**TUGAS KECIL 01**

**PENYELESAIAN WORD SEARCH PUZZLE**

**DENGAN ALGORITMA BRUTE FORCE**

**IF2211 – STRATEGI ALGORITMA**

A picture containing text, clipart

Description automatically generated

**Disusun oleh:**

13520034 - Bryan Bernigen

**Institut Teknologi Bandung**

**Jl. Ganesha NO. 10, Bandung 40132**

**2022**

Daftar Isi

[Penjelasan Struktur Data pada kode: 1](#_Toc94031433)

[Penjelasan Algoritma Brute Force **Tanpa Heuristik** pada program: 2](#_Toc94031434)

[Penjelasan Algoritma Brute Force **Dengan Heuristik** pada program: 2](#_Toc94031435)

[Visualisasi Algoritma Brute Force **Dengan Heuristik** pada program 3](#_Toc94031436)

[Source Code **Tanpa Heuristik** 5](#_Toc94031437)

[Source code **Dengan Heuristik** 12](#_Toc94031438)

[Source Code Program Pendukung 19](#_Toc94031439)

[Hasil Pengujian Algoritma Brute Force **Tanpa Heuristik** pada program 24](#_Toc94031440)

[Hasil Pengujian Algoritma Brute Force **Dengan Heuristik** pada program: 33](#_Toc94031441)

[Lampiran 42](#_Toc94031442)

# Penjelasan Struktur Data pada kode:

1. Matrix

Array of karakter yang bertindak sebagai “papan” pada game *word search puzzle*. Huruf-huruf acak yang secara visual terlihat dalam bentuk 2D disimpan dalam array 1D secara terurut berdasarkan baris. Suatu elemen pada baris tertentu dapat diakses dengan cara mengalikan indeks baris tersebut dengan jumlah kolom, lalu ditambahkan dengan indeks kolom

Input dari File:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | C |  | Matrix | |
| D | E | F | 🡪 | A | B | | C | D | E | F | G | H | I |
| G | H | I |

1. array\_panjang\_tiap\_kata

array of integer yang menampung informasi mengenai panjang setiap kata yang akan dicari dalam sebuah array 1D. array ini digunakan untuk mengakses setiap kata pada array\_all\_kata.

Kata yang harus Dicari:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1. Strategi |  | array\_panjang\_tiap\_kata | | | | |
| 2. Algoritma | 🡪 | 8 | 9 | 10 | |
| 3. BruteForce |  |  |  | |  | |

1. array\_all\_kata

array of karakter yang berisi seluruh kata yang harus dicari. Seluruh kata yang akan dicari disusun secara terurut berdasarkan urutan kata yang akan dicari dalam sebuah array 1D.

Kata yang harus dicari

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. STRATEGI |  | array\_all\_kata | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| 2. ALGORITMA | 🡪 | S | T | R | A | T | E | G | I | A | L | G | O | R | I | . | . | C | E |
| 3. BRUTEFORCE |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

1. pewarnaan

array of karakter dengan ukuran jumlah baris x jumlah kolom. Array ini berfungsi untuk menentukan warna sebuah karakter ketika di print. Hal tersebut untuk menandakan huruf-huruf mana saja yang akan membentuk sebuah kata. Kelemahan dari struktur data ini adalah ketika 2 buah kata menggunakan huruf yang sama. Maka warna yang akan di print adalah warna dari kata terkhir.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| matrix | | |  |  |  | pewarnaan | | |  | matrix | | |
| O | K | E |  | Cari: |  | ‘2’ | ‘2’ | ‘2’ |  | O | K | E |
| S | I | P | 🡪 | 1. OSN | 🡪 | ‘1’ | ‘0’ | ‘0’ | 🡪 | S | I | P |
| N | P | S |  | 2. OKE |  | ‘1’ | ‘0’ | ‘0’ |  | N | P | S |

\*catatan: perhatikan bahwa kata OSN tidak sepenuhnya merah karena pada bagian ‘O’, pewarnaan huruf tersebut tertimpa oleh kata OKE yang dicari setelah kata OSN.

# Penjelasan Algoritma Brute Force **Tanpa Heuristik** pada program:

1. Iterasi matrix yang berisi kata tersembunyi dari kiri ke kanan (dalam bentuk array 1D)
2. Jika ada huruf pada matrix yang diiterasi dengan huruf pertama pada kata yang akan dicari, maka:
3. Iterasi huruf pada kata tersebut ke sebuah arah sambil membandingkan dengan huruf pada matrix
4. Jika seluruh huruf sama, maka tandai huruf-huruf yang bersangkutan dengan cara megubah nilai di array pewarnaan dengan lokasi yang bersesuaian dengan huruf yang diiterasi. Lalu lompat ke Langkah 3.
5. Jika ada huruf yang berbeda, maka iterasi diterminasi
6. Ulangi Langkah 2)i) dengan mengubah arah perbandigan sampai seluruh arah sudah dicoba(arah yang memungkinkan: atas, serong kanan atas, kanan, serong kanan bawah, bawah, serang kiri bawah, kiri, dan serang kiri atas).
7. Ulangi Langkah 1 sebanyak jumlah kata yang ingin dicari (sampai seluruh kata sudah dicoba dicari)
8. Tampilkan matrix yang berisi kata tersembunyi sambil memperhatikan array pewarnaan. Array pewarnaan akan memberikan instruksi mengnai warna print setiap karakter pada lokasi yang besesuaian.

# Penjelasan Algoritma Brute Force **Dengan Heuristik** pada program:

1. Iterasi matrix yang berisi kata tersembunyi dari kiri ke kanan (dalam bentuk array 1D)
2. Jika ada huruf pada matrix yang diiterasi dengan huruf pertama pada kata yang akan dicari, maka:
3. Bandingkan panjang maximal kata yang dapat dibentuk dari posisi tersebut dengan sebuah arah tertentu dengan panjang kata yang akan dicari
4. Jika panjang kata maximal yang dapat dibentuk lebih panjang atau sama dengan panjang kata yang akan dicari, maka
5. Iterasi huruf pada kata tersebut sambil membandingkan dengan huruf pada matrix pada arah yang dipilih
6. Jika seluruh huruf sama, maka tandai huruf-huruf yang bersangkutan dengan cara megubah nilai di array pewarnaan dengan lokasi yang bersesuaian dengan huruf yang diiterasi. Lalu lompat ke Langkah 3.
7. Jika ada huruf yang berbeda, maka iterasi diterminasi.
8. Jika panjang kata maximal yang dapat dibentuk tidak lebih panjang atau sama dengan panjang kata yang akan dicari, maka iterasi huruf pada kata tidak dilakukan
9. Ulangi Langkah 2)i) dengan mengubah arah perbandigan sampai seluruh arah sudah dicoba(arah yang memungkinkan: atas, serong kanan atas, kanan, serong kanan bawah, bawah, serang kiri bawah, kiri, dan serang kiri atas).
10. Ulangi Langkah 1 sebanyak jumlah kata yang ingin dicari (sampai seluruh kata sudah dicoba dicari)
11. Tampilkan matrix yang berisi kata tersembunyi sambil memperhatikan array pewarnaan. Array pewarnaan akan memberikan instruksi mengnai warna print setiap karakter pada lokasi yang besesuaian.

# Visualisasi Algoritma Brute Force **Dengan Heuristik** pada program

1)

Contoh traversal yang dilakukan

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **O** | K | E |  | O | **K** | E |  | O | K | **E** |  | O | K | E |  |  |  |  |
| S | I | P | 🡪 | S | I | P | 🡪 | S | I | P | 🡪 | **S** | I | P | 🡪 | dst |  |  |
| N | P | S |  | N | P | S |  | N | P | S |  | N | P | S |  |  |  |  |

\*catatan: huruf yang di bold menandakan bahwa huruf tersebut sedang di bandingkan dengan huruf pertama kata yang dicari

2)

i) Misal panjang kata yang dicari = 3

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | **-** | 🡨 panjang ‘O’ ke batas lebih kecil dari panjang kata (lanjut ke 2.iii) | | |
| **O** | K | E |  | **O** | K | E |
| S | I | P | 🡪 | S | I | P |
| N | P | S |  | N | P | S |

Panjang ‘O’ ke batas ≥ panjang kata (lanjut ke 2.ii)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **O** | K | E |  | **O** | K | **E** |
| S | I | P | 🡪 | S | I | P |
| N | P | S |  | N | P | S |

ii)

I)

Contoh Iterasi:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| matrix | | |  |  | | |  |  |  |
| **O** | K | E |  | O | **K** | E |  |  |  |
| S | I | P |  | S | I | P |  |  |  |
| N | P | S | 🡪 | N | P | S | 🡪 | dst |  |
| Kata | | |  |  | | |  |  |  |
| **S** | I | P |  | S | **I** | P |  |  |  |

II) Misal Kata yang dicari : OSN

Jika Iterasi ke bawah

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **O** | K | E |  |  |
| **S** | I | P | VS | OSN |
| **N** | P | S |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **O** | S | N |  | O | **S** | N |  | O | S | **N** |
| **O** | S | N | 🡪 | O | **S** | N | 🡪 | O | S | **N** |
| Hasil: | | v |  | Hasil: | | v |  | Hasil: | | v |

\*karena hasilnya sana, maka array pewarnaan akan berubah

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sebelum : | | |  | Sesudah | | |
| O | K | E |  | O | K | E |
| S | I | P |  | S | I | P |
| N | P | S | 🡪 | N | P | S |
| pewarnaan | | | pewarnaan | | |
| ‘0’ | ‘0’ | ‘0’ |  | ‘1’ | ‘0’ | ‘0’ |
| ‘0’ | ‘0’ | ‘0’ |  | ‘1’ | ‘0’ | ‘0’ |
| ‘0’ | ‘0’ | ‘0’ |  | ‘1’ | ‘0’ | ‘0’ |

III) Misal Kata yang dicari : OSN

Jika iterasi ke kanan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **O** | **K** | **E** |  |  |
| S | I | P | VS | OSN |
| N | P | S |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **O** | K | E |  | O | **K** | E |
| **O** | S | N | 🡪 | O | **S** | N |
| Hasil: | | v |  | Hasil: | | x |

\*karena hasilnya berbeda, maka akan keluar loop

iii) langsung lanjut 2)iv)

1. Ubah arah iterasi dari atas 🡪 serang kanan atas 🡪 kanan 🡪 ….. 🡪 serong kiri atas

3) Ulangi Langkah 1 sampai seluruh kata yang ingin dicari sudah dicoba

4) Misal kata yang dicari:

1.OKE

2.SIP

3.OSN

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Pewarnaan | | |  | matrix | | |
| ‘3’ | ‘1’ | ‘1’ | 🡪 | O | K | E |
| ‘3’ | ‘2’ | ‘2’ | S | I | P |
| ‘3’ | ‘0’ | ‘0’ |  | N | P | S |

\*’O’ pada OKE dan ‘S’ pada SIP ter-*overwrite* oleh OSN sehingga warna yang dihasilkan pada O dan S adalah hijau bukan merah dan oranye

# Source Code **Tanpa Heuristik**

main.c (Algoritma Brute Force **Tanpa Heuristik** pada program)

#include <stdlib.h>

#include <stdio.h>

#include <string.h> //concate directory path

#include "readfile.h"

#include "ADTColor/pcolor.h"

#include "boolean.h"

#include <time.h>

int main()

{

    FILE \*ptr;

    char chr;

    char input\_nama\_file[256];

    char nama\_file[256] = "test/";

    int row = 0;

    int col = 0;

    char warna = 'A';

    int ukuran\_matrix;

    int jumlah\_kata = 1; // bernilai 1 jika pada akhir file tidak diikuti enter,

                         // bernilai 0 jika pada akhir file diikuti enter.

    int jumlah\_huruf = 0;

    int jumlah\_perbandingan = 0;

    int jumlah\_ditemukan = 0;

    boolean found;

    boolean cocok;

    int panjang\_all\_kata\_yang\_sudah\_ditemukan = 0;

    int i;

    int j;

    int k;

    // perhitungan waktu

    clock\_t elapsed\_time;

    // Bikin matrix

    printf("Input Nama File yang akan di buka di folder test: ");

    scanf("%s", &input\_nama\_file);

    strcat(nama\_file, input\_nama\_file);

    elapsed\_time = clock();

    readfile(&row, &col, &jumlah\_kata, &jumlah\_huruf, nama\_file);

    char \*matrix = malloc((col \* row) \* sizeof(char));

    ptr = fopen(nama\_file, "r");

    for (i = 0; i < row \* col; i++)

    {

        chr = fgetc(ptr);

        while (chr == '\n' || chr == ' ')

        {

            chr = fgetc(ptr);

        }

        matrix[i] = chr;

    }

    ukuran\_matrix = row \* col;

    // bikin kata

    int \*array\_panjang\_tiap\_kata = malloc(jumlah\_kata \* sizeof(int));

    char \*array\_all\_kata = malloc(jumlah\_huruf \* sizeof(char));

    make\_kata(array\_panjang\_tiap\_kata, array\_all\_kata, jumlah\_huruf, nama\_file);

    // main

    printf("ukuran matrix: %d\n", ukuran\_matrix);

    printf("baris x kolom: %d x %d\n", row, col);

    printf("jumlah kata  : %d\n", jumlah\_kata);

    printf("jumlah huruf : %d\n", jumlah\_huruf);

    char \*pewarnaan = malloc((col \* row) \* sizeof(char));

    for (i = 0; i < col \* row; i++)

    {

        pewarnaan[i] = '0';

    }

    // mulai iterasi

    for (i = 0; i < jumlah\_kata; i++)

    {

        found = false;

        j = 0;

        while (!found && j < ukuran\_matrix)

        {

            jumlah\_perbandingan++;

            if (matrix[j] == array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan])

            {

                // atas

                jumlah\_perbandingan++;

                cocok = true;

                k = 1;

                while (cocok && k < array\_panjang\_tiap\_kata[i])

                {

                    jumlah\_perbandingan++;

                    if (j - k \* col >= 0)

                    {

                        jumlah\_perbandingan++;

                        if (matrix[j - k \* col] != array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan + k])

                        {

                            cocok = false;

                        }

                    }

                    else

                    {

                        cocok = false;

                    }

                    k++;

                }

                jumlah\_perbandingan++;

                if (cocok)

                {

                    jumlah\_ditemukan++;

                    found = true;

                    panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

                    for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

                    {

                        pewarnaan[j - k \* col] = warna + i;

                    }

                }

                // serong kanan atas

                jumlah\_perbandingan++;

                cocok = true;

                k = 1;

                while (cocok && k < array\_panjang\_tiap\_kata[i])

                {

                    jumlah\_perbandingan++;

                    if (j - k \* col >= 0 && (j - (j / col) \* col + k - col) < 1)

                    {

                        jumlah\_perbandingan++;

                        if (matrix[j - k \* col + k] != array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan + k])

                        {

                            cocok = false;

                        }

                    }

                    else

                    {

                        cocok = false;

                    }

                    k++;

                }

                jumlah\_perbandingan++;

                if (cocok)

                {

                    jumlah\_ditemukan++;

                    found = true;

                    panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

                    for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

                    {

                        pewarnaan[j - k \* col + k] = warna + i;

                    }

                }

                // kanan

                jumlah\_perbandingan++;

                cocok = true;

                k = 1;

                while (cocok && k < array\_panjang\_tiap\_kata[i])

                {

                    jumlah\_perbandingan++;

                    if ((j - (j / col) \* col + k - col) < 1)

                    {

                        jumlah\_perbandingan++;

                        if (matrix[j + k] != array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan + k])

                        {

                            cocok = false;

                        }

                    }

                    else

                    {

                        cocok = false;

                    }

                    k++;

                }

                jumlah\_perbandingan++;

                if (cocok)

                {

                    jumlah\_ditemukan++;

                    found = true;

                    panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

                    for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

                    {

                        pewarnaan[j + k] = warna + i;

                    }

                }

                // serong kanan bawah

                jumlah\_perbandingan++;

                cocok = true;

                k = 1;

                while (cocok && k < array\_panjang\_tiap\_kata[i])

                {

                    jumlah\_perbandingan++;

                    if ((j - (j / col) \* col + k - col) < 1 && j + col \* k < col \* row)

                    {

                        jumlah\_perbandingan++;

                        if (matrix[j + k \* col + k] != array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan + k])

                        {

                            cocok = false;

                        }

                    }

                    else

                    {

                        cocok = false;

                    }

                    k++;

                }

                jumlah\_perbandingan++;

                if (cocok)

                {

                    jumlah\_ditemukan++;

                    found = true;

                    panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

                    for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

                    {

                        pewarnaan[j + k \* col + k] = warna + i;

                    }

                }

                // bawah

                jumlah\_perbandingan++;

                cocok = true;

                k = 1;

                while (cocok && k < array\_panjang\_tiap\_kata[i])

                {

                    jumlah\_perbandingan++;

                    if (j + col \* k < col \* row)

                    {

                        jumlah\_perbandingan++;

                        if (matrix[j + k \* col] != array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan + k])

                        {

                            cocok = false;

                        }

                    }

                    else

                    {

                        cocok = false;

                    }

                    k++;

                }

                jumlah\_perbandingan++;

                if (cocok)

                {

                    jumlah\_ditemukan++;

                    found = true;

                    panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

                    for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

                    {

                        pewarnaan[j + k \* col] = warna + i;

                    }

                }

                // serang kiri bawah

                jumlah\_perbandingan++;

                cocok = true;

                k = 1;

                while (cocok && k < array\_panjang\_tiap\_kata[i])

                {

                    jumlah\_perbandingan++;

                    if (j + col \* k < col \* row && j - (j / col) \* col - k >= 0)

                    {

                        jumlah\_perbandingan++;

                        if (matrix[j + k \* col - k] != array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan + k])

                        {

                            cocok = false;

                        }

                    }

                    else

                    {

                        cocok = false;

                    }

                    k++;

                }

                jumlah\_perbandingan++;

                if (cocok)

                {

                    jumlah\_ditemukan++;

                    found = true;

                    panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

                    for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

                    {

                        pewarnaan[j + k \* col - k] = warna + i;

                    }

                }

                // kiri

                jumlah\_perbandingan++;

                cocok = true;

                k = 1;

                while (cocok && k < array\_panjang\_tiap\_kata[i])

                {

                    jumlah\_perbandingan++;

                    if (j - (j / col) \* col - k >= 0)

                    {

                        jumlah\_perbandingan++;

                        if (matrix[j - k] != array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan + k])

                        {

                            cocok = false;

                        }

                    }

                    else

                    {

                        cocok = false;

                    }

                    k++;

                }

                jumlah\_perbandingan++;

                if (cocok)

                {

                    jumlah\_ditemukan++;

                    found = true;

                    panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

                    for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

                    {

                        pewarnaan[j - k] = warna + i;

                    }

                }

                // serong kiri atas

                jumlah\_perbandingan++;

                cocok = true;

                k = 1;

                while (cocok && k < array\_panjang\_tiap\_kata[i])

                {

                    jumlah\_perbandingan++;

                    if (j - (j / col) \* col - k >= 0 && j - k \* col >= 0)

                    {

                        jumlah\_perbandingan++;

                        if (matrix[j - k \* col - k] != array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan + k])

                        {

                            cocok = false;

                        }

                    }

                    else

                    {

                        cocok = false;

                    }

                    k++;

                }

                jumlah\_perbandingan++;

                if (cocok)

                {

                    jumlah\_ditemukan++;

                    found = true;

                    panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

                    for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

                    {

                        pewarnaan[j - k \* col - k] = warna + i;

                    }

                }

            }

            j++;

        }

        // Menampilkan kata yang tidak berhasil ditemukan

        jumlah\_perbandingan++;

        if (!found)

        {

            for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

            {

                printf("%c", array\_all\_kata[k + panjang\_all\_kata\_yang\_sudah\_ditemukan]);

            }

            printf("\n");

            panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

        }

    }

    elapsed\_time = clock() - elapsed\_time;

    printf("Jumlah Perbandingan: %d\n", jumlah\_perbandingan);

    printf("Jumlah Kata Ditemukan: %d\n", jumlah\_ditemukan);

    double time\_taken = ((double)elapsed\_time) / CLOCKS\_PER\_SEC; // calculate the elapsed time

    printf("Lama program: %lf detik\n", time\_taken);

    print\_warna\_matrix(matrix, pewarnaan, row, col);

    // penutup

    free(pewarnaan);

    free(array\_panjang\_tiap\_kata);

    free(array\_all\_kata);

    free(matrix);

    return 0;

}

# Source code **Dengan Heuristik**

main\_heuristik.c (Algoritma Brute Force **Dengan Heuristik** pada program)

#include <stdlib.h>

#include <stdio.h>

#include <string.h> //concate directory path

#include "readfile.h"

#include "ADTColor/pcolor.h"

#include "boolean.h"

#include <time.h>

int main()

{

    FILE \*ptr;

    char chr;

    char input\_nama\_file[256];

    char nama\_file[256] = "test/";

    int row = 0;

    int col = 0;

    char warna = 'A';

    int ukuran\_matrix;

    int jumlah\_kata = 1; // bernilai 1 jika pada akhir file tidak diikuti enter,

                         // bernilai 0 jika pada akhir file diikuti enter.

    int jumlah\_huruf = 0;

    int jumlah\_perbandingan = 0;

    int jumlah\_ditemukan = 0;

    boolean found;

    boolean cocok;

    int panjang\_all\_kata\_yang\_sudah\_ditemukan = 0;

    int i;

    int j;

    int k;

    // perhitungan waktu

    clock\_t elapsed\_time;

    // Bikin matrix

    printf("Input Nama File yang akan di buka di folder test: ");

    scanf("%s", &input\_nama\_file);

    strcat(nama\_file, input\_nama\_file);

    elapsed\_time = clock();

    readfile(&row, &col, &jumlah\_kata, &jumlah\_huruf, nama\_file);

    char \*matrix = malloc((col \* row) \* sizeof(char));

    ptr = fopen(nama\_file, "r");

    for (i = 0; i < row \* col; i++)

    {

        chr = fgetc(ptr);

        while (chr == '\n' || chr == ' ')

        {

            chr = fgetc(ptr);

        }

        matrix[i] = chr;

    }

    ukuran\_matrix = row \* col;

    // bikin kata

    int \*array\_panjang\_tiap\_kata = malloc(jumlah\_kata \* sizeof(int));

    char \*array\_all\_kata = malloc(jumlah\_huruf \* sizeof(char));

    make\_kata(array\_panjang\_tiap\_kata, array\_all\_kata, jumlah\_huruf, nama\_file);

    // main

    printf("ukuran matrix: %d\n", ukuran\_matrix);

    printf("baris x kolom: %d x %d\n", row, col);

    printf("jumlah kata  : %d\n", jumlah\_kata);

    printf("jumlah huruf : %d\n", jumlah\_huruf);

    char \*pewarnaan = malloc((col \* row) \* sizeof(char));

    for (i = 0; i < col \* row; i++)

    {

        pewarnaan[i] = '0';

    }

    // mulai iterasi

    for (i = 0; i < jumlah\_kata; i++)

    {

        found = false;

        j = 0;

        while (!found && j < ukuran\_matrix)

        {

            jumlah\_perbandingan++;

            if (matrix[j] == array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan])

            {

                // atas

                jumlah\_perbandingan++;

                if (array\_panjang\_tiap\_kata[i] <= (j / col) + 1)

                {

                    cocok = true;

                    k = 1;

                    while (cocok && k < array\_panjang\_tiap\_kata[i])

                    {

                        jumlah\_perbandingan++;

                        if (matrix[j - k \* col] != array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan + k])

                        {

                            cocok = false;

                        }

                        k++;

                    }

                    jumlah\_perbandingan++;

                    if (cocok)

                    {

                        jumlah\_ditemukan++;

                        found = true;

                        panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

                        for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

                        {

                            pewarnaan[j - k \* col] = warna + i;

                        }

                    }

                }

                // serong kanan atas

                jumlah\_perbandingan++;

                if (!found && array\_panjang\_tiap\_kata[i] <= (j / col) + 1 && (j - ((j / col)) \* col) + array\_panjang\_tiap\_kata[i] <= col)

                {

                    cocok = true;

                    k = 1;

                    while (cocok && k < array\_panjang\_tiap\_kata[i])

                    {

                        jumlah\_perbandingan++;

                        if (matrix[j - k \* col + k] != array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan + k])

                        {

                            cocok = false;

                        }

                        k++;

                    }

                    jumlah\_perbandingan++;

                    if (cocok)

                    {

                        jumlah\_ditemukan++;

                        found = true;

                        panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

                        for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

                        {

                            pewarnaan[j - k \* col + k] = warna + i;

                        }

                    }

                }

                // kanan

                jumlah\_perbandingan++;

                if (!found && (j - ((j / col)) \* col) + array\_panjang\_tiap\_kata[i] <= col)

                {

                    cocok = true;

                    k = 1;

                    while (cocok && k < array\_panjang\_tiap\_kata[i])

                    {

                        jumlah\_perbandingan++;

                        if (matrix[j + k] != array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan + k])

                        {

                            cocok = false;

                        }

                        k++;

                    }

                    jumlah\_perbandingan++;

                    if (cocok)

                    {

                        jumlah\_ditemukan++;

                        found = true;

                        panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

                        for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

                        {

                            pewarnaan[j + k] = warna + i;

                        }

                    }

                }

                // serong kanan bawah

                jumlah\_perbandingan++;

                if (!found && (j - ((j / col)) \* col) + array\_panjang\_tiap\_kata[i] <= col && (j / col) + array\_panjang\_tiap\_kata[i] <= row)

                {

                    cocok = true;

                    k = 1;

                    while (cocok && k < array\_panjang\_tiap\_kata[i])

                    {

                        jumlah\_perbandingan++;

                        if (matrix[j + k \* col + k] != array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan + k])

                        {

                            cocok = false;

                        }

                        k++;

                    }

                    jumlah\_perbandingan++;

                    if (cocok)

                    {

                        jumlah\_ditemukan++;

                        found = true;

                        panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

                        for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

                        {

                            pewarnaan[j + k \* col + k] = warna + i;

                        }

                    }

                }

                // bawah

                jumlah\_perbandingan++;

                if (!found && (j / col) + array\_panjang\_tiap\_kata[i] <= row)

                {

                    cocok = true;

                    k = 1;

                    while (cocok && k < array\_panjang\_tiap\_kata[i])

                    {

                        jumlah\_perbandingan++;

                        if (matrix[j + k \* col] != array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan + k])

                        {

                            cocok = false;

                        }

                        k++;

                    }

                    jumlah\_perbandingan++;

                    if (cocok)

                    {

                        jumlah\_ditemukan++;

                        found = true;

                        panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

                        for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

                        {

                            pewarnaan[j + k \* col] = warna + i;

                        }

                    }

                }

                // serang kiri bawah

                jumlah\_perbandingan++;

                if (!found && (j / col) + array\_panjang\_tiap\_kata[i] <= row && (j - ((j / col) \* col) + 1 - array\_panjang\_tiap\_kata[i]) >= 0)

                {

                    cocok = true;

                    k = 1;

                    while (cocok && k < array\_panjang\_tiap\_kata[i])

                    {

                        jumlah\_perbandingan++;

                        if (matrix[j + k \* col - k] != array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan + k])

                        {

                            cocok = false;

                        }

                        k++;

                    }

                    jumlah\_perbandingan++;

                    if (cocok)

                    {

                        jumlah\_ditemukan++;

                        found = true;

                        panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

                        for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

                        {

                            pewarnaan[j + k \* col - k] = warna + i;

                        }

                    }

                }

                // kiri

                jumlah\_perbandingan++;

                if (!found && (j - ((j / col) \* col) + 1 - array\_panjang\_tiap\_kata[i]) >= 0)

                {

                    cocok = true;

                    k = 1;

                    while (cocok && k < array\_panjang\_tiap\_kata[i])

                    {

                        jumlah\_perbandingan++;

                        if (matrix[j - k] != array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan + k])

                        {

                            cocok = false;

                        }

                        k++;

                    }

                    jumlah\_perbandingan++;

                    if (cocok)

                    {

                        jumlah\_ditemukan++;

                        found = true;

                        panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

                        for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

                        {

                            pewarnaan[j - k] = warna + i;

                        }

                    }

                }

                // serong kiri atas

                jumlah\_perbandingan++;

                if (!found && (j - ((j / col) \* col) + 1 - array\_panjang\_tiap\_kata[i]) >= 0 && array\_panjang\_tiap\_kata[i] <= (j / col) + 1)

                {

                    cocok = true;

                    k = 1;

                    while (cocok && k < array\_panjang\_tiap\_kata[i])

                    {

                        jumlah\_perbandingan++;

                        if (matrix[j - k \* col - k] != array\_all\_kata[panjang\_all\_kata\_yang\_sudah\_ditemukan + k])

                        {

                            cocok = false;

                        }

                        k++;

                    }

                    jumlah\_perbandingan++;

                    if (cocok)

                    {

                        jumlah\_ditemukan++;

                        found = true;

                        panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

                        for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

                        {

                            pewarnaan[j - k \* col - k] = warna + i;

                        }

                    }

                }

            }

            j++;

        }

        // Menampilkan kata yang tidak berhasil ditemukan

        jumlah\_perbandingan++;

        if (!found)

        {

            for (k = 0; k < array\_panjang\_tiap\_kata[i]; k++)

            {

                printf("%c", array\_all\_kata[k + panjang\_all\_kata\_yang\_sudah\_ditemukan]);

            }

            printf("\n");

            panjang\_all\_kata\_yang\_sudah\_ditemukan += array\_panjang\_tiap\_kata[i];

        }

    }

    elapsed\_time = clock() - elapsed\_time;

    printf("Jumlah Perbandingan: %d\n", jumlah\_perbandingan);

    printf("Jumlah Kata Ditemukan: %d\n", jumlah\_ditemukan);

    double time\_taken = ((double)elapsed\_time) / CLOCKS\_PER\_SEC; // calculate the elapsed time

    printf("Lama program: %lf detik\n", time\_taken);

    print\_warna\_matrix(matrix, pewarnaan, row, col);

    // penutup

    free(pewarnaan);

    free(array\_panjang\_tiap\_kata);

    free(array\_all\_kata);

    free(matrix);

    return 0;

}

# Source Code Program Pendukung

readfile.c

#include <stdlib.h>

#include <stdio.h>

#include "readfile.h"

void readfile(int \*row, int \*col, int \*jumlah\_kata, int \*jumlah\_huruf, char \*nama\_file)

{

    // Kamus

    FILE \*ptr;

    char chr;

    // Reading File

    ptr = fopen(nama\_file, "r");

    chr = fgetc(ptr);

    if (ptr == NULL)

    {

        printf("ERROR");

    }

    else

    {

        while (chr != '\n')

        {

            if (chr != ' ')

            {

                \*col += 1;

            }

            chr = fgetc(ptr);

        }

        \*row += 1;

        while (1)

        {

            if (chr == '\n')

            {

                chr = fgetc(ptr);

                if (chr == '\n')

                {

                    break;

                }

                else

                {

                    \*row += 1;

                }

            }

            chr = fgetc(ptr);

        }

        while (chr != EOF)

        {

            chr = fgetc(ptr);

            \*jumlah\_huruf += 1;

            if (chr == '\n')

            {

                \*jumlah\_kata += 1;

                \*jumlah\_huruf -= 1;

            }

        }

        \*jumlah\_huruf -= 1;

    }

}

void make\_kata(int \*array\_panjang\_tiap\_kata, char \*array\_all\_kata, int jumlah\_huruf, char \*nama\_file)

{

    FILE \*ptr;

    char chr;

    int jumlah\_perkata = 0;

    int i = 0;

    int j = 0;

    ptr = fopen(nama\_file, "r");

    while (1)

    {

        chr = fgetc(ptr);

        if (chr == '\n')

        {

            chr = fgetc(ptr);

            if (chr == '\n')

            {

                break;

            }

        }

    }

    chr = fgetc(ptr);

    j = 0;

    for (i = 0; i < jumlah\_huruf; i++)

    {

        if (chr != '\n')

        {

            jumlah\_perkata++;

            array\_all\_kata[i] = chr;

            chr = fgetc(ptr);

        }

        else

        {

            array\_panjang\_tiap\_kata[j] = jumlah\_perkata;

            jumlah\_perkata = 0;

            j++;

            i--;

            chr = fgetc(ptr);

        }

    }

    array\_panjang\_tiap\_kata[j] = jumlah\_perkata;

}

pcolor.c

/\*\*

Filename: pcolor.c (implementation of pcolor.h)

Description: Print character with some color in terminal

Prepared by: Asisten IF2110

Created at: 17/10/2019

\*\*/

#include <stdio.h>

#include "pcolor.h"

void print\_red(char c)

{

    printf("%s%c", RED, c);

    printf("%s", NORMAL);

}

void print\_green(char c)

{

    printf("%s%c", GREEN, c);

    printf("%s", NORMAL);

}

void print\_yellow(char c)

{

    printf("%s%c", YELLOW, c);

    printf("%s", NORMAL);

}

void print\_blue(char c)

{

    printf("%s%c", BLUE, c);

    printf("%s", NORMAL);

}

void print\_magenta(char c)

{

    printf("%s%c", MAGENTA, c);

    printf("%s", NORMAL);

}

void print\_cyan(char c)

{

    printf("%s%c", CYAN, c);

    printf("%s", NORMAL);

}

void print\_bg\_red(char c)

{

    printf("%s%c", BACKGROUND\_RED, c);

    printf("%s", NORMAL);

}

void print\_bg\_green(char c)

{

    printf("%s%c", BACKGROUND\_GREEN, c);

    printf("%s", NORMAL);

}

void print\_bg\_blue(char c)

{

    printf("%s%c", BACKGROUND\_BLUE, c);

    printf("%s", NORMAL);

}

void print\_bg\_magenta(char c)

{

    printf("%s%c", BACKGROUND\_MAGENTA, c);

    printf("%s", NORMAL);

}

void print\_bg\_cyan(char c)

{

    printf("%s%c", BACKGROUND\_CYAN, c);

    printf("%s", NORMAL);

}

void print\_warna(char c, char x)

{

    while (x > 'K')

    {

        x -= 11;

    }

    switch (x)

    {

    case 'A':

        print\_red(c);

        break;

    case 'B':

        print\_green(c);

        break;

    case 'C':

        print\_yellow(c);

        break;

    case 'D':

        print\_blue(c);

        break;

    case 'E':

        print\_magenta(c);

        break;

    case 'F':

        print\_cyan(c);

        break;

    case 'G':

        print\_bg\_red(c);

        break;

    case 'H':

        print\_bg\_green(c);

        break;

    case 'I':

        print\_bg\_blue(c);

        break;

    case 'J':

        print\_bg\_magenta(c);

        break;

    case 'K':

        print\_bg\_cyan(c);

        break;

    default:

        break;

    }

}

void print\_warna\_matrix(char \*matrix, char \*pewarnaan, int row, int col)

{

    int i;

    int j;

    for (i = 0; i < row; i++)

    {

        for (j = 0; j < col; j++)

            if (pewarnaan[i \* col + j] == '0')

            {

                printf("%c ", matrix[i \* col + j]);

            }

            else

            {

                print\_warna(matrix[i \* col + j], pewarnaan[i \* col + j]);

                printf(" ");

            }

        printf("\n");

    }

}

void print\_matrix(char \*matrix, int row, int col)

{

    int i;

    int j;

    for (i = 0; i < row; i++)

    {

        for (j = 0; j < col; j++)

            printf("%c ", matrix[i \* col + j]);

        printf("\n");

    }

}

# Hasil Pengujian Algoritma Brute Force **Tanpa Heuristik** pada program

small1.txt

Graphical user interface

Description automatically generated with medium confidenceText

Description automatically generatedA screenshot of a computer

Description automatically generated with low confidence

small2.txt menggunakan Algoritma Brute Force **Tanpa Heuristik** pada program

A screenshot of a computer

Description automatically generated with medium confidenceA picture containing text

Description automatically generated

Text

Description automatically generated with medium confidence

small3.txt menggunakan Algoritma Brute Force **Tanpa Heuristik** pada program

Graphical user interface

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidenceText

Description automatically generated

medium1.txt menggunakan Algoritma Brute Force **Tanpa Heuristik** pada program

Shape, arrow

Description automatically generatedGraphical user interface, text

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

medium2.txt menggunakan Algoritma Brute Force **Tanpa Heuristik** pada program

Graphical user interface

Description automatically generated with medium confidenceA picture containing keyboard

Description automatically generatedText

Description automatically generated

medium3.txt menggunakan Algoritma Brute Force **Tanpa Heuristik** pada program

Graphical user interface

Description automatically generatedA picture containing electronics, keyboard

Description automatically generatedText

Description automatically generated

large1.txt menggunakan Algoritma Brute Force **Tanpa Heuristik** pada program

A screenshot of a computer

Description automatically generated with medium confidenceBackground pattern

Description automatically generatedGraphical user interface, text

Description automatically generated

large2.txt menggunakan Algoritma Brute Force **Tanpa Heuristik** pada program

A screenshot of a computer

Description automatically generated with medium confidenceBackground pattern

Description automatically generatedText

Description automatically generated

large3.txt menggunakan Algoritma Brute Force **Tanpa Heuristik** pada program

Background pattern

Description automatically generatedText

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

# Hasil Pengujian Algoritma Brute Force **Dengan Heuristik** pada program:

Graphical user interface

Description automatically generatedText

Description automatically generatedA screenshot of a computer

Description automatically generated with low confidencesmall1.txt

small2.txt menggunakan Algoritma Brute Force **Dengan Heuristik**

A picture containing text

Description automatically generatedA picture containing text

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidence

small3.txt menggunakan Algoritma Brute Force **Dengan Heuristik**

Graphical user interface

Description automatically generatedText

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidence

medium1.txt menggunakan Algoritma Brute Force **Dengan Heuristik**

Graphical user interface

Description automatically generatedGraphical user interface, text

Description automatically generatedShape, arrow

Description automatically generated

medium2.txt menggunakan Algoritma Brute Force **Dengan Heuristik**

Graphical user interface

Description automatically generated with medium confidenceText

Description automatically generatedA picture containing keyboard

Description automatically generated

Text

Description automatically generatedmedium3.txt menggunakan Algoritma Brute Force **Dengan Heuristik**

Graphical user interface

Description automatically generatedA picture containing electronics, keyboard

Description automatically generated

large1.txt menggunakan Algoritma Brute Force **Dengan Heuristik**

Graphical user interface

Description automatically generatedGraphical user interface, text

Description automatically generatedBackground pattern

Description automatically generated

Text

Description automatically generatedBackground pattern

Description automatically generatedlarge2.txt menggunakan Algoritma Brute Force **Dengan Heuristik**

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generatedlarge3.txt menggunakan Algoritma Brute Force **Dengan Heuristik**

Background pattern

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

# Lampiran

Text

Description automatically generated with medium confidence

√

√

√

√

Link Drive: <https://drive.google.com/drive/folders/1lcs-Y4AqvBNqQXa8jr57rD1l6cHg9HxT?usp=sharing>