**TUGAS KECIL 1 IF2211 STRATEGI ALGORITMA**

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

****

Disusun oleh

Christine Hutabarat (13520005)

**TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG 2022**

1. ALGORITMA BRUTE FORCE

Permainan word search puzzle diselesaikan dengan menggunakan algoritma brute force. Algoritma tersebut meninjau satu per satu huruf pada puzzle yang dinyatakan dalam matriks. Pada setiap posisi akan diperiksa kesamaan antara huruf dari kata yang sedang dicari dengan huruf yang sedang ditinjau pada matriks ke segala arah. Terdapat delapan arah pembacaan huruf yang dapat dinyatakan dalam nama-nama arah mata angin untuk mempermudah penulisan.

Pencarian ke setiap arah masing-masing dinyatakan dalam fungsi yang berbeda. Fungsi pencarian mengembalikan nilai boolean yang bernilai true jika kata berhasil ditemukan dalam arah tersebut, atau false jika tidak. Fungsi yang digunakan adalah fungsi rekursif, dengan basis-basisnya ialah kondisi ketika seluruh huruf dalam kata berhasil ditemukan, kondisi ketika posisi yang ditinjau berada pada batas-batas luar matriks, dan kondisi ketika huruf dari kata yang dicari tidak sama dengan kata yang ditinjau pada matriks. Rekursi dilakukan jika huruf dari kata yang dicari sama dengan huruf yang sedang ditinjau pada matriks.

1. SOURCE PROGRAM

Program disusun dalam tiga bagian, yaitu dua bagian struktur data abstrak berupa matriks dan list berkait, serta bagian kode untuk program utama. Keseluruhan program ditulis dalam bahasa C. Stuktur data matriks terdapat pada file matrix.h dan matrix.c, sementara struktur data list berkait terdapat pada file wordlist.h dan wordlist.c. Matriks pada program digunakan untuk merepresentasikan puzzle yang ada, dan list berkait digunakan untuk menyimpan daftar kata-kata yang harus dicari pada permainan.

Pada matriks, elemen yang disimpan merupakan tipe data bentukan yang adalah gabungan dari karakter dan integer sebagai kode dari warna karakter. Definisi serta fungsi struktur data matriks yang tertera pada file matrix.h adalah seperti sebagai berikut.

#ifndef MATRIX\_H

#define MATRIX\_H

#include <stdio.h>

typedef **struct** eltype {

**char** letter;

**int** color\_code;

} eltype;

*/\* COLOR CODE*

*0 - default color*

*1 - red*

*2 - green*

*3 - yellow*

*4 - blue*

*5 - purple*

*6 - cyan \*/*

typedef **struct** matrix {

    eltype buffer[50][50];

**int** nRow;

**int** nCol;

} matrix;

#define elmt(M,i,j) (M).buffer[i][j].letter

#define color(M,i,j) (M).buffer[i][j].color\_code

#define row(M) (M).nRow

#define col(M) (M).nCol

**void** copy\_matrix (matrix m1, matrix **\***m2);

**void** print\_matrix (matrix m);

#endif

Sementara itu, pada struktur list berkait setiap elemennya menyimpan informasi berupa string, bilangan bulat yang menyatakan panjang string, serta alamat dari simpul berikutnya. Definisi, selektor, serta fungsi dari struktur ini terdapat pada file wordlist.h.

#ifndef WORDLIST\_H

#define WORDLIST\_H

#include <stdio.h>

#include <stdlib.h>

#include "boolean.h"

typedef **struct** tnode\* Address;

typedef **struct** tnode {

**char** word[20];

**int** length;

    Address next;

} Node;

typedef Address wordList;

#define word(L) (L)->word

#define next(L) (L)->next

#define first(L) (L)

#define length(L) (L)->length

Address newNode (**char** w[20]);

**void** enqueue (wordList **\***wl, **char** w[20]);

**void** dequeue (wordList **\***wl);

**void** print\_wordList (wordList w);

#endif

Program utama sendiri dapat dibagi menjadi tiga bagian utama, yaitu pembacaan dan pengolahan file, pencarian kata, dan pengolahan serta penampilan matriks jawaban. Pembacaan file dilakukan oleh satu prosedur yaitu read\_file. Pencarian kata dengan memanfaatkan rekursi dilakukan oleh fungsi check\_N, check\_S, check\_E, check\_W, check\_NE, check\_SE, check\_SW, dan check\_NW. Pengolahan matriks jawaban terdapat pada prosedur ans\_matrix. Untuk menghitung waktu pencarian kata, digunakan prosedur clock\_gettime yang berada pada file header sys/time.h.

#include <stdio.h>

#include <sys/time.h>

#include <stdlib.h>

#include <string.h>

#include "boolean.h"

#include "matrix.h"

#include "wordlist.h"

**void** read\_file (**char** \*file\_name, matrix \*m, wordList \*wl) {

*// membaca file*

**char** dir**[]** = "./test/";

**char** ch;

    boolean read\_grid;

**int** i, j, k;

**char** wrd[20];

    strncat(dir, file\_name, 20);

    FILE \*fp = fopen(dir, "r");

    if (fp == NULL) {

        printf("Fail to read the file!\n");

    }

    else {

        ch = fgetc(fp);

        read\_grid = true;

        i = 0;

        j = 0;

        for (k = 0; k < 20; k++) {

            wrd[k] = '\0';

        }

        k = 0;

        while (ch != EOF) {

            if (read\_grid) {

                if (j == 0 && ch == '\n') {

                    read\_grid = false; *// found a blank line*

                }

                else if (ch == '\n') {

*// go to the next row*

                    i++;

                    row(\*m) = i;

                    j = 0;

                }

                else {

                    if ((**int**) ch != 32) {

                        elmt(\*m,i,j) = ch;

                        color(\*m,i,j) = 0;

                        j++;

                        col(\*m) = j;

                    }

                }

            }

            else {

                if (ch == '\n') {

                    enqueue(wl, wrd);

                    for (k = 0; k < 20; k++) {

                        wrd[k] = '\0';

                    }

                    k = 0;

                }

                else {

                    wrd[k] = ch;

                    k++;

                }

            }

            ch = fgetc(fp);

        }

        enqueue(wl, wrd);

        fclose(fp);

    }

}

boolean check\_N (wordList w, **int** k, matrix m, **int** i, **int** j) {

*// memeriksa ke arah utara*

    if (k == length(w)) {

        return true;

    }

    else if (i == -1 || word(w)[k] != elmt(m,i,j)) {

        return false;

    }

    else {

        return (check\_N(w, k+1, m, i-1, j));

    }

}

boolean check\_S (wordList w, **int** k, matrix m, **int** i, **int** j) {

*// memeriksa ke arah selatan*

    if (k == length(w)) {

        return true;

    }

    else if (i == row(m) || word(w)[k] != elmt(m,i,j)) {

        return false;

    }

    else {

        return (check\_S(w, k+1, m, i+1, j));

    }

}

boolean check\_W (wordList w, **int** k, matrix m, **int** i, **int** j) {

*// memeriksa ke arah barat*

    if (k == length(w)) {

        return true;

    }

    else if (j == -1 || word(w)[k] != elmt(m,i,j)) {

        return false;

    }

    else {

        return (check\_W(w, k+1, m, i, j-1));

    }

}

boolean check\_E (wordList w, **int** k, matrix m, **int** i, **int** j) {

*// memeriksa ke arah timur*

    if (k == length(w)) {

        return true;

    }

    else if (j == col(m) || word(w)[k] != elmt(m,i,j)) {

        return false;

    }

    else {

        return (check\_E(w, k+1, m, i, j+1));

    }

}

boolean check\_NE (wordList w, **int** k, matrix m, **int** i, **int** j) {

*// memeriksa ke arah timur laut*

    if (k == length(w)) {

        return true;

    }

    else if (i == -1 ||j == col(m) || word(w)[k] != elmt(m,i,j)) {

        return false;

    }

    else {

        return (check\_NE(w, k+1, m, i-1, j+1));

    }

}

boolean check\_SE (wordList w, **int** k, matrix m, **int** i, **int** j) {

*// memeriksa ke arah tenggara*

    if (k == length(w)) {

        return true;

    }

    else if (i == row(m) || j == col(m) || word(w)[k] != elmt(m,i,j)) {

        return false;

    }

    else {

        return (check\_SE(w, k+1, m, i+1, j+1));

    }

}

boolean check\_SW (wordList w, **int** k, matrix m, **int** i, **int** j) {

*// memeriksa ke arah barat daya*

    if (k == length(w)) {

        return true;

    }

    else if (i == row(m) || j == -1 || word(w)[k] != elmt(m,i,j)) {

        return false;

    }

    else {

        return (check\_SW(w, k+1, m, i+1, j-1));

    }

}

boolean check\_NW (wordList w, **int** k, matrix m, **int** i, **int** j) {

*// memeriksa ke arah barat laut*

    if (k == length(w)) {

        return true;

    }

    else if (i == -1 || j == -1 || word(w)[k] != elmt(m,i,j)) {

        return false;

    }

    else {

        return (check\_NW(w, k+1, m, i-1, j-1));

    }

}

**void** ans\_matrix (wordList w, **int** i, **int** j, matrix \*m\_ans, **char** dir) {

*// mengupdate matriks yang berisi kumpulan jawaban dengan memberi warna pada kata yang telah ditemukan*

**int** x, y, l;

**int** rand\_color;

    rand\_color = (rand() % 6) + 1;

    x = i;

    y = j;

    l = 0;

    if (dir == 'N') { *//north*

        while (l < length(w)) {

            if (color(\*m\_ans, x, y) == rand\_color) {

                rand\_color = (rand() % 6) + 1;

                l = 0;

                x = i;

                y = j;

            }

            else {

                color(\*m\_ans, x, y) = rand\_color;

                l++;

                x--;

            }

        }

    }

    else if (dir == 'S') { *//south*

        while (l < length(w)) {

            if (color(\*m\_ans, x, y) == rand\_color) {

                rand\_color = (rand() % 6) + 1;

                l = 0;

                x = i;

                y = j;

            }

            else {

                color(\*m\_ans, x, y) = rand\_color;

                l++;

                x++;

            }

        }

    }

    else if (dir == 'E') { *//east*

        while (l < length(w)) {

            if (color(\*m\_ans, x, y) == rand\_color) {

                rand\_color = (rand() % 6) + 1;

                l = 0;

                x = i;

                y = j;

            }

            else {

                color(\*m\_ans, x, y) = rand\_color;

                l++;

                y++;

            }

        }

    }

    else if (dir == 'W') { *//west*

        while (l < length(w)) {

            if (color(\*m\_ans, x, y) == rand\_color) {

                rand\_color = (rand() % 6) + 1;

                l = 0;

                x = i;

                y = j;

            }

            else {

                color(\*m\_ans, x, y) = rand\_color;

                l++;

                y--;

            }

        }

    }

    else if (dir == 'U') { *//northeast*

        while (l < length(w)) {

            if (color(\*m\_ans, x, y) == rand\_color) {

                rand\_color = (rand() % 6) + 1;

                l = 0;

                x = i;

                y = j;

            }

            else {

                color(\*m\_ans, x, y) = rand\_color;

                l++;

                x--;

                y++;

            }

        }

    }

    else if (dir == 'M') { *//southeast*

        while (l < length(w)) {

            if (color(\*m\_ans, x, y) == rand\_color) {

                rand\_color = (rand() % 6) + 1;

                l = 0;

                x = i;

                y = j;

            }

            else {

                color(\*m\_ans, x, y) = rand\_color;

                l++;

                x++;

                y++;

            }

        }

    }

    else if (dir == 'B') { *//southwest*

        while (l < length(w)) {

            if (color(\*m\_ans, x, y) == rand\_color) {

                rand\_color = (rand() % 6) + 1;

                l = 0;

                x = i;

                y = j;

            }

            else {

                color(\*m\_ans, x, y) = rand\_color;

                l++;

                x++;

                y--;

            }

        }

    }

    else if (dir == 'T') { *//northwest*

        while (l < length(w)) {

            if (color(\*m\_ans, x, y) == rand\_color) {

                rand\_color = (rand() % 6) + 1;

                l = 0;

                x = i;

                y = j;

            }

            else {

                color(\*m\_ans, x, y) = rand\_color;

                l++;

                x--;

                y--;

            }

        }

    }

}

**int** main () {

    matrix m, m\_ans;

    wordList l;

**char** filename[20];

**int** i, j, k, g;

    boolean found;

**struct** timespec begin, end;

**float** searchtime;

*// membaca file*

    printf("Enter file name : ");

    scanf("%s", &filename);

    read\_file(filename, &m, &l);

    copy\_matrix(m, &m\_ans); *// membuat matriks jawaban*

    printf("matrix size is : %dx%d\n\n", row(m), col(m));

    while (l != NULL) {

*// begin the brute force for each word in the list*

        found = false;

        i = 0;

        j = 0;

*// begin time count*

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

        while (!found) {

*// dipastikan ketemu*

            if (j == col(m)) {

*// ke baris berikutnya*

                i++;

                j = 0;

            }

            else if (check\_N(first(l), 0, m, i, j)) {

                ans\_matrix(first(l), i, j, &m\_ans, 'N');

                found = true;

            }

            else if (check\_S(first(l), 0, m, i, j)) {

                ans\_matrix(first(l), i, j, &m\_ans, 'S');

                found = true;

            }

            else if (check\_E(first(l), 0, m, i, j)) {

                ans\_matrix(first(l), i, j, &m\_ans, 'E');

                found = true;

            }

            else if (check\_W(first(l), 0, m, i, j)) {

                ans\_matrix(first(l), i, j, &m\_ans, 'W');

                found = true;

            }

            else if (check\_NE(first(l), 0, m, i, j)) {

                ans\_matrix(first(l), i, j, &m\_ans, 'U');

                found = true;

            }

            else if (check\_SE(first(l), 0, m, i, j)) {

                ans\_matrix(first(l), i, j, &m\_ans, 'M');

                found = true;

            }

            else if (check\_SW(first(l), 0, m, i, j)) {

                ans\_matrix(first(l), i, j, &m\_ans, 'B');

                found = true;

            }

            else if (check\_NW(first(l), 0, m, i, j)) {

                ans\_matrix(first(l), i, j, &m\_ans, 'T');

                found = true;

            }

            else {

*// ke kolom berikutnya*

                j++;

            }

        }

*// stop time count*

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

        searchtime = (end.tv\_nsec - begin.tv\_nsec) / 1000000000.0 + (end.tv\_sec  - begin.tv\_sec);

        printf("word ");

        for (g = 0; g < length(first(l)); g++) {

            printf("%c", word(first(l))[g]);

        }

        printf(" found in %f seconds\n", searchtime);

        dequeue(&l);

    }

    printf("\n");

    print\_matrix(m\_ans);

    printf("\n");

    printf("Fin\n");

    return 0;

}

1. HASIL PERCOBAAN

Terdapat sembilan puzzle dengan tiga ukuran berbeda yang digunakan untuk melakukan percobaan pada program. Ukuran puzzle dibagi menjadi tiga jenis, yaitu small, medium, dan large. Hasil percobaan terdapat pada tabel berikut.

|  |  |  |
| --- | --- | --- |
| Nama file | Isi file | Hasil pencarian oleh program |
| small1.txt |  |  |
| Small2.txt |  |  |
| Small3.txt |  |  |
| Medium1.txt |  |  |
| Medium2.txt |  |  |
| Medium3.txt |  |  |
| Large1.txt |  |  |
| Large2.txt |  |  |
| Large3.txt |  |  |

Keseluruhan program dapat diakses pada :