Laporan Tugas Kecil 1 Strategi Algoritma IF2211 Semester II Tahun 2021/2022

Penyelesaian *Word Search Puzzle* dengan Algoritma *Brute Force*

Nayotama Pradipta - 13520089

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia

1. Pendahuluan
2. Word Search Puzzle

Word Search Puzzle merupakan permainan yang melibatkan sebuah kumpulan huruf acak dan beberapa kata yang harus dicari. Kumpulan huruf acak ini disusun pada papan/kotak/matriks yang berbneutk segi-empat atau dapat disebut juga sebagai matriks huruf. Kata-kata yang harus dicari dapat ditemukan dalam delapan arah, yaitu vertikal ke atas, vertikal ke bawah, horizontal ke kanan, horizontal ke kiri, diagonal ke kanan atas, diagonal ke kanan bawah, diagonal ke kiri atas, dan diagonal ke kiri bawah. Tujuan akhir dari permainan ini adalah menemukan seluruh kata-kata yang harus dicari pada matriks huruf.

A picture containing calendar

Description automatically generated

Gambar 1.1 Illustrasi Word Search Puzzle

1. Algoritma Brute Force

Algoritma Brute Force merupakan algoritma pada suatu program yang memiliki karakteristik *straightforward*, sederhana, dan *naïve*. Algoritma ini dapat diimplementasikan dengan sangat mudah karena tidak ada unsur kreativitas dan tidak memandang efisiensi. Algoritma ini secara garis besar akan selalu memakan waktu lebih banyak dan boros memori. Brute Force seringkali digunakan sebagai basis pembanding dengan algoritma lain yang lebih efisien.

1. Langkah-langkah Algoritma Brute Force

Algoritma brute force pada *word search puzzle* adalah dengan cara mencocokkan huruf pada kata yang dicari dengan huruf yang ada di matriks. Agar perbandingan dapat dilakukan, data yang ada di text file harus diubah terlebih dahulu menjadi matriks/2D array dan 1D array. Perbandingan ini dilakukan secara delapan arah jika huruf pertama pada matriks sama dengan huruf pertama kata yang akan dicari. Program akan mengecek huruf kedua pada semua arah terlebih dahulu sebelum lanjut mencari mencari huruf pertama yang sama. Jika huruf kedua kata sama dengan huruf kedua kata yang dicari, maka selanjutnya akan dicek huruf ketiga, keempat, dst. pada arah yang sama. Pencocokan dilakukan hingga salah satu dari dua kondisi berikut terjadi:

1. Huruf tidak cocok dan panjang pencarian lebih kecil daripada panjang kata
2. Huruf selalu cocok dan jumlah huruf sama dengan panjang kata yang dicari

Jika kondisi pertama terjadi, maka program akan melanjutkan pencarian pada huruf selanjutnya pada matriks (mengular). Jika kondisi kedua terjadi, maka program sukses mencari kata, letak kata yang ditemukan akan ditampilkan dan program akan melanjutkan mencari kata selanjutnya. Proses ini akan diulang hingga seluruh kata ditemukan. Program juga memiliki fitur berupa menghitung waktu eksekusi beserta jumlah perbandingan yang dilakukan.

1. Source Program

Program yang saya buat menggunakan bahasa C++ dan dibagi menjadi tiga file sesuai dengan konsep modularitas. Ketiga file ini bernama function.hpp, function.cpp, dan main.cpp. Berikut adalah source program:

File function.hpp

extern *int* xAxis[];

extern *int* yAxis[];

*bool* isValid(std::string *userinput*);

*void* getSpec(std::string *userinput*, *int* \**matrixCol*, *int* \**matrixRow*, *int* \**words*);

*bool* searchMatrix(*char* \**matrix*, std::string *word*, *int* *row*, *int* *col*, *int* *matrixRow*, *int* *matrixCol*, *int* \**numOfCheck*, *int* \**x*, *int* \**y*);

*void* findPattern(*char* \**matrix*, std::string *word*, *int* *matrixRow*, *int* *matrixCol*, *int* \**numOfCheck*, *int* \**x*, *int* \**y*);

*void* printSolution(*int* *matrixRow*, *int* *matrixCol*, *int* *row*, *int* *col*, *int* *x*, *int* *y*);

File function.cpp

#include <iostream>

#include "function.hpp"

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

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

*bool* isValid(std::string *userinput*){

    return (*userinput* == "small.txt" || *userinput* == "medium1.txt" || *userinput* == "medium2.txt" ||*userinput* == "medium3.txt" || *userinput* == "medium4.txt" || *userinput* == "medium5.txt" || *userinput* == "hard1.txt" || *userinput* == "hard2.txt");

}

*void* getSpec(std::string *userinput*, *int* \**matrixCol*, *int* \**matrixRow*, *int* \**words*){

    // Get array and matrix size depending on the user input

    if (*userinput* == "small.txt"){ // P.S. Small Crossword Puzzle saya ambil contoh dari spesifikasi tucil karena tidak ada di web pada pdf file

        \**matrixCol* = 8;

        \**matrixRow* = 7;

        \**words* = 8;

    }

    else if (*userinput* == "medium1.txt" || *userinput* == "medium2.txt" || *userinput* == "medium3.txt" || *userinput* == "medium4.txt" || *userinput* == "medium5.txt"){ // Medium berukuran 22x20

        \**matrixCol* = 15;

        \**matrixRow* = 15;

        \**words* = 15;

    }

    else { // Large berukuran 34x32

        \**matrixCol* = 22;

        \**matrixRow* = 20;

        \**words* = 20;

    }

}

*bool* searchMatrix(*char* \**matrix*, std::string *word*, *int* *row*, *int* *col*, *int* *matrixRow*, *int* *matrixCol*, *int* \**numOfCheck*, *int* \**x*, *int* \**y*){

    // Core/Main function to search and find the words inside the matrix

    // If first letter is not equal -> break

    if (\*(*matrix*+*row*\**matrixCol*+*col*) != *word*[0]){

        return false;

    }

*int* path;

    for (path = 0; path < 8; path++){

*int* a;

*int* ra = *row* + xAxis[path];

*int* ca = *col* + yAxis[path];

        for (a = 1; a < *word*.length(); a++){

            if (ra >= *matrixRow* || ca >= *matrixCol* || ra < 0 || ca < 0){

                \**numOfCheck* += 1;

                break;

            }

            if (\*(*matrix*+ra\**matrixCol*+ca) != *word*[a]){

                \**numOfCheck* += 1;

                break;

            }

            ra += xAxis[path];

            ca += yAxis[path];

        }

        if (a == *word*.length()){

            \**x* = xAxis[path];

            \**y* = yAxis[path];

            return true;

        }

    }

    return false;

}

*void* printSolution(std::string *word*, *int* *matrixRow*, *int* *matrixCol*, *int* *row*, *int* *col*, *int* *x*, *int* *y*){

    // This function is used to display the solution in a matrix form (Assume that the word has already been found)

*char* matrixSolution[*matrixRow*][*matrixCol*];

    for (*int* i = 0; i < *matrixRow*; i++){

        for (*int* j = 0; j < *matrixCol*; j++){

            matrixSolution[i][j] = '-';

        }

    }

    matrixSolution[*row*][*col*] = *word*[0];

    for (*int* i = 1; i < *word*.length(); i++){

        matrixSolution[*row*+*x*\*i][*col*+*y*\*i] = *word*[i];

    }

    for (*int* i = 0; i < *matrixRow*; i++){

        for (*int* j = 0; j < *matrixCol*; j++){

            std::cout << matrixSolution[i][j] << " ";

        }

        std::cout << "\n";

    }

    std::cout << "\n";

}

*void* findPattern(*char* \**matrix*, std::string *word*, *int* *matrixRow*, *int* *matrixCol*, *int* \**numOfCheck*, *int* \**x*, *int* \**y*){

    for (*int* row = 0; row < *matrixRow*; row++){

        for (*int* col = 0; col < *matrixCol*; col++){

            if (searchMatrix(*matrix*, *word*, row, col, *matrixRow*, *matrixCol*, *numOfCheck*, *x*, *y*)){

                printSolution(*word*, *matrixRow*, *matrixCol*, row, col, \**x*, \**y*);

            }

            \**numOfCheck* += 1;

        }

        \**numOfCheck* += 1;

    }

}

File main.cpp:

#include <iostream>

#include "function.hpp"

#include <bits/stdc++.h>

using *namespace* std;

*int* main(){

    clock\_t start, end;

    string userinput;

    string output;

*char* alfabet = 0;

*bool* allfound = false;

*int* matrixCol = 0;

*int* matrixRow = 0;

*int* line = 0;

*int* i = 0;

*int* j = 0;

*int* k = -1;

*int* words;

*int* numOfCheck = 0;

*int* checktest = 0;

*int* x = 0;

*int* y = 0;

    cout << "Welcome to Word Search!\n";

    cout << "Please enter an input file: ";

    cin >> userinput;

    // Invalid input

    while (!isValid(userinput)){

        cout << "Invalid file name! Please input the correct file: ";

        cin >> userinput;

    }

    getSpec(userinput, &matrixCol, &matrixRow, &words);

    // Create 2D and 1D Array to store text file into array

*char* crossWordMatrix[matrixRow][matrixCol];

    string wordsToSearch[words];

    // Read File

    FILE\* inputFile = fopen(userinput.c\_str(), "r");

    // Read crossword and store inside matrix/2D array

    while (line != matrixRow){

        alfabet = getc(inputFile);

        if (alfabet != '\n'){

            crossWordMatrix[i][j] = alfabet;

        }

        if (alfabet == '\n'){

            line++;

            i++;

            j=0;

        } else {

            if (alfabet != ' '){

                j++;

            }

        }

    }

    // Read words and store inside 1D array

    while (!feof(inputFile)){

        alfabet = getc(inputFile);

        if (alfabet != '\n'){

            wordsToSearch[k] += alfabet;

            if (alfabet == ' '){

                wordsToSearch[k].pop\_back();

            }

        } else {

            k++;

        }

    }

    // Erase last character of the last word

    wordsToSearch[k].pop\_back();

    // Close file

    fclose(inputFile);

    // Algoritma Brute Force

    cout << "CrossWord Puzzle solution for " << userinput << ": \n\n";

    start = clock();

    for (*int* m = 0; m < words; m++){

        findPattern((*char* \*)crossWordMatrix, wordsToSearch[m], matrixRow, matrixCol, &numOfCheck, &x, &y);

    }

    // Testing Display Array of String

    end = clock();

*double* time\_taken = *double*(end - start)/ *double*(CLOCKS\_PER\_SEC);

    cout << "Number of Checks: " << numOfCheck << "\n";

    cout << "Execution time: " << fixed << time\_taken << setprecision(5) << " second(s)";

    cout << "\n";

}

1. Screenshot Input & Output
2. Input & Output Small/Easy (Diambil dari contoh spesifikasi Tugas Kecil):

A picture containing text

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated with medium confidence

1. A picture containing text

   Description automatically generatedInput & Output Medium 1

Background pattern

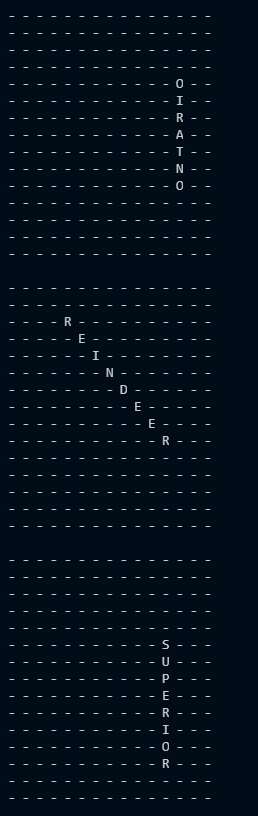
Description automatically generated

Background pattern

Description automatically generatedA picture containing text, window blind, silhouette

Description automatically generated

Background pattern

Description automatically generated

1. Input & Output Medium 2

Background pattern

Description automatically generatedBackground pattern

Description automatically generatedA picture containing graphical user interface

Description automatically generated

A picture containing window blind

Description automatically generatedA picture containing window blind

Description automatically generatedBackground pattern

Description automatically generated with low confidence

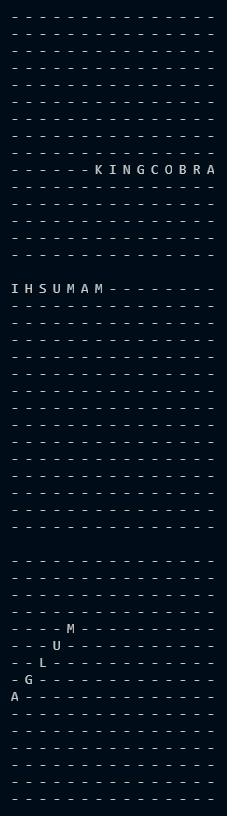
1. Input & Output Medium 3

Background pattern

Description automatically generatedBackground pattern

Description automatically generatedA picture containing graphical user interface

Description automatically generated

A screenshot of a phone

Description automatically generated with low confidenceBackground pattern

Description automatically generated

1. Background pattern

   Description automatically generatedInput & Output Medium 4

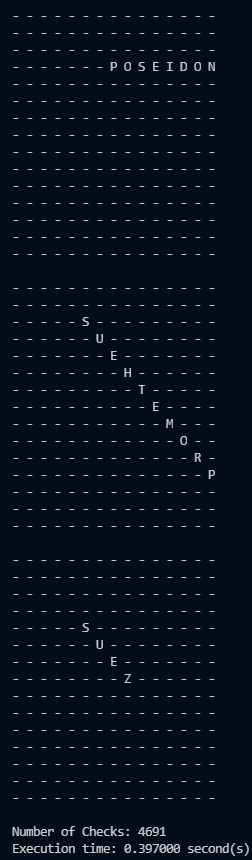
Background pattern

Description automatically generatedA picture containing text

Description automatically generated

Background pattern

Description automatically generatedA picture containing white, grate

Description automatically generated

1. Background pattern

   Description automatically generatedBackground pattern

   Description automatically generatedA picture containing text

   Description automatically generatedInput & Output Medium 5

A picture containing white, silhouette

Description automatically generatedBackground pattern

Description automatically generated

1. Background pattern

   Description automatically generatedText

   Description automatically generatedInput & Output Hard 1

Background pattern

Description automatically generatedBackground pattern

Description automatically generatedBackground pattern

Description automatically generatedBackground pattern

Description automatically generatedBackground pattern

Description automatically generatedBackground pattern

Description automatically generatedBackground pattern

Description automatically generatedBackground pattern

Description automatically generatedBackground pattern

Description automatically generated

1. Background pattern

   Description automatically generatedText

   Description automatically generated with low confidenceInput & Output Hard 2

Background pattern

Description automatically generatedBackground pattern

Description automatically generatedBackground pattern

Description automatically generatedBackground pattern

Description automatically generatedBackground pattern

Description automatically generatedBackground pattern

Description automatically generatedBackground pattern

Description automatically generatedBackground pattern

Description automatically generatedBackground pattern

Description automatically generated

1. Checklist Fitur Program

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

1. Link Drive Kode Program

<https://drive.google.com/drive/folders/1miRwRQNcH1bkfWazhA18YJcyrsQjmdU0?usp=sharing>

\*\*\* File yang terdapat di drive diatas sudah saya bagi menjadi empat folder demi kerapihan, akan tetapi untuk mencoba/testing dengan .txt file, file yang akan di test harus dipindahkan terlebih dahulu ke directory yang sama dengan directory executable. \*\*\*