Tugas Kecil 1 IF2211 Strategi Algoritma

Semester II Tahun 2021/2022

Laporan Penyelesaian *Word Search Puzzle* dengan Algoritma Brute Force

Oleh:

Adiyansa Prasetya Wicaksana

13520044

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

A. Algoritma *Brute Force*

Algoritma *brute force* merupakan cara penyelesaian masalah dengan program yang meninjau semua kasus yang mungkin muncul berdasarkan deskripsi masalah yang diberikan atau bisa dibilang algoritma diimpelemantasikan secara *straight-forward*. Meskipun dengan definisi yang diberikan, tetap ada banyak cara berbeda dalam menyelesaikan satu persoalan yang sama dengan metode *brute force*. Keragaman ini disebabkan pendekatan yang berbeda dalam mengerjakan soal, beberapa observasi yang dapat mengoptimalkan dan membuang kasus yang tidak perlu, atau pemilihan aspek mana yang di-*brute force*-kan. Secara keseluruhan, *brute force* merupakan algoritma yang mudah ditemukan dan diimplementasikan, namun membutuhkan waktu dan memori yang banyak.

Word search puzzle adalah permainan kata dimana pemain harus menemukan beberapa kata tersembunyi dalam kumpulan huruf acak. Kumpulan huruf tersebut biasa diletakkan pada "papan" berbentuk segi empat atau dapat disebut juga matriks huruf. Kata-kata pada matriks huruf ini dapat ditemukan dalam delapan arah yang mungkin, 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.

B. Penyelesaian *Word Search Puzzle* dengan *Brute Force*

Salah satu permasalahan yang dapat diselesaikan dengan implementasi *brute force* adalah permainan *word search puzzle* yang bertujuan untuk mencari kata yang tepat pada urutan tertentu di dalam *grid* yang berisi huruf-huruf. Dalam implementasi yang dilakukan, digunakan Bahasa C++ untuk menyelasaikan problematika ini. Terdapat dua file yang berada dalam *source code*. File pertama bernama solver.cpp, yang berisi algoritma untuk menampilkan output dan juga algoritma pencarian kata terhadap puzzlenya. File kedua yang sekaligus menjadi file utama bernama word_search_puzzle.cpp yang berisi algoritma utama dalam penyelesaian program.

File word_search_puzzle.cpp

Beberapa fungsi atau prosedur yang digunakan di dalam file ini adalah:

- getPuzzle

Fungsi getPuzzle berfungsi untuk membaca output bagian pertama dalam file yang akan menjadi puzzle huruf-huruf. Puzzle ini diimpelemtasikan dalam matriks 2 dimensi yang menggunakan vektor sebagai implementasinya.

- getWord

Fuingsi getWord berfungsi untuk membaca output bagian kedua dalam file yang akan menjadi kata-kata yang dicari dalam puzzle. Kata-kata ini diimplementasikan dalam vektor (array) sehingga dapat dilakukan iterasi untuk pencarian setiap katanya.

- main

Fungsi main ini merupakan fungsi utama dalam program ini. Di dalam fungsi ini diimplementasikan berbagai fungsi lainnya seperti getPuzzle dan getWord. Disini dilakukan iterasi terhadap setiap kata yang terdapat dalam vektor word, lalu dilakukan search delapan arah ke setiap elemen puzzle.

File solver.cpp

Beberapa fungsi atau prosedur yang digunakan di dalam file ini adalah:

- printHash

Fungsi printHash mengeluarkan output terhadap hash table yang telah dibuat. Hash table ini merupakan matriks 2 dimensi dengan implementasi dengan besarnya sesuai dengan besar dari puzzlenya. Nilai dari hash table berisi nilai dari 0 sampai 7 sebagai tanda untuk pewarnaan kata yang ditemukan. 0 berarti tidak ditemukannya kata pada huruf di puzzlenya dan 1 sampai 7 berkorespondensi berdasarkan warnanya (ditemukan kata).

- searchHorizontalRight
- searchHorizontalLeft
- searchVerticalUp
- searchVerticalDown
- searchDiagonalRightUp
- searchDiagonalRightDown
- searchDiagonalLeftUp
- searchDiagonalLeftDown

Delapan fungsi search bertujuan dan berstruktur mirip yaitu untuk melakukan pencarian terhadap elemen puzzle untuk menemukan kata tujuan. Setiap fungsi merepresentasikan 8 arah yang ingin dicari. Konsep dari setiap fungsinya adalah dengan melakukan iterasi kepada setiap elemen dan setiap elemennya dilakukan iterasi tergantung dari arahnya sebanyak panjang dari kata yang ingin dicari untuk setiap elemen. Jika setiap huruf cocok dengan setiap huruf dari kata yang dicari maka algoritma akan menyimpan kata yang telah ditemukan ke dalam hash table dengan nilai 1 sampai 7 tergantung dari indeks pada inputnya.

C. Source Program

1. File word_puzzle_search.cpp

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <chrono>
#include "solver.cpp"
using namespace std::chrono;
using namespace std;

```

```cpp
vector<vector<char>> getPuzzle(ifstream &InputText)
{
    vector<vector<char>> puzzle;
    vector<char> puzzleLine;
    char input;
    int countNewLine = 0;

    while (InputText.get(input))
    {
        if (input == '\n')
        {
            countNewLine++;

            if (countNewLine == 2)
            {
                break;
            }
            puzzle.push_back(puzzleLine);
            puzzleLine = {};

            continue;
        }

        countNewLine = 0;

        if (input != ' ')
        {
            puzzleLine.push_back(input);
        }
    }

    return puzzle;
}
```

```cpp
vector<string> getWord(ifstream &InputText)
{
    vector<string> word;
    char input;
    string text;

    while (InputText.get(input))
    {
        if (input == '\n')
        {
            word.push_back(text);
            text = "";
        }

        else
        {
            text.push_back(input);
        }
    }
    return word;
}
```

```cpp
int main()
{
    string fileName;
    cout << "Input your file name: ";
    cin >> fileName;

    // Read from the text file
    string path;
    path = "../test/" + fileName;
    ifstream InputText(path);

    while (!InputText)
    {
        cout << "File name doesn't exist, please try again!" << endl;
        cout << "Input your file name: ";
        cin >> fileName;
        path = "../test/" + fileName;
        InputText.clear();
        InputText.seekg(0, InputText.beg);
        InputText.open(path);
    }

    // Get Puzzle
    vector<vector<char>> puzzle = getPuzzle(InputText);

    // Get Word
    vector<string> word = getWord(InputText);

    // Create Boolean Hash Table
    vector<vector<int>> puzzleHash;
    for (int i = 0; i < puzzle.size(); i++)
    {
        puzzleHash.push_back({});
        for (int j = 0; j < puzzle[0].size(); j++)
        {
            puzzleHash[i].push_back(0);
        }
    }

    // Start runtime
    auto start = high_resolution_clock::now();

    // Check for every iteration for every word
    int compareCount = 0;
    for (int i = 0; i < word.size(); i++)
    {
        searchHorizontalRight(puzzle, puzzle.size(), puzzle[0].size(), word[i], puzzleHash, i, compareCount);
        searchHorizontalLeft(puzzle, puzzle.size(), puzzle[0].size(), word[i], puzzleHash, i, compareCount);
        searchVerticalUp(puzzle, puzzle.size(), puzzle[0].size(), word[i], puzzleHash, i, compareCount);
        searchVerticalDown(puzzle, puzzle.size(), puzzle[0].size(), word[i], puzzleHash, i, compareCount);
        searchDiagonalRightUp(puzzle, puzzle.size(), puzzle[0].size(), word[i], puzzleHash, i, compareCount);
        searchDiagonalRightDown(puzzle, puzzle.size(), puzzle[0].size(), word[i], puzzleHash, i, compareCount);
        searchDiagonalLeftUp(puzzle, puzzle.size(), puzzle[0].size(), word[i], puzzleHash, i, compareCount);
        searchDiagonalLeftDown(puzzle, puzzle.size(), puzzle[0].size(), word[i], puzzleHash, i, compareCount);
    }

    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);

    cout << endl;
    printHash(puzzle, puzzleHash);
    cout << "Execution Time    : " << (float)(duration.count()) / 1000000 << " seconds" << endl;
    cout << "Total Comparision : " << compareCount << endl;

    InputText.close();
}
```

2. File solver.cpp

```cpp
#define RESET   "\033[0m"
#define BLACK   "\033[30m"      /* Black */
#define RED     "\033[31m"      /* Red */
#define GREEN   "\033[32m"      /* Green */
#define YELLOW  "\033[33m"      /* Yellow */
#define BLUE    "\033[34m"      /* Blue */
#define MAGENTA "\033[35m"      /* Magenta */
#define CYAN    "\033[36m"      /* Cyan */
#define WHITE   "\033[37m"      /* White */
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
using namespace std;
```

```cpp
void printHash(vector<vector<char>> puzzle, vector<vector<int>> puzzleHash)
{
    int i, j;
    int row = puzzle.size();
    int col = puzzle[0].size();

    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            if (puzzleHash[i][j] == 1)
            {
                cout << WHITE << puzzle[i][j] << RESET << " ";
            }

            else if (puzzleHash[i][j] == 2)
            {
                cout << CYAN << puzzle[i][j] << RESET << " ";
            }

            else if (puzzleHash[i][j] == 3)
            {
                cout << RED << puzzle[i][j] << RESET << " ";
            }

            else if (puzzleHash[i][j] == 4)
            {
                cout << GREEN << puzzle[i][j] << RESET << " ";
            }

            else if (puzzleHash[i][j] == 5)
            {
                cout << YELLOW << puzzle[i][j] << RESET << " ";
            }

            else if (puzzleHash[i][j] == 6)
            {
                cout << BLUE << puzzle[i][j] << RESET << " ";
            }

            else if (puzzleHash[i][j] == 7)
            {
                cout << MAGENTA << puzzle[i][j] << RESET << " ";
            }

            else
            {
                cout << BLACK << puzzle[i][j] << RESET << " ";
            }
        }
        cout << endl;
    }

    cout << endl;
}
```

```cpp
void searchHorizontalRight(vector<vector<char>> puzzle, int row, int col,
string word, vector<vector<int>> &puzzleHash, int wordNumber)
{
    bool found = false;
    int wordSize = word.size();
    int wordCount = 0;
    int i, j, k;
    int p, q;

    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            wordCount = 0;
            for (k = j; k - j < wordSize; k++)
            {
                if (k >= col)
                {
                    break;
                }

                if (puzzle[i][k] == word[k - j])
                {
                    wordCount++;
                }
                else
                {
                    break;
                }
            }

            if (wordCount == wordSize)
            {
                found = true;
                q = j;
                break;
            }
        }
        if (found)
        {
            p = i;
            break;
        }
    }

    if (found)
    {

        for (j = q; j - q < wordSize; j++)
        {
            puzzleHash[p][j] = (wordNumber % 7) + 1;
        }
    }
}
```

```cpp
void searchHorizontalLeft(vector<vector<char>> puzzle, int row, int col,
string word, vector<vector<int>> &puzzleHash, int wordNumber)
{
    bool found = false;
    int wordSize = word.size();
    int wordCount = 0;
    int i, j, k;
    int p, q;

    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            wordCount = 0;
            for (k = j; j - k < wordSize; k--)
            {
                if (k < 0)
                {
                    break;
                }

                if (puzzle[i][k] == word[j - k])
                {
                    wordCount++;
                }
                else
                {
                    break;
                }
            }

            if (wordCount == wordSize)
            {
                found = true;
                q = j;
                break;
            }
        }
        if (found)
        {
            p = i;
            break;
        }
    }

    if (found)
    {

        for (j = q; q - j < wordSize; j--)
        {
            puzzleHash[p][j] = (wordNumber % 7) + 1;
        }
    }
}
```

```cpp
void searchVerticalUp(vector<vector<char>> puzzle, int row, int col,
string word, vector<vector<int>> &puzzleHash, int wordNumber)
{
    bool found = false;
    int wordSize = word.size();
    int wordCount = 0;
    int i, j, k;
    int p, q;

    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            wordCount = 0;
            for (k = i; i - k < wordSize; k--)
            {
                if (k < 0)
                {
                    break;
                }

                if (puzzle[k][j] == word[i - k])
                {
                    wordCount++;
                }
                else
                {
                    break;
                }
            }

            if (wordCount == wordSize)
            {
                found = true;
                q = j;
                break;
            }
        }
        if (found)
        {
            p = i;
            break;
        }
    }

    if (found)
    {

        for (i = p; p - i < wordSize; i--)
        {
            puzzleHash[i][q] = (wordNumber % 7) + 1;
        }
    }
}
```

```cpp
void searchVerticalDown(vector<vector<char>> puzzle, int row, int col,
string word, vector<vector<int>> &puzzleHash, int wordNumber)
{
    bool found = false;
    int wordSize = word.size();
    int wordCount = 0;
    int i, j, k;
    int p, q;

    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            wordCount = 0;
            for (k = i; k - i < wordSize; k++)
            {
                if (k >= row)
                {
                    break;
                }

                if (puzzle[k][j] == word[k - i])
                {
                    wordCount++;
                }
                else
                {
                    break;
                }
            }

            if (wordCount == wordSize)
            {
                found = true;
                q = j;
                break;
            }
        }
        if (found)
        {
            p = i;
            break;
        }
    }

    if (found)
    {

        for (i = p; i - p < wordSize; i++)
        {
            puzzleHash[i][q] = (wordNumber % 7) + 1;
        }
    }
}
```

```cpp
void searchDiagonalRightUp(vector<vector<char>> puzzle, int row, int col,
string word, vector<vector<int>> &puzzleHash, int wordNumber)
{
    bool found = false;
    int wordSize = word.size();
    int wordCount = 0;
    int i, j, k, l;
    int p, q;

    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            wordCount = 0;
            k = i;
            l = j;
            while (i - k < wordSize && l - j < wordSize)
            {
                if (k < 0 || l >= col)
                {
                    break;
                }

                if (puzzle[k][l] == word[i - k])
                {
                    wordCount++;
                }
                else
                {
                    break;
                }
                k--;
                l++;
            }

            if (wordCount == wordSize)
            {
                found = true;
                q = j;
                break;
            }
        }
        if (found)
        {
            p = i;
            break;
        }
    }

    if (found)
    {

        i = p;
        j = q;
        while (p - i < wordSize && j - q < wordSize)
        {
            puzzleHash[i][j] = (wordNumber % 7) + 1;
            i--;
            j++;
        }
    }
}
```

```cpp
void searchDiagonalRightDown(vector<vector<char>> puzzle, int row, int col,
string word, vector<vector<int>> &puzzleHash, int wordNumber)
{
    bool found = false;
    int wordSize = word.size();
    int wordCount = 0;
    int i, j, k, l;
    int p, q;

    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            wordCount = 0;
            k = i;
            l = j;
            while (k - i < wordSize && l - j < wordSize)
            {
                if (k >= row || l >= col)
                {
                    break;
                }

                if (puzzle[k][l] == word[k - i])
                {
                    wordCount++;
                }
                else
                {
                    break;
                }
                k++;
                l++;
            }

            if (wordCount == wordSize)
            {
                found = true;
                q = j;
                break;
            }
        }
        if (found)
        {
            p = i;
            break;
        }
    }

    if (found)
    {

        i = p;
        j = q;
        while (i - p < wordSize && j - q < wordSize)
        {
            puzzleHash[i][j] = (wordNumber % 7) + 1;
            i++;
            j++;
        }
    }
}
```

```cpp
void searchDiagonalLeftUp(vector<vector<char>> puzzle, int row, int col,
string word, vector<vector<int>> &puzzleHash, int wordNumber)
{
    bool found = false;
    int wordSize = word.size();
    int wordCount = 0;
    int i, j, k, l;
    int p, q;

    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            wordCount = 0;
            k = i;
            l = j;
            while (i - k < wordSize && j - l < wordSize)
            {
                if (k < 0 || l < 0)
                {
                    break;
                }

                if (puzzle[k][l] == word[i - k])
                {
                    wordCount++;
                }
                else
                {
                    break;
                }
                k--;
                l--;
            }

            if (wordCount == wordSize)
            {
                found = true;
                q = j;
                break;
            }
        }
        if (found)
        {
            p = i;
            break;
        }
    }

    if (found)
    {

        i = p;
        j = q;
        while (p - i < wordSize && q - j < wordSize)
        {
            puzzleHash[i][j] = (wordNumber % 7) + 1;
            i--;
            j--;
        }
    }
}
```

```cpp
void searchDiagonalLeftDown(vector<vector<char>> puzzle, int row, int col,
string word, vector<vector<int>> &puzzleHash, int wordNumber)
{
    bool found = false;
    int wordSize = word.size();
    int wordCount = 0;
    int i, j, k, l;
    int p, q;

    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            wordCount = 0;
            k = i;
            l = j;
            while (k - i < wordSize && j - l < wordSize)
            {
                if (k >= row || l < 0)
                {
                    break;
                }

                if (puzzle[k][l] == word[j - l])
                {
                    wordCount++;
                }
                else
                {
                    break;
                }
                k++;
                l--;
            }

            if (wordCount == wordSize)
            {
                found = true;
                q = j;
                break;
            }
        }
        if (found)
        {
            p = i;
            break;
        }
    }

    if (found)
    {

        i = p;
        j = q;
        while (i - p < wordSize && q - j < wordSize)
        {
            puzzleHash[i][j] = (wordNumber % 7) + 1;
            i++;
            j--;
        }
    }
}
```

D.Input dan Output

1. tc1_small.txt

2. tc2_small.txt

3. tc3_small.txt

```
S P I H M H P U P P Y L O V E L F
C H E R I S H B S I H P M E M V W
Y E L E N O R E G L Y T U R T O B
A M T E V L E V E U L B S I I A L
D L H I T Q A N G M Y N A F B P P
T Y E D K R P Y T A I D P Y B N G
L I T G N Z M H D V O X L S A Q C
O M W X N M R R E W O O H I R S R
H Z I W Y A E U N A V O R R E O Y
Y V S U L T N I N E T E R J T U I
P A T V S N O E F A L W X G I L N
A V T E S W D O E L W Q A G H M G
N X Y S N S S Y A T R A M V W A W
A I M A R A C V Y M Y D Y A E N W

BABYLOVE
BLUEVELVET
CARAMIA
CHERISH
CRYING
DOWNTOWN
ELENORE
FIRE
GROOVIN
HEATWAVE
HOLIDAY
MEMPHIS
MYGUY
PUPPYLOVE
RUNAWAY
SOULMAN
STAY
TEENANGEL
THETWIST
VALLERI
WHITERABBIT
WINDY
YESTERDAY
```

```
Input your file name: tc3_small.txt

S P I H M H P U P P Y L O V E L F
C H E R I S H B S I H P M E M V W
Y E L E N O R E G L Y T U R T O B
A M T E V L E V E U L B S I I A L
D L H I T Q A N G M Y N A F B P P
T Y E D K R P Y T A I D P Y B N G
L I T G N Z M H D V O X L S A Q C
O M W X N M R R E W O O H I R S R
H Z I W Y A E U N A V O R R E O Y
Y V S U L T N I N E T E R J T U I
P A T V S N O E F A L W X G I L N
A V T E S W D O E L W Q A G H M G
N X Y S N S S Y A T R A M V W A W
A I M A R A C V Y M Y D Y A E N W

Execution Time    : 0.001995 seconds
Total Comparision : 43178
```

4. tc4_med.txt

```
D Z N F B F F F F Y P T H R U X Y B R I N Y
V G C B X D X I R R E G U L A R J Y J I R R
X J L R G M N U U T M V O H J B X T J H F E
I B W G K D W P U S O L I D F L T L Q U E C
T I N E R T N E S S Z E S F W B T D A D C O
O O N F C H B L K D G C V N G Z X E D H D V
S M H I N V N A B V O N C N Y S M B L X I E
O J W R D N C N H O U E J W T M J I L P A R
J Q O M V E A I C B F R G K S L D R O W R U
F Y E N U J I F D P A E Y Y N K A C G X B P
P R E S C R I B E D D H A R A N X S Q F K Z
C K N I I Q G H N X M D I U L R L N P K X D
R U R U G U A Y D G C A C J U R V I M L Q B
K Q C V J W N Z K S Q J V R T W B I J G V I
U H O E D I K H E O G I F E A R S R Q B I B
Y I H E R E W O H S J H Y P P G G Q T P H L
S I S A S Q D E S U M A S J S N W P M A R I
O J O T T O T A L L Y A A O T H U V S H O C
K C K C R E R R I X T O G G N I K C A S C A
T S E N O H S I D G T Z M F U P J M Z J Y L

ADHERENCE
AMUSE
BIBLICAL
BRAID
BRINY
DISHONEST
FINAL
INERTNESS
INSCRIBED
IRREGULAR
JUNE
PERJURY
PRESCRIBED
RECOVER
SACKING
SHOWER
SOLID
SPATULA
```

```
Input your file name: tc4_med.txt

D Z N F B F F F F Y P T H R U X Y B R I N Y
V G C B X D X I R R E G U L A R J Y J I R R
X J L R G M N U U T M V O H J B X T J H F E
I B W G K D W P U S O L I D F L T L Q U E C
T I N E R T N E S S Z E S F W B T D A D C O
O O N F C H B L K D G C V N G Z X E D H D V
S M H I N V N A B V O N C N Y S M B L X I E
O J W R D N C N H O U E J W T M J I L P A R
J Q O M V E A I C B F R G K S L D R O W R U
F Y E N U J I F D P A E Y Y N K A C G X B P
P R E S C R I B E D D H A R A N X S Q F K Z
C K N I I Q G H N X M D I U L R L N P K X D
R U R U G U A Y D G C A C J U R V I M L Q B
K Q C V J W N Z K S Q J V R T W B I J G V I
U H O E D I K H E O G I F E A R S R Q B I B
Y I H E R E W O H S J H Y P P G G Q T P H L
S I S A S Q D E S U M A S J S N W P M A R I
O J O T T O T A L L Y A A O T H U V S H O C
K C K C R E R R I X T O G G N I K C A S C A
T S E N O H S I D G T Z M F U P J M Z J Y L

Execution Time    : 0.001998 seconds
Total Comparision : 62420
```

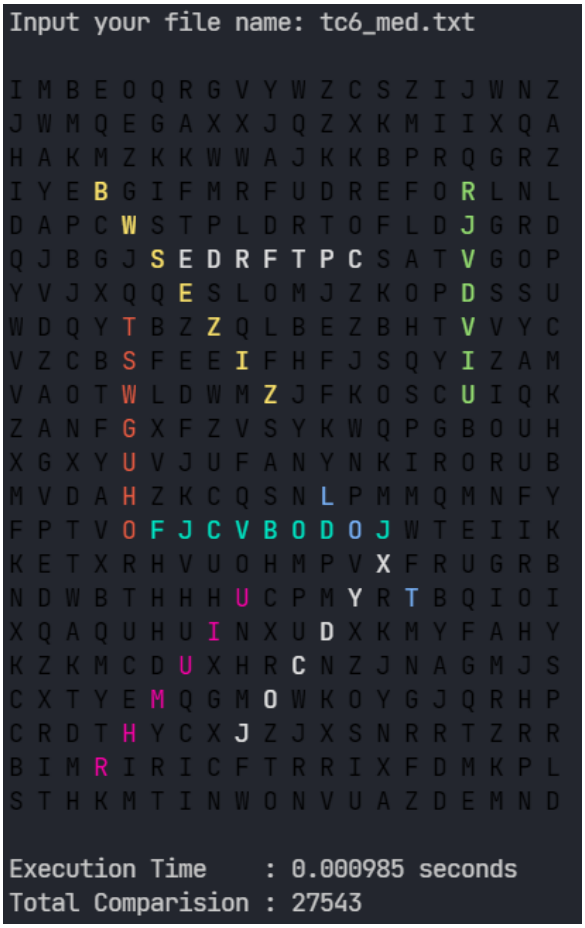
5. tc5_med.txt

```
V H S T R I P E S O R Q J Q Z S X Z R A P V
I T N L A N O O M X E I H L A M M A M S J I
O Z X F G H A W D X T G W O G Y L T A E L S
I R E E E L E I X Z N V E L G N U J S E P Y
C K N W S C H G R R U I C Z N Q O E S O H W
A Z I B U W E M E E H Q L R Q V N W Q W Z Y
R A L B M W W R T O B C A O G I N A I S A Q
N Z E G A R V Z K M O I W S H M A L A Y A N
I U F V T O O W A N D D S C C C Z J T N D E
V S W N R O G T S U J G O U R D T H E O J D
O S D A A Z V E A U E D G Q D O I N X A V T
R P N I N C R S U D N I S T J Z D Q A F M D
E G I N V V N E R I E J B C B A P O E R U N
E R E T A E T A E M C R B S N F W Z X I A C
T F B T L A G N E B X V P G T M I I Q C W H
Q N I O I I K N D L I W E E E J D K B A V G
D O L I A T A T U R Z R E T A E N A M N V V
N C Y U J B A S P W E T Z I X V A Q A P A C
J V B W F B I L G D H X B K H Q T X V N M P
K F F T S G N I R A O R J E Z I P J F M D H

AFRICAN
ASIAN
BENGAL
CLAWS
FELINE
HUNTER
JUNGLE
ORANGE
PREDATOR
ROARING
SIBERIAN
STRIPES
TAIL
TEETH
WILD
```

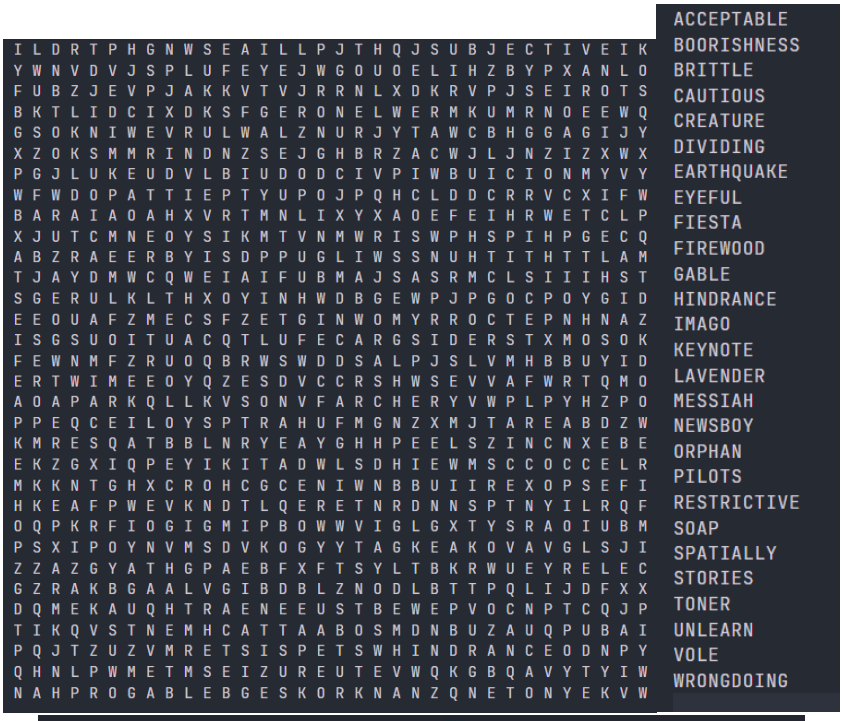
```
Input your file name: tc5_med.txt

V H S T R I P E S O R Q J Q Z S X Z R A P V
I T N L A N O O M X E I H L A M M A M S J I
O Z X F G H A W D X T G W O G Y L T A E L S
I R E E E L E I X Z N V E L G N U J S E P Y
C K N W S C H G R R U I C Z N Q O E S O H W
A Z I B U W E M E E H Q L R Q V N W Q W Z Y
R A L B M W W R T O B C A O G I N A I S A Q
N Z E G A R V Z K M O I W S H M A L A Y A N
I U F V T O O W A N D D S C C C Z J T N D E
V S W N R O G T S U J G O U R D T H E O J D
O S D A A Z V E A U E D G Q D O I N X A V T
R P N I N C R S U D N I S T J Z D Q A F M D
E G I N V V N E R I E J B C B A P O E R U N
E R E T A E T A E M C R B S N F W Z X I A C
T F B T L A G N E B X V P G T M I I Q C W H
Q N I O I I K N D L I W E E E J D K B A V G
D O L I A T A T U R Z R E T A E N A M N V V
N C Y U J B A S P W E T Z I X V A Q A P A C
J V B W F B I L G D H X B K H Q T X V N M P
K F F T S G N I R A O R J E Z I P J F M D H

Execution Time    : 0.001003 seconds
Total Comparision : 51699
```

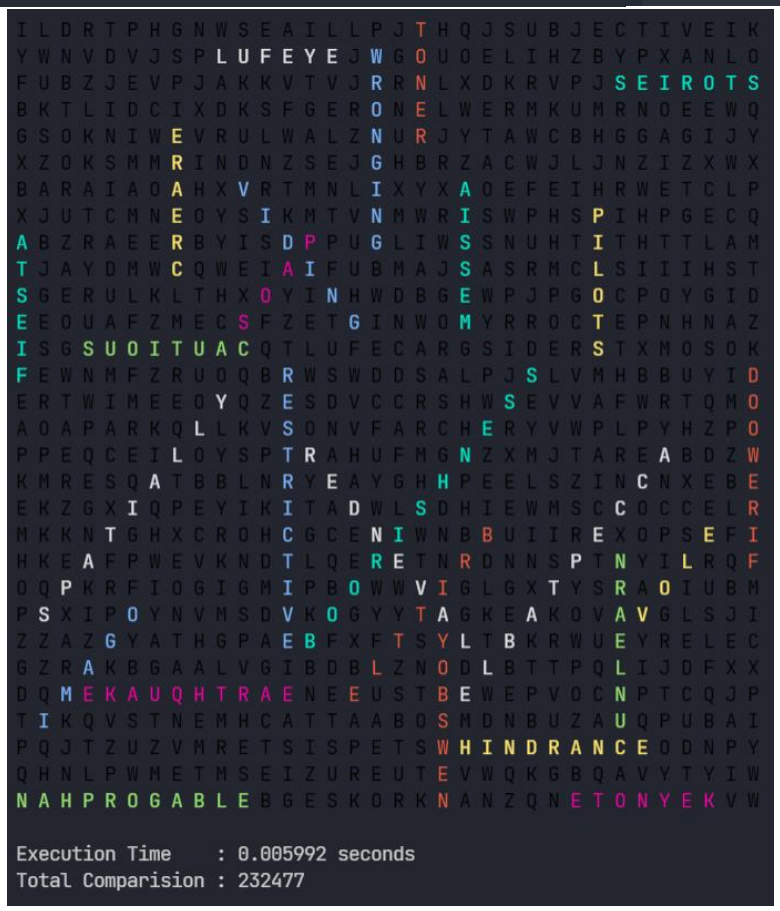
6. tc6_med.txt


I M B E O Q R G V Y W Z C S Z I J W N Z
J W M Q E G A X X J Q Z X K M I I X Q A
H A K M Z K K W W A J K K B P R Q G R Z
I Y E B G I F M R F U D R E F O R L N L
D A P C W S T P L D R T O F L D J G R D
Q J B G J S E D R F T P C S A T V G O P
Y V J X Q Q E S L O M J Z K O P D S S U
W D Q Y T B Z Z Q L B E Z B H T V V Y C
V Z C B S F E E I F H F J S Q Y I Z A M
V A O T W L D W M Z J F K O S C U I Q K
Z A N F G X F Z V S Y K W Q P G B O U H
X G X Y U V J U F A N Y N K I R O R U B
M V D A H Z K C Q S N L P M M Q M N F Y
F P T V O F J C V B O D O J W T E I I K
K E T X R H V U O H M P V X F R U G R B
N D W B T H H H U C P M Y R T B Q I O I
X Q A Q U H U I N X U D X K M Y F A H Y
K Z K M C D U X H R C N Z J N A G M J S
C X T Y E M Q G M O W K O Y G J Q R H P
C R D T H Y C X J Z J X S N R R T Z R R
B I M R I R I C F T R R I X F D M K P L
S T H K M T I N W O N V U A Z D E M N D
SEDRFTPC
JODOBVCJF
TSWGUHO
UIVDVJR
BWSEZIZ
TXOL
RHMUIU
XYDCOJ

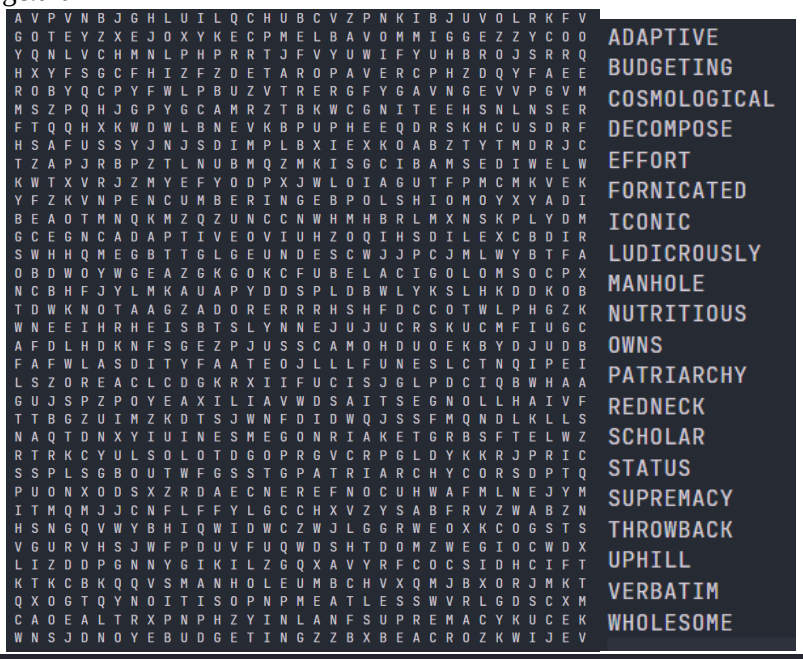

Input your file name: tc6_med.txt
I M B E O Q R G V Y W Z C S Z I J W N Z
J W M Q E G A X X J Q Z X K M I I X Q A
H A K M Z K K W W A J K K B P R Q G R Z
I Y E B G I F M R F U D R E F O R L N L
D A P C W S T P L D R T O F L D J G R D
Q J B G J S E D R F T P C S A T V G O P
Y V J X Q Q E S L O M J Z K O P D S S U
W D Q Y T B Z Z Q L B E Z B H T V V Y C
V Z C B S F E E I F H F J S Q Y I Z A M
V A O T W L D W M Z J F K O S C U I Q K
Z A N F G X F Z V S Y K W Q P G B O U H
X G X Y U V J U F A N Y N K I R O R U B
M V D A H Z K C Q S N L P M M Q M N F Y
F P T V O F J C V B O D O J W T E I I K
K E T X R H V U O H M P V X F R U G R B
N D W B T H H H U C P M Y R T B Q I O I
X Q A Q U H U I N X U D X K M Y F A H Y
K Z K M C D U X H R C N Z J N A G M J S
C X T Y E M Q G M O W K O Y G J Q R H P
C R D T H Y C X J Z J X S N R R T Z R R
B I M R I R I C F T R R I X F D M K P L
S T H K M T I N W O N V U A Z D E M N D
Execution Time : 0.000985 seconds
Total Comparision : 27543

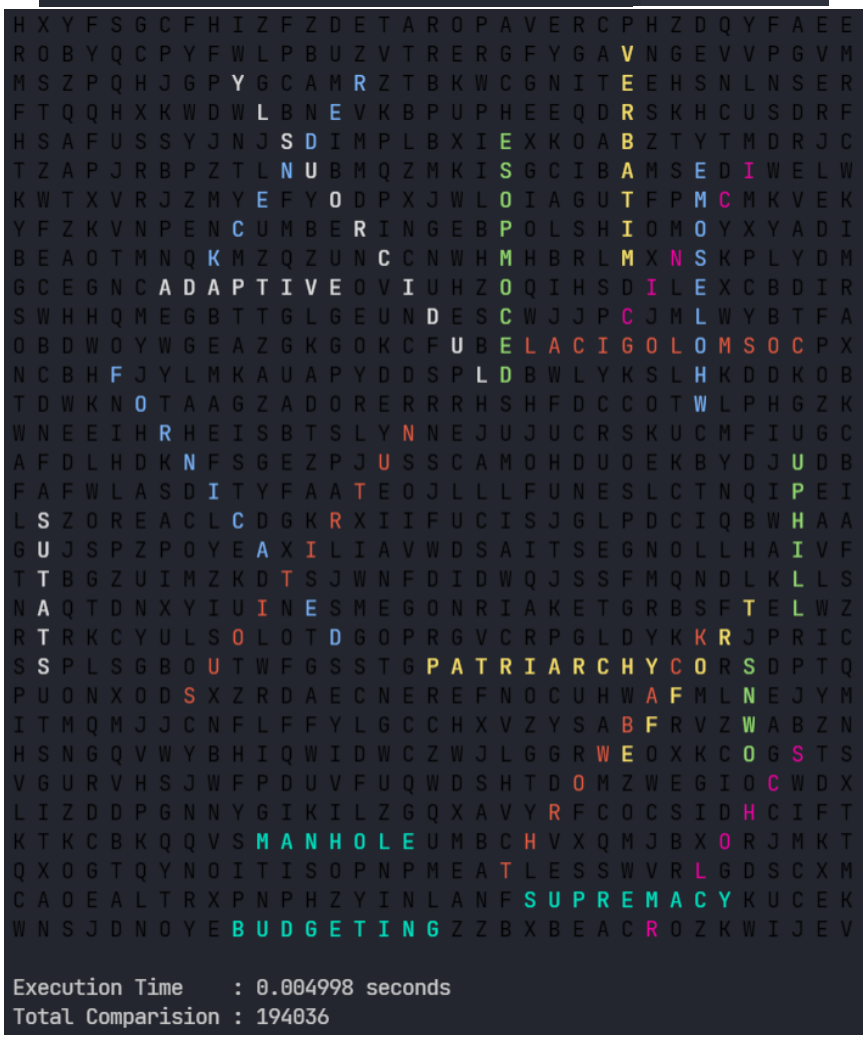
7. tc7_large.txt



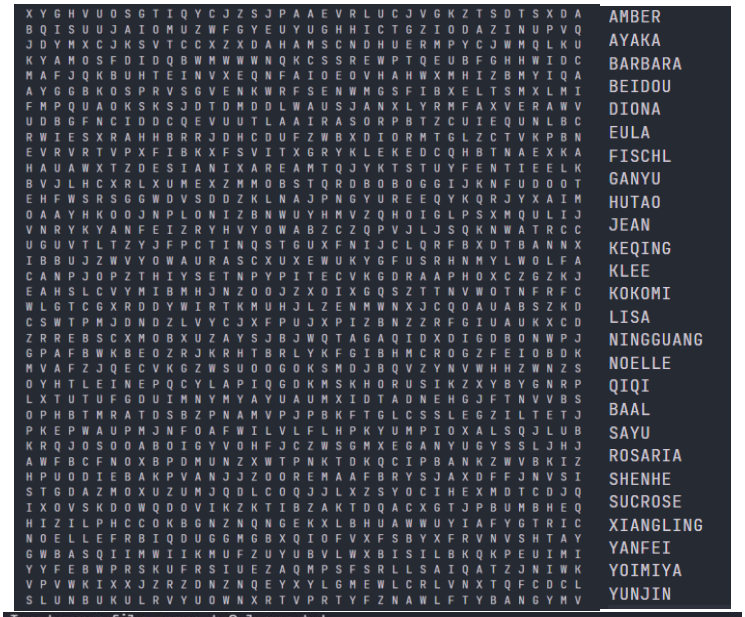


8. tc8_large.txt





9. tc9_large.txt



AMBER
AYAKA
BARBARA
BEIDOU
DIONA
EULA
FISCHL
GANYU
HUTAO
JEAN
KEQING
KLEE
KOKOMI
LISA
NINGGUANG
NOELLE
QIQI
BAAL
SAYU
ROSARIA
SHENHE
SUCROSE
XIANGLING
YANFEI
YOIMIYA
YUNJIN

Input your file name: tc9_large.txt

E. Alamat Drive

Kode bisa dilihat pada repository Github di tautan berikut setelah 26 Januari 2022 pukul 12.30 WIB.

F. Tabel *Checklist*

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 puzzle.	√	