**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**

**DAFTAR ISI**

1. **ALGORITMA BRUTE FORCE2**
2. **SOURCE PROGRAM2**
3. **HASIL PERCOBAAN DAN EVALUASI13**
4. **ALAMAT KODE PROGRAM22**
5. **ALGORITMA BRUTE FORCE**

Algoritma *brute force* adalah salah satu algoritma yang langsung, sederhana, dan didasarkan pada definisi dan konsep pada persoalan yang akan diselesaikan. Algoritma *brute* force dapat menyelesaikan hampir semua persoalan. Salah satu persoalan yang dapat diselesaikan oleh algoritma ini adalah permainan mencari kata atau *word search puzzle.* Permainan ini mengharuskan pemainnya untuk mencari kata-kata yang telah diberikan pada suatu segiempat yang disusun oleh huruf-huruf. Kata-kata tersebut dapat tersusun ke delapan arah yang berbeda, yaitu ke atas, bawah, kanan, kiri, kanan atas, kanan bawah, kiri atas, dan kiri bawah.

Algoritma *brute force* yang diterapkan pada permainan ini akan meninjau satu per satu huruf pada segiempat, kemudian membandingkan huruf-huruf yang mengikutinya di delapan arah yang berbeda dengan kata yang sedang dicari. Jika kata ditemukan, maka algoritma akan berhenti dan mulai mencari kata baru. Sementara itu, jika kata tidak ditemukan, akan ditinjau huruf pada posisi lainnya di segiempat yang belum ditinjau. Adapun perbandingan antara kata yang sedang dicari dengan huruf-huruf dalam segiempat pada suatu arah dilakukan dengan mensejajarkan kata yang sedang dicari dengan rentetan huruf dalam segiempat pada arah tertentnu. Jika urutan dari huruf-huruf pada kata yang dicari dan pada segiempat sama, maka kata dinyatakan telah ditemukan, dan akan dicari kata berikutnya. Namun, jika kedua huruf tersebut berbeda, maka akan dilakukan perbandingan ke arah lain.

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. Untuk menghasilkan pencarian yang sistematis, daftar kata diperlakukan seperti sebuah antrian. 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, wordList **\***wOut);

**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");

        row(\*m) = 0;

        col(\*m) = 0;

    }

    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, l\_notfound, l\_found;

**char** filename[20];

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

    boolean found;

**struct** timespec begin, end;

**float** searchtime;

*// membaca file*

    printf("Enter file name : ");

    scanf("%s", &filename);

    read\_file(filename, &m, &l);

    if (row(m) != 0 || col(m) != 0) {

        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 && i < row(m)) {

                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++;

                }

            }

            if (found) {

*// 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, &l\_found);

            }

            else {

                dequeue(&l, &l\_notfound);

            }

        }

        printf("\n");

        print\_matrix(m\_ans);

        printf("\n");

        if (l\_notfound != NULL) {

            printf("These are the words which cannot be found :\n");

            print\_wordList(l\_notfound);

        }

        printf("\n");

        printf("Fin\n");

    }

    return 0;

}

1. **HASIL PERCOBAAN DAN EVALUASI**

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 3.1. Pada beberapa percobaan, tangkapan layar dari keluaran program dipotong karena ukuran matriks yang cukup besar dan jumlah kata yang cukup banyak.

Tabel 3.1. Hasil Percobaan pada Program

|  |  |  |
| --- | --- | --- |
| **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 |  |  |

Berdasarkan hasil yang didapat, evaluasi untuk program dapat dirangkum dalam Tabel 3.2.

Tabel 3.2. Evaluasi Program

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

1. **ALAMAT KODE PROGRAM**

Keseluruhan program dapat diakses pada repository GitHub dengan alamat :

<https://github.com/chryes220/Tucil1_Stima.git>