**TUGAS KECIL**

**Implementasi Algoritma Brute Force dalam Penyelesaian Word Search Puzzle**

**LAPORAN**

**Diajukan sebagai salah satu tugas mata kuliah IF2211 Strategi Algoritma pada**

**Semester II**

**Tahun Akademik 2021-2022**

**oleh**

**Owen Christian Wijaya 13520124**

**A picture containing text

Description automatically generated**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG**

**2022**

DAFTAR ISI

[BAB I. ALGORITMA *BRUTE FORCE* 3](#_Toc93856443)

[BAB II. *SOURCE PROGRAM* DALAM BAHASA JAVA 4](#_Toc93856444)

[2.1 CharObj,java 4](#_Toc93856445)

[2.2 Exec.java 5](#_Toc93856446)

[2.3 MainProgram.java 5](#_Toc93856447)

[2.4 Matcher.java 6](#_Toc93856448)

[2.5 Matrix.java 7](#_Toc93856449)

[2.6 SearchWord.java 7](#_Toc93856450)

[BAB III. PENGUJIAN (INPUT/OUTPUT) 8](#_Toc93856451)

[3.1 Pengujian terhadap file small1.txt (ukuran 18 x 16) 8](#_Toc93856452)

[3.2 Pengujian terhadap file small2.txt (ukuran 16 x 14) 9](#_Toc93856453)

[3.3 Pengujian terhadap file small3.txt (ukuran 14 x 12) 10](#_Toc93856454)

[3.4 Pengujian terhadap file medium1.txt (ukuran 20 x 18) 11](#_Toc93856455)

[3.5 Pengujian terhadap file medium2.txt (ukuran 22 x 20) 12](#_Toc93856456)

[3.6 Pengujian terhadap file medium3.txt (ukuran 24 x 22) 13](#_Toc93856457)

[3.7 Pengujian terhadap file large1.txt (ukuran 32 x 30) 14](#_Toc93856458)

[3.8 Pengujian terhadap file large2.txt (ukuran 34 x 32) 15](#_Toc93856459)

[3.9 Pengujian terhadap file large3.txt (ukuran 36 x 34) 16](#_Toc93856460)

[3.10 Hasil Pengujian 19](#_Toc93856461)

[BAB 4. KESIMPULAN 20](#_Toc93856462)

[BAB 5. REFERENSI 20](#_Toc93856463)

# BAB I. ALGORITMA *BRUTE FORCE*

Algoritma *brute force* adalah algoritma yang menggunakan metode-metode yang diimplementasikan secara langsung tanpa mempertimbangkan kapabilitas komputasi atau memori komputer. Algoritma *brute force* diimplementasikan dengan menganalisa deskripsi masalah dan mencoba setiap kemungkinan yang dapat dilakukan hingga ditemukan solusi dari permasalahan yang diinginkan. Algoritma *brute force* mengandalkan pendekatan yang terus terang adanya dan cenderung mengabaikan teknik-teknik lanjutan yang meningkatkan efisiensi program.

Kelebihan dari algoritma *brute force* adalah kemampuannya untuk dapat menyelesaikan hampir semua persoalan yang ada (*widely applicable*). Meskipun tidak seefisien algoritma-algoritma lainnya, algoritma *brute force* cenderung lebih sederhana untuk diimplementasikan, dan karena itu, algoritma *brute force* sering digunakan sebagai basis pembanding dengan algoritma-algoritma lain yang lebih efisien. Algoritma *brute force* menghasilkan algoritma-algoritma yang layak untuk masalah-masalah penting seperti pencarian atau pencocokan *string* dan operasi matriks. Akan tetapi, algoritma *brute force* tidak dapat digunakan di sistem dengan sumber daya terbatas dan mengambil waktu yang lama untuk memproses masukan dengan ukuran besar.

Dalam tugas kecil ini, algoritma *brute force* diimplementasikan dalam pencarian kata-kata yang ada dalam permainan *word search puzzle*. *Puzzle* disimpan dalam bentuk matriks karakter, sementara kata kuncidisimpan ke dalam sebuah *list*. Algoritma *brute force* akan mencari keberadaan sebuah kata dalam *puzzle* dengan pencocokan karakter pertama dari kata kunci yang hendak dicari dengan karakter yang ada di dalam *puzzle* secara sekuensial. Apabila ditemukan karakter yang sesuai dengan karakter pertama kata kunci, maka pencarian dilanjutkan dengan mencocokkan karakter-karakter lain dengan karakter di kata kunci.

Pencocokan dilakukan terhadap delapan arah: ke arah kanan, ke arah kiri, ke arah atas, ke arah bawah, ke arah diagonal atas kiri, ke arah diagonal atas kanan, ke arah diagonal bawah kiri, dan ke arah diagonal bawah kanan. Sebelum melakukan pencocokan, program akan melakukan pengecekan apakah panjang kata kunci melebihi dimensi matriks *puzzle*. Apabila panjang kata kunci yang hendak dicek sesuai arah melebihi dimensi matriks *puzzle* (akan menyebabkan *out of bounds*), maka pengecekan ke arah tersebut tidak akan dilakukan dan program akan melakukan pengecekan ke arah lainnya. Sebaliknya, apabila panjang kata kunci tidak menyebabkan *out of bounds*, maka pengecekan akan dilakukan. Pengecekan akan dihentikan apabila ditemukan satu karakter di arah tersebut yang tidak cocok. Jika pengecekan yang dilakukan berhasil, maka pencarian akan dihentikan dan program akan memberikan keluaran berupa arah yang ditemukan, jumlah perbandingan yang dilakukan, dan waktu yang dibutuhkan untuk melakukan perbandingan. Setelah semua kata kunci dicocokkan, program akan memberikan keluaran berupa matriks awal dengan kata kunci yang di-*highlight* untuk menandai posisi kata kunci di dalam *puzzle*.

# BAB II. *SOURCE PROGRAM* DALAM BAHASA JAVA

*Source program* dibuat dalam bahasa pemrograman *Java* versi "17" 2021-09-14 LTS dan *Java(TM) SE Runtime Environment (build 17+35-LTS-2724*). *Source program* dibagi ke tujuh file:

* CharObj.java
* Exec.java
* MainProgram.java
* Matcher.java
* Matrix.java
* ReadFile.java
* SearchWord.java

## 2.1 CharObj,java

File CharObj.java adalah *source code* berisi *abstract datatype* yang akan digunakan sebagai elemen dari matriks *puzzle*. ADT CharObj terdiri atas dua variabel:

* oriChar: String untuk menyimpan karakter awal
* colorChar: String untuk menyimpan karakter yang telah di-*highlight*

Pada waktu program membaca file masukan, setiap karakter akan di simpan ke sel matriks yang berisi ADT CharObj, di mana karakter akan disimpan di oriChar dan colorChar. Saat memperbarui karakter matriks untuk diberikan pewarnaan, variabel colorChar yang digunakan untuk menyimpan karakter yang telah diwarnai menggunakan pewarnaan ANSI. Setelah itu, untuk menampilkan matriks akhir, variabel colorChar yang akan dijadikan keluaran.

## 2.2 Exec.java

File Exec.java adalah *source code* berisi *abstract datatype* untuk menyimpan hasil eksekusi dari fungsi. ADT Exec terdiri atas dua variabel:

* flag: boolean untuk menandai apakah pencocokan berhasil atau tidak
* compCount: integer untuk menyimpan jumlah perbandingan

ADT Exec adalah datatype yang dikembalikan oleh fungsi-fungsi pencocokan. Apabila nilai variabel flag adalah true, maka proses pencarian akan dihentikan, namun jika nilainya false, maka proses pencarian akan terus dilanjutkan. Nilai perbandingan total kata akan dijumlahkan dengan variabel compCount.

## 2.3 MainProgram.java

File MainProgram.java berisi class yang dijalankan sebagai program utama. File MainProgram mengatur *workflow* program dari pemanggilan fungsi ReadFile, fungsi SearchWord, dan keluaran matriks akhir dan penghitungan waktu dan perbandingan.

Sebelum melakukan perbandingan, program Main akan memanggil fungsi ReadFile untuk melakukan prosedur pembacaan file. Setelah *puzzle* dan kata kunci diperoleh, program akan melakukan pemanggilan ke instansi SearchWord dengan argumen berupa kata kunci pertama di daftar dan indeks -1. Hal ini dilakukan untuk melakukan “pemanasan” terhadap instansi tersebut, sehingga pada saat prosedur perbandingan, *compiler* tidak perlu melakukan kompilasi ulang terhadap modul-modul yang dipanggil. Hal ini akan membuat hasil perbandingan lebih akurat.

Setelah melakukan “pemanasan”, prosedur pencarian akan dilakukan dengan memanggil instansi SearchWord dalam iterasi setiap elemen di daftar kata kunci. Hasil dari pencarian akan menghasilkan jumlah perbandingan dan waktu perbandingan yang akan dijumlahkan dengan variabel di dalam program Main. Setelah semua perbandingan dilakukan, program akan memberikan keluaran berupa matriks berwarna, total perbandingan, dan waktu untuk melakukan perbandingan.

## 2.4 Matcher.java

File Matcher.java berisi fungsi-fungsi yang akan dipanggil untuk melakukan pencocokan kata. Ada delapan fungsi yang digunakan untuk melakukan pencocokan:

* checkHL: mengecek kata searah horizontal kiri *(horizontally left / HL)*
* checkHR: mengecek kata searah horizontal kanan *(horizontally right / HR)*
* checkVU: mengecek kata searah vertikal atas *(vertically upper / VU)*
* checkVL: mengecek kata searah vertikal bawah *(vertically lower / VL)*
* checkDLU: mengecek kata searah diagonal kiri atas *(diagonally left upper / DLU)*
* checkDLL: mengecek kata searah diagonal kiri bawah *(diagonally left lower / DLL)*
* checkDRU: mengecek kata searah diagonal kanan atas *(diagonally right upper / DRU)*
* checkDRL: mengecek kata searah diagonal kanan bawah *(diagonally right lower / DRL)*

Secara umum, fungsi-fungsi tersebut bekerja dengan algoritma yang sama. Apabila ditemukan karakter di matriks yang sama dengan karakter pertama kata kunci yang hendak dicari, fungsi-fungsi tersebut akan dipanggil. Fungsi menerima argumen berupa matriks *puzzle*, kata kunciyang telah dipisah dalam bentuk *array*, dan koordinat dimana karakter pertama ditemukan. Setiap fungsi akan melakukan pengecekan sesuai arah. Apabila panjang kata kunci yang dicocokkan di luar dimensi matriks (*out of bounds*), maka pengecekan tidak akan dilakukan dan fungsi akan mengembalikan Exec dengan flag bernilai false. Sebaliknya, fungsi melakukan pencocokan *array* kata kunci dengan karakter-karakter di koordinat sekitar.

Dengan menggunakan skema pencarian menggunakan *boolean*, fungsi akan melakukan pencocokan per karakter. Apabila ada satu karakter yang tidak cocok, pencarian akan berhenti dan fungsi akan mengembalikan Exec dengan flag bernilai false. Sebaliknya, apabil pencocokan berhasil secara keseluruhan, maka program akan melakukan perubahan terhadap elemen matriks yang dicek. Nilai variabel colorChar dalam elemen charObj dalam matriks akan diubah menjadi karakter berwarna sesuai dengan nilai koordinat dan panjang kata yang dicek menggunakan kode ANSI. Pada akhir program, matriks yang akan ditampilkan adalah matriks dengan variabel colorChar yang telah berwarna.

## 2.5 Matrix.java

File Matrix.java berisi *abstract datatype* berupa matriks yang digunakan untuk menyimpan *puzzle* dalam bentuk matriks. Matriks yang disusun terdiri atas elemen-elemen bentukan CharObj. File ini juga mengandung fungsi-fungsi yang diperlukan untuk mencetak matriks di awal (dalam bentuk karakter asli) dan matriks dengan karakter yang telah diwarnai.

## 2.6 SearchWord.java

File SearchWord.java berisi implementasi dari fungsi-fungsi yang ada dalam file Matcher.java. File ini mengandung fungsi yang diperlukan untuk melakukan semua pencocokan secara sekuensial. Fungsi search menerima argumen berupa matriks *puzzle* dan kata kunciyang harus dicari. Fungsi akan melakukan pencocokan karakter pertama dari kata kunci terhadap elemen-elemen di matriks secara sekuensial. Apabila ditemukan karakter yang sesuai, fungsi akan melakukan pencocokan. Jika pencocokan berhasil, pencarian akan dihentikan, fungsi memberikan keluaran berupa informasi eksekusi, dan fungsi mengembalikan daftar berupa waktu dan banyaknya perbandingan eksekusi.

**2.7 ReadFile.java**

File ReadFile.java berisi fungsi yang melakukan *parsing* terhadap file masukan. Pengguna dapat memasukkan nama file tanpa .txt yang berisi matriks dan daftar kata kuncisesuai ketentuan.

Text

Description automatically generated

Setelah itu, program akan membaca isi file dan memperbarui matriks kosong dengan karakter-karakter dalam *puzzle* dan memperbarui daftar dengan daftar kata kunci yang harus dicari. *Puzzle* dan kata kunci ini yang akan diproses oleh program nantinya.

## BAB III. PENGUJIAN (INPUT/OUTPUT)

## 3.1 Pengujian terhadap file small1.txt (ukuran 18 x 16)

|  |  |
| --- | --- |
|  |  |
| Gambar 3.1.1 Isian berkas small1.txt | Gambar 3.1.2 Hasil pembacaan file masukan |
|  |  |
| Gambar 3.1.3 Informasi hasil eksekusi per kata | Gambar 3.1.4 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.2 Pengujian terhadap file small2.txt (ukuran 16 x 14)

|  |  |
| --- | --- |
|  |  |
| Gambar 3.2.1 Isian berkas small2.txt | Gambar 3.2.2 Hasil pembacaan file masukan |
|  |  |
| Gambar 3.2.3 Informasi hasil eksekusi per kata | Gambar 3.2.4 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.3 Pengujian terhadap file small3.txt (ukuran 14 x 12)

|  |  |
| --- | --- |
|  |  |
| Gambar 3.3.1 Isian berkas small3.txt | Gambar 3.3.2 Hasil pembacaan file masukan |
|  |  |
| Gambar 3.3.3 Informasi hasil eksekusi per kata | Gambar 3.3.4 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.4 Pengujian terhadap file medium1.txt (ukuran 20 x 18)

|  |  |
| --- | --- |
|  | Text  Description automatically generated |
| Gambar 3.4.1 Isian berkas medium1.txt | Gambar 3.4.2 Hasil pembacaan file masukan |
|  |  |
| Gambar 3.4.3 Informasi hasil eksekusi per kata | Gambar 3.4.4 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.5 Pengujian terhadap file medium2.txt (ukuran 22 x 20)

|  |  |
| --- | --- |
|  |  |
| Gambar 3.5.1 Isian berkas medium1.txt | Gambar 3.5.2 Hasil pembacaan file masukan |
|  |  |
| Gambar 3.5.3 Informasi hasil eksekusi per kata | Gambar 3.5.4 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.6 Pengujian terhadap file medium3.txt (ukuran 24 x 22)

|  |  |
| --- | --- |
|  |  |
| Gambar 3.6.1 Isian berkas medium3.txt | Gambar 3.6.2 Hasil pembacaan file masukan |
|  |  |
| Gambar 3.6.3 Informasi hasil eksekusi per kata | Gambar 3.6.4 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.7 Pengujian terhadap file large1.txt (ukuran 32 x 30)

|  |  |
| --- | --- |
|  |  |
| Gambar 3.7.1 Isian berkas large1.txt | Gambar 3.7.2 Hasil pembacaan file masukan |
|  |  |
| Gambar 3.7.3 Informasi hasil eksekusi per kata | Gambar 3.7.4 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.8 Pengujian terhadap file large2.txt (ukuran 34 x 32)

|  |  |
| --- | --- |
| Background pattern  Description automatically generated |  |
| Gambar 3.8.1 Isian berkas large2.txt | Gambar 3.8.2 Hasil pembacaan file masukan |
|  |  |
| Gambar 3.8.3 Informasi hasil eksekusi per kata | Gambar 3.8.4 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.9 Pengujian terhadap file large3.txt (ukuran 36 x 34)

|  |  |
| --- | --- |
|  |  |
| Gambar 3.9.1 Isian berkas large3.txt (Puzzle) | Gambar 3.9.2 Isian berkas large3.txt (Kata kunci) |
|  |  |
| Gambar 3.9.3 Hasil pembacaan file masukan (Puzzle) | Gambar 3.9.4 Hasil pembacaan file masukan (Kata kunci) |
|  |  |
| Gambar 3.9.5 Informasi hasil eksekusi per kata (1) | Gambar 3.9.6 Informasi hasil eksekusi per kata (2) |
|  |  |
| Gambar 3.9.7 Informasi hasil eksekusi per kata (3) | Gambar 3.9.8 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.10 Hasil Pengujian

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nama file (.txt) | Ukuran | Banyak kata kunci | Waktu perbandingan  (ns / ms) | Banyak hitungan |
| small1 | 18 x 16 | 10 | 1966300 / 1.966 | 1239 |
| small2 | 16 x 14 | 12 | 2321000 / 2.321 | 1373 |
| small3 | 14 x 12 | 12 | 2624200 / 2.624 | 1053 |
| medium1 | 20 x 18 | 12 | 3580900 / 3.581 | 3506 |
| medium2 | 22 x 20 | 15 | 3345500 / 3.346 | 4325 |
| medium3 | 24 x 22 | 13 | 2978800 / 2.979 | 4603 |
| large1 | 32 x 30 | 19 | 4623300 / 4.623 | 11740 |
| large2 | 34 x 32 | 20 | 4071400 / 4.071 | 13195 |
| large3 | 36 x 34 | 50 | 8482100 / 8.482 | 32683 |

Dari hasil pengujian, tampak bahwa jumlah perhitungan dan waktu perbandingan tidak berbanding terhadap ukuran, namun berbanding terhadap lokasi perbandingan dan pencocokan kata-kata yang hendak dicari. Program yang dibuat sudah bisa memproses peta berukuran 36 x 34 dengan 50 kata kunci yang harus ditemukan. Untuk kasus tersebut, algoritma yang dibuat mampu melakukan perbandingan dalam waktu 8.482 milisekon. Karena sifat bahasa Java yang menggunakan *just-in-time compiler*, sebelum melakukan iterasi kata kunci, fungsi pencarian dipanggil terlebih dahulu sehingga kompilasi dapat dilakukan sebelum perulangan untuk percobaan. Dengan demikian, hasil penghitungan waktu perbandingan diperoleh dengan lebih akurat, karena percobaan dilakukan setelah semua *instance* yang dibutuhkan telah terpanggil. Dari hasil tersebut, dapat dilihat bahwa algoritma *brute-force* dapat digunakan untuk membuat program pencarian kata dalam *word search puzzle* dengan efisiensi yang layak.

|  |  |  |
| --- | --- | --- |
| Poin | Ya | Tidak |
| 1. Program berhasil dikompilasi tanpa kesalahan (no syntax error) | √ |  |
| 2. Program berhasil running | √ |  |
| 3. Program dapat membaca file masukan dan menuliskan luaran | √ |  |
| 4. Program berhasil menemukan semua kata di dalam puzzle | √ |  |

# BAB 4. KESIMPULAN

Algoritma *brute force* adalah algoritma yang menggunakan pendekatan yang sederhana dan mudah diimplementasikan untuk menghasilkan solusi terhadap sebuah masalah. Meskipun algoritma *brute force* tidak mempunyai efisiensi yang sebanding dengan algoritma-algoritma lainnya, algoritma ini dapat digunakan untuk memecahkan mayoritas masalah yang ada, sehingga sering digunakan sebagai basis pembanding untuk algoritma-algoritma lainnya. Algoritma *brute force* mudah untuk diimplementasikan dengan mencoba seluruh kemungkinan yang ada.

Menggunakan bahasa Java, kita dapat membuat suatu program sederhana untuk memecahkan *word search puzzle* dengan implementasi algoritma *brute force*. Sebuah permainan *word search puzzle* dengan kata-kata kunci yang harus dicari dapat diselesaikan menggunakan algoritma *brute force* yang layak. Algoritma diimplementasikan dengan membandingkan karakter pertama kata kunci dengan karakter-karakter yang ada secara sekuensial, dan setelah itu melakukan perbandingan terhadap arah-arah yang mungkin dari karakter yang ditemukan.

# BAB 5. REFERENSI

* <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf>
* <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2016-2017/Makalah2017/Makalah-IF2211-2017-077.pdf>
* <http://www.swingtradesystems.com/prp/books.html#shapes>