. Pergeseran satu persatu dilakukan hingga pencocokan karakter sudah mencapai akhir atau ditemukan *string* yang dicari oleh *pattern* pada *text*. Berbeda dengan algoritma Brute Force, algoritma KMP melakukan pergeseran dalam pencocokan *string* dengan mempertimbangkan *pattern* yang telah dibandingkan. Sehingga pada algoritma KMP dibutuhkan *pre-processing* untuk menentukan jumlah pergeseran yang dilakukan untuk setiap indeks pada *pattern* atau yang disebut dengan fungsi pinggiran.

Fungsi pinggiran atau *failure function* pada algoritma KMP adalah fungsi yang menghitung jumlah pergeseran yang dilakukan dengan menghitung panjang terbesar prefiks dari *pattern,* P[0..k], yang merupakan sufiks dari pattern tersebut, P[1..k], dengan k = (indeks *mismatch* pada pattern) - 1. Dengan demikian, proses pergeseran akan menjadi lebih efisien karena karakter yang sudah dicocokkan pada prefiks tidak dicocokkan kembali. Algoritma KMP menghasilkan kompleksitas waktu O(m+n) dengan O(m) adalah kompleksitas menghitung fungsi pinggiran dan O(n) adalah kompleksitas pencarian *string*.

Berikut adalah contoh pencarian menggunakan algoritma KMP :

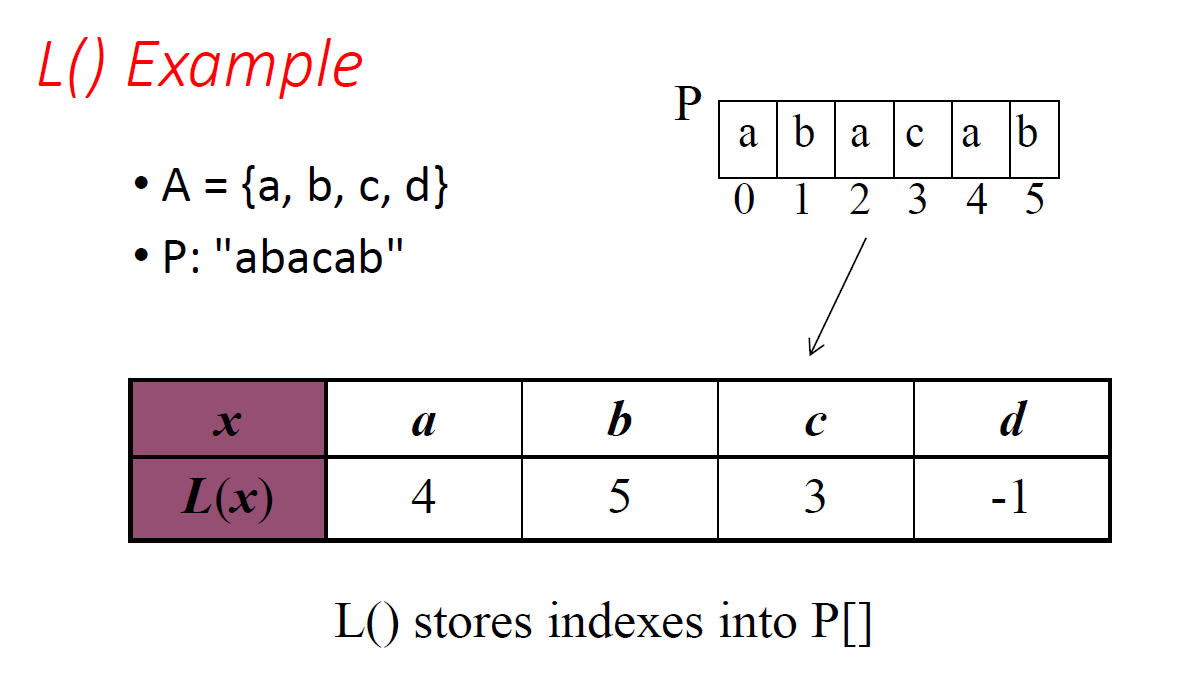


Gambar 1. Pencocokan string dengan algoritma KMP (diambil dari Bahan Kuliah IF2211 Strategi Algoritma, Pencocokan String (String/Pattern Matching)

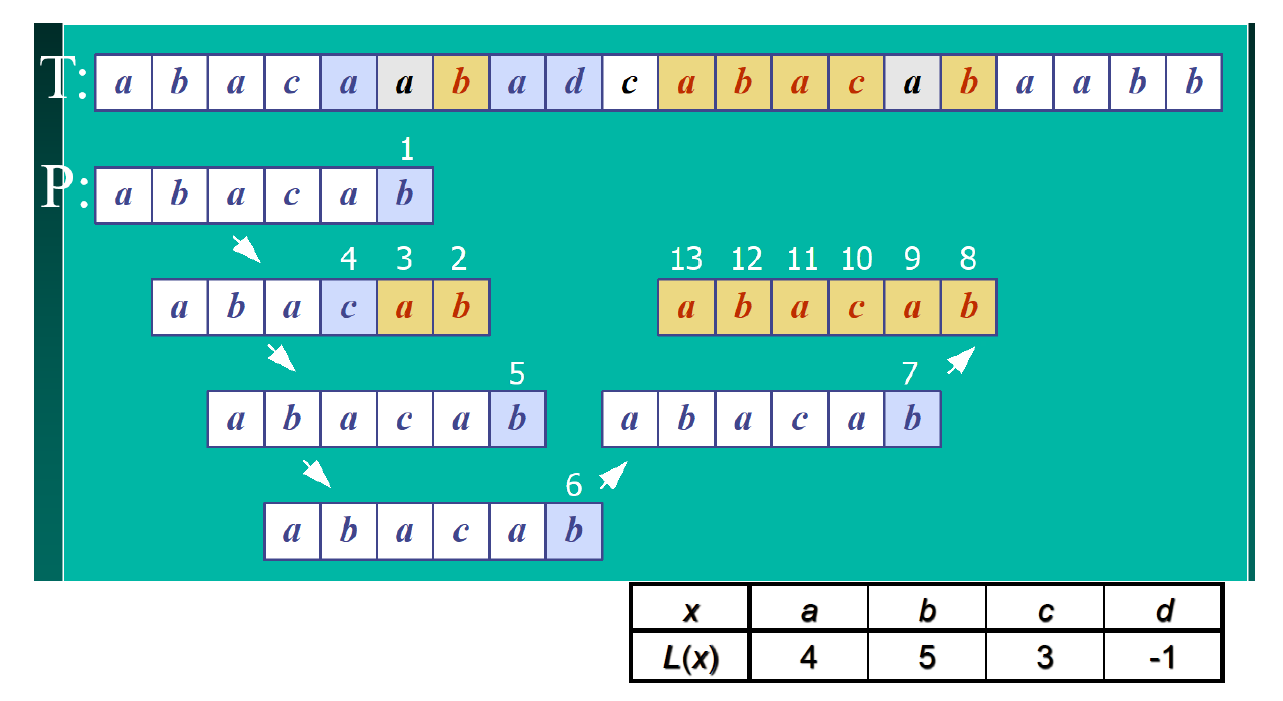
1. **Algoritma Boyer-Moore (BM)**

Algoritma Boyer-Moore adalah salah satu algoritma pencocokan *string* dengan teknik yang cukup unik namun dinilai paling efisien. Algoritma Boyer-Moore menggunakan 2 teknik, yaitu *looking-glass technique* dan *character-jump technique*. Teknik *looking-glass* ini akan mencocokkan *pattern* dengan *text* dengan pengecekan mundur. Pengecekan mundur seperti ini efisien karena dapat memprediksi pergeseran yang akan dilakukan selanjutnya secara optimal. Lalu teknik *character-jump* adalah teknik pergeseran yang dilakukan pada algoritma Boyer-Moore dengan mempertimbangkan lokasi kemunculan karakter yang *mismatch* antara *pattern* dengan *text*. Terdapat tiga kasus dalam teknik *character-jump* ini, yaitu karakter yang *mismatch* pada *text* ditemukan di sebelah kiri karakter *mismatch* pada *pattern*, karakter yang *mismatch* pada *text* ditemukan di sebelah kanan karakter *mismatch* pada *pattern*, dan karakter yang *mismatch* pada *text* tidak terdapat pada *pattern*.

Sama seperti algoritma Knuth-Morris-Pratt, algoritma Boyer-Moore juga membutuhkan *pre-processing* untuk mendapatkan lokasi kemunculan karakter yang ada pada *text*. *Pre-processing* pada algoritma ini adalah digunakan untuk mencari lokasi *last occurance* dari setiap karakter pada *text*. Jika karakter ada pada *pattern*, maka nilai *last occurance*nya adalah indeks terbesar kemunculan karakter tersebut pada *pattern* P, namun jika tidak ditemukan, maka nilai *last occurance*nya adalah -1. Contoh mencari nilai *last occurance* adalah sebagai berikut :



*Last occurance* ini berguna untuk menentukan berapa banyak pergeseran yang akan dilakukan. Misalkan *mismatch* terjadi pada karakter X. Karakter X akan dicek apakah ada pada *pattern* menggunakan *last occurance*. Jika berada di sebelah kiri indeks karakter *mismatch*, maka *pattern* digeser ke kanan sehingga karakter yang *mismatch* tadi dapat benar dicocokkan dengan karakter pada *pattern*. Berikut adalah contoh pencocokan *string* dengan algoritma Boyer-Moore :



1. **Regex**

# **BAB II KODE PROGRAM**

1. **Kode Program**

# **BAB III UJI COBA**