LAPORAN TUGAS KECIL

IF2211 – STRATEGI ALGORITMA

ALGORITMA PENYELESAIAN *WORD SEARCH PUZZLE* DENGAN METODE *BRUTE-FORCE*

# 

Dipersiapkan oleh:

Dzaky Fattan Rizqullah

13520003

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

JL. GANESHA 10, BANDUNG 40132

## Algoritma *Brute-Force*

Untuk program penyelesaian *word search puzzle* ini, algoritma yang digunakan yaitu algoritma *brute-force*, yang dalam hal ini program akan mengecek dan membandingkan huruf antara kata pada petak puzzle dengan huruf pada kata kunci secara satu per satu. Algoritma ini terdiri dari beberapa tahap, yaitu memilih kata kunci yang akan dicari, mencocokkan huruf pertama, mencocokkan huruf kedua, yang letaknya berada di delapan arah kemungkinan, mencocokkan huruf ketiga (bila panjang huruf lebih dari dua) hingga sisanya, dan proses diulang untuk kata kunci selanjutnya hingga seluruh kata kunci ditemukan.

Perlu diperhatikan bahwa pendekatan heuristik tidak digunakan pada algoritma ini, sehingga huruf kedua dan selanjutnya tetap akan dicek meskipun panjang kata tidak cukup untuk arah pencarian kata pada petak puzzle yang dimaksud.

## Source Program

Berikut dilampirkan kode program penyelesaian *word search puzzle* yang sudah mengimplementasi metode *brute-force*. Program ini ditulis dalam bahasa C.

/\* PROGRAM WORD SEARCH PUZZLE DENGAN BRUTE FORCE

Nama    : Dzaky Fattan Rizqullah

NIM     : 13520003

Kelas   : K3

\*/

#include <stdio.h>

#include <string.h>

#include <time.h>

*int* bacaPuzzle();       // membaca dan membentuk matriks wordsearch

*void* bacaKata();        // membaca kata kunci yang akan dicari pada wordsearch

*void* tulisPuzzle();     // menulis matriks dan kata kunci wordsearch

*void* solve();           // selesaikan wordsearch

*void* cetakmarkmtx();    // mencetak matriks

*int* cekKedua(*int* *a*, *int* *b*, *int* *wctr*);   // cek huruf kedua

*int* cekSisa(*int* *a*, *int* *b*, *int* *dir*, *int* *len*, *char* *kata*[]); // cek huruf ketiga dan seterusnya

*char* puzzle[50][50]; // penampung matriks

*char* words[100][50];  // penampung kata

*int* puzzleH, puzzleW, banyakKata;

*int* compcount, total;

// auxiliary untuk menampilkan jawaban

*void* markmtx(*int* *y*, *int* *x*, *int* *dir*, *int* *len*);

const *char* \*setclr(*int* *x*);

*char* ansmtx[50][50];

*int* xr[] = {1, 1, 0, -1, -1, -1, 0, 1};

*int* yr[] = {0, 1, 1, 1, 0, -1, -1, -1};

*int* cval = 1;

*int* main()

{

    // Fungsi Utama, termasuk perhitungan waktu real-time eksekusi

    printf("\x1b[0m"

           "PEMECAHAN WORD SEARCH PUZZLE MENGGUNAKAN ALGORITMA BRUTEFORCE (IMPLEMENTASI DENGAN BAHASA C)\n");

    printf("Nama\t: Dzaky Fattan Rizqullah\nNIM\t: 13520003\nKelas\t: K3\n");

*int* opcode = bacaPuzzle();

    if (opcode)

    {

        tulisPuzzle();

        printf("\nSelesaikan? Tekan Enter untuk menyelesaikan puzzle.\n");

        getchar();

        getchar();

*struct* timespec begin, end;

        clock\_gettime(CLOCK\_REALTIME, &begin);

        solve();

        clock\_gettime(CLOCK\_REALTIME, &end);

*double* ttaken = ((*double*)(end.tv\_nsec - begin.tv\_nsec)) / ((*double*)100000);

        printf("\n");

        cetakmarkmtx();

        printf("Waktu eksekusi program = %.2f nanodetik.\n", ttaken);

        printf("Total perbandingan huruf = %d kali.\n\n", total);

        printf("Tekan ENTER untuk keluar\n");

        getchar();

    }

    return 0;

}

*int* bacaPuzzle()

{

    // Fungsi membaca file .txt eksternal, HANYA SUPPORT EOL CRLF

    FILE \*fP;

*char* filename[50];

*char* basepath[60] = "../test/";

*int* lgth;

    printf("\nMasukkan nama file tanpa menulis ekstensi (format harus .txt)\n");

    scanf("%s", filename);

    strcat(basepath, filename);

    strcat(basepath, ".txt");

    fP = fopen(basepath, "r");

    if (fP == NULL)

    {

        printf("File error/tidak ada!\n");

        return 0;

    }

    else

    {

*char* line[100];

*int* i;

        puzzleH = 0;

        fgets(line, sizeof(line), fP);

        while (line[1] != '\n')

        {

            puzzleW = 0;

            for (i = 0; i < strlen(line) - 2; i++)

            {

                if (line[i] != ' ')

                {

                    puzzle[puzzleH][puzzleW] = line[i];

                    puzzleW++;

                }

            }

            fgets(line, sizeof(line), fP);

            puzzleH++;

        }

        banyakKata = 0;

        while (fgets(line, sizeof(line), fP) != NULL)

        {

            if (line[strlen(line) - 1] == '\n')

            {

                for (i = 0; i < (strlen(line) - 2); i++)

                {

                    words[banyakKata][i] = line[i];

                }

                banyakKata++;

            }

            else

            {

                for (i = 0; i < (strlen(line)); i++)

                {

                    words[banyakKata][i] = line[i];

                }

                banyakKata++;

            }

        }

        printf("\nBaris: %d\nKolom: %d\nBanyak Kata: %d\n", puzzleH, puzzleW, banyakKata);

        fclose(fP);

        return 1;

    }

}

*void* tulisPuzzle()

{

    // Prosedur menampilkan matriks puzzle dan daftar kata kunci yang akan dicari

    printf("\nPuzzle: \n");

    for (*int* i = 0; i < puzzleH; i++)

    {

        for (*int* j = 0; j < puzzleW; j++)

            printf("%c ", puzzle[i][j]);

        printf("\n");

    }

    printf("\nDaftar kata: \n");

    for (*int* i = 0; i < banyakKata; i++)

    {

        printf("%d. ", i + 1);

*int* j = 0;

        while (words[i][j] != '\0')

        {

            printf("%c", words[i][j]);

            j++;

        }

        printf("\n");

    }

}

*int* cekKedua(*int* *a*, *int* *b*, *int* *wctr*)

{

    // Prosedur mengecek huruf kedua untuk kata kunci yang sedang dicari

    compcount++;

    if (puzzle[*a*][*b*] == words[*wctr*][1])

        return 1;

    else

        return 0;

}

*int* cekSisa(*int* *a*, *int* *b*, *int* *dir*, *int* *len*, *char* *kata*[])

{

    // Prosedur untuk mengecek sisa huruf untuk kata kunci yang sedang dicari

*char* temp[*len*];

*int* i = 0;

*int* chk = 1;

    if (*len* > 2)

    {

        while (i < *len* - 2 && chk == 1)

        {

            if (*kata*[i + 2] != puzzle[*a*][*b*])

            {

                chk = 0;

            }

*a* += yr[*dir*];

*b* += xr[*dir*];

            i++;

        }

    }

    compcount += i;

    return chk;

}

*void* solve()

{

    // Prosedur utama untuk penyelesaian puzzle

    total = 0;

*int* wctr, dir, len;

    for (wctr = 0; wctr < banyakKata; wctr++)

    {

        compcount = 0;

*int* chked = 0;

*int* found = 0;

        len = strlen(words[wctr]);

*int* i = 0;

        while (i < puzzleH && chked == 0)

        {

*int* j = 0;

            while (j < puzzleW && chked == 0)

            {

                if (words[wctr][0] == puzzle[i][j] && chked == 0)

                {

                    compcount++;

                    if (cekKedua(i, j + 1, wctr) == 1)

                    {

                        if (cekSisa(i, j + 2, 0, len, words[wctr]) == 1)

                        {

                            markmtx(i, j, 0, len);

                            chked = 1;

                            found = 1;

                            printf("Ditemukan kata ke-%d (panjang %d huruf) pada posisi (%d,%d) ke kanan.\n", wctr + 1, len, j, i);

                        }

                    }

                    if (cekKedua(i + 1, j + 1, wctr) == 1 && found == 0)

                    {

                        if (cekSisa(i + 2, j + 2, 1, len, words[wctr]) == 1)

                        {

                            markmtx(i, j, 1, len);

                            chked = 1;

                            found = 1;

                            printf("Ditemukan kata ke-%d (panjang %d huruf) pada posisi (%d,%d) ke kanan bawah.\n", wctr + 1, len, j, i);

                        }

                    }

                    if (cekKedua(i + 1, j, wctr) == 1 && found == 0)

                    {

                        if (cekSisa(i + 2, j, 2, len, words[wctr]) == 1)

                        {

                            markmtx(i, j, 2, len);

                            chked = 1;

                            found = 1;

                            printf("Ditemukan kata ke-%d (panjang %d huruf) pada posisi (%d,%d) ke bawah.\n", wctr + 1, len, j, i);

                        }

                    }

                    if (cekKedua(i + 1, j - 1, wctr) == 1 && found == 0)

                    {

                        if (cekSisa(i + 2, j - 2, 3, len, words[wctr]) == 1)

                        {

                            markmtx(i, j, 3, len);

                            chked = 1;

                            found = 1;

                            printf("Ditemukan kata ke-%d (panjang %d huruf) pada posisi (%d,%d) ke kiri bawah.\n", wctr + 1, len, j, i);

                        }

                    }

                    if (cekKedua(i, j - 1, wctr) == 1 && found == 0)

                    {

                        if (cekSisa(i, j - 2, 4, len, words[wctr]) == 1)

                        {

                            markmtx(i, j, 4, len);

                            chked = 1;

                            found = 1;

                            printf("Ditemukan kata ke-%d (panjang %d huruf) pada posisi (%d,%d) ke kiri.\n", wctr + 1, len, j, i);

                        }

                    }

                    if (cekKedua(i - 1, j - 1, wctr) == 1 && found == 0)

                    {

                        if (cekSisa(i - 2, j - 2, 5, len, words[wctr]) == 1)

                        {

                            markmtx(i, j, 5, len);

                            chked = 1;

                            found = 1;

                            printf("Ditemukan kata ke-%d (panjang %d huruf) pada posisi (%d,%d) ke kiri atas.\n", wctr + 1, len, j, i);

                        }

                    }

                    if (cekKedua(i - 1, j, wctr) == 1 && found == 0)

                    {

                        if (cekSisa(i - 2, j, 6, len, words[wctr]) == 1)

                        {

                            markmtx(i, j, 6, len);

                            chked = 1;

                            found = 1;

                            printf("Ditemukan kata ke-%d (panjang %d huruf) pada posisi (%d,%d) ke atas.\n", wctr + 1, len, j, i);

                        }

                    }

                    if (cekKedua(i - 1, j + 1, wctr) == 1 && found == 0)

                    {

                        if (cekSisa(i - 2, j + 2, 7, len, words[wctr]) == 1)

                        {

                            markmtx(i, j, 7, len);

                            chked = 1;

                            found = 1;

                            printf("Ditemukan kata ke-%d (panjang %d huruf) pada posisi (%d,%d) ke kanan atas.\n", wctr + 1, len, j, i);

                        }

                    }

                }

                else

                {

                    compcount++;

                }

                j++;

            }

            i++;

        }

        total += compcount;

        printf("Jumlah perbandingan huruf: %d kali\n", compcount);

    }

}

// AUXILIARY

*void* markmtx(*int* *y*, *int* *x*, *int* *dir*, *int* *len*)

{

    // Prosedur menandai posisi huruf-huruf jawaban pada matriks terpisah, untuk ditampilkan nantinya

*int* count = *len*;

*int* xs = *x*;

*int* ys = *y*;

    while (count != 0)

    {

        ansmtx[ys][xs] = cval;

        ys += yr[*dir*];

        xs += xr[*dir*];

        count--;

    }

    cval = (cval % 24) + 1;

}

*void* cetakmarkmtx()

{

    // Prosedur mencetak matriks jawaban yang sudah diberi tanda warna

    for (*int* i = 0; i < puzzleH; i++)

    {

        for (*int* j = 0; j < puzzleW; j++)

        {

*int* x = ansmtx[i][j];

            switch (x)

            {

            case 1:

                printf("\x1b[31m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 2:

                printf("\x1b[32m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 3:

                printf("\x1b[33m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 4:

                printf("\x1b[34m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 5:

                printf("\x1b[35m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 6:

                printf("\x1b[36m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 7:

                printf("\x1b[30;1m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 8:

                printf("\x1b[31;1m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 9:

                printf("\x1b[32;1m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 10:

                printf("\x1b[33;1m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 11:

                printf("\x1b[34;1m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 12:

                printf("\x1b[35;1m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 13:

                printf("\x1b[36;1m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 14:

                printf("\x1b[38;5;208m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 15:

                printf("\x1b[38;5;225m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 16:

                printf("\x1b[38;5;190m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 17:

                printf("\x1b[38;5;229m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 18:

                printf("\x1b[38;5;111m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 19:

                printf("\x1b[38;5;75m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 20:

                printf("\x1b[38;5;93m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 21:

                printf("\x1b[38;5;196m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 22:

                printf("\x1b[38;5;130m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 23:

                printf("\x1b[38;5;49m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            case 24:

                printf("\x1b[38;5;213m" "%c " "\x1b[0m", puzzle[i][j]);

                break;

            default:

                printf("%c ", puzzle[i][j]);

                break;

            }

        }

        printf("\n");

    }

    printf("\n");

}

## Dokumentasi *Input* dan *Output* Program

1. Input petak puzzle berukuran 14x14

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

1. Input petak puzzle berukuran 16x16

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

1. Input petak puzzle berukuran 15x15 dan 20x20

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

1. Input petak puzzle berukuran 21x21

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

1. Input petak puzzle berukuran 24x22

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

1. Input petak puzzle berukuran 32x30

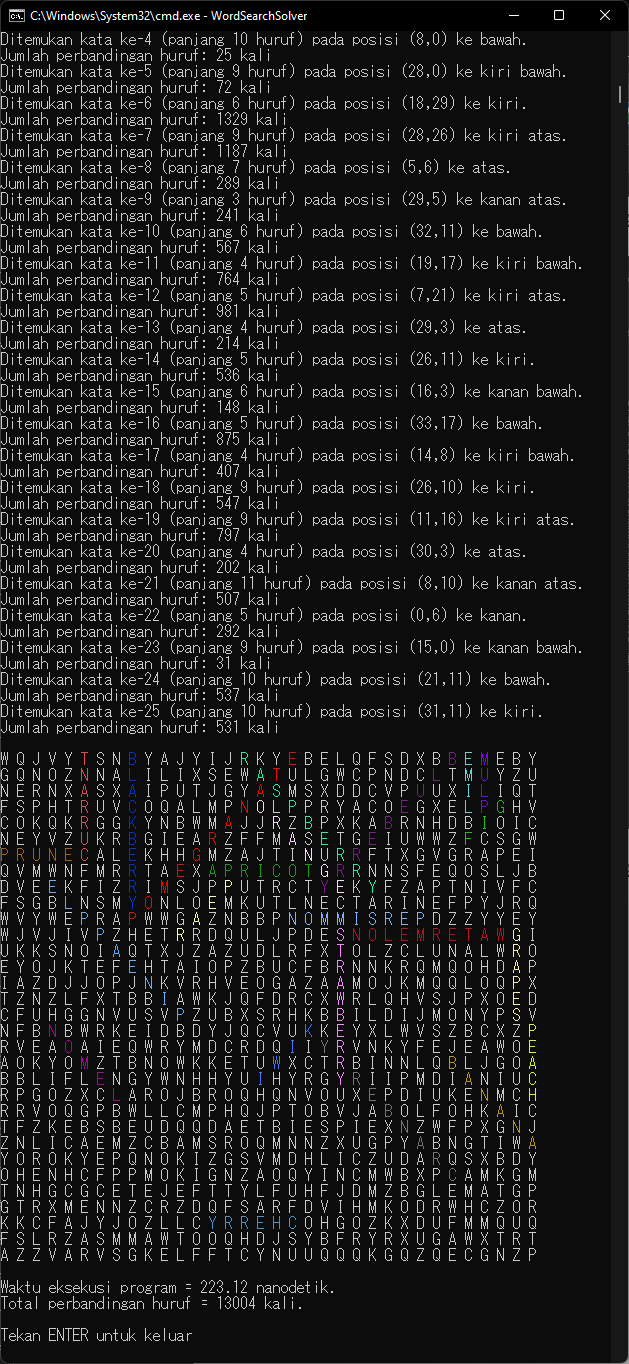
Sebuah gambar berisi teks

Deskripsi dibuat secara otomatisSebuah gambar berisi teks

Deskripsi dibuat secara otomatis

1. Input petak puzzle berukuran 34x32

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis 

1. Input petak puzzle berukuran 34x34

Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis Sebuah gambar berisi teks

Deskripsi dibuat secara otomatis

## Alamat Drive

<https://drive.google.com/drive/folders/14C_hdB1B9oVBVdL6LQnn3yp-8cw3dMN7?usp=sharing>

Link di atas akan menampilkan drive berisi kode program, executable, testcase, serta file laporan ini.

## Check List

|  |  |  |
| --- | --- | --- |
| 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 | √ |  |