Penyelesaian Word Search Puzzle dengan Algoritma Brute Force

LAPORAN TUGAS KECIL

Diajukan Untuk Memenuhi Tugas Kecil IF2211 Strategi Algoritma Semester II 2021/2022



Oleh:

Jason Kanggara 13520080

PROGRAM STUDI TEKNIK INFORMATIKA SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA INSTITUT TEKNOLOGI BANDUNG 2022

A. Algoritma Brute Force

Algoritma *Brute Force* adalah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu persoalan. Biasanya algoritma *brute force* didasarkan pada pernyataan yang terdapat di persoalan dan definisi/konsep yang dilibatkan. Algoritma *brute force* ini memecahkan persoalan dengan sangat sederhana, langsung, dan jelas caranya. Algoritma *brute force* merupakan algoritma yang dapat menyelesaikan hampir segala macam persoalan karena pendekatannya yang langsung. Salah satunya adalah persoalan pada tugas kecil ini, yaitu *Word Search Puzzle*, yang dapat diselesaikan dengan Algoritma *Brute Force*.

Pada persoalan *Word Search Puzzle*, langkah – langkah algoritma yang digunakan adalah sebagai berikut:

- 1. Menyusun karakter karakter yang merupakan *puzzle* dari file input ke dalam bentuk array dua dimensi dan kata kata ke dalam bentuk list.
- 2. *Looping* setiap kata pada list untuk ditemukan masing masing kata pada array dua dimensi yang berisi karakter.
- 3. Menelusuri dari kiri ke kanan pada kata, bandingkan setiap karakter pada kata dengan karakter yang bersesuaian di dalam array dua dimensi sampai:
 - a. Semua karakter yang dibandingkan cocok (pencarian berhasil), atau
 - b. Ditemukan sebuah ketidakcocokan karakter (pencarian belum berhasil)
- 4. Bila kata belum ditemukan kecocokannya, geser ke elemen berikutnya pada array dua dimensi lalu ulangi langkah ke 3.
- 5. Bila semua elemen sudah dicoba dan masih belum menemukan katanya, ulangi langkah ke 3 dengan arah yang berbeda. (misalnya: perbandingan dari atas ke bawah, dari kiri ke kanan, secara horizontal, dll).

B. Source Code

Source code dibuat dengan bahasa C++. Program yang dibuat berasumsi bahwa karakter tersusun dalam bentuk persegi maupun persegi panjang dan setiap kata yang ingin dicari dipisahkan oleh *newline*.

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <bits/stdc++.h>
#include <time.h>
using namespace std;

#define MAX 100

typedef struct
{
```

```
char contents[MAX][MAX];
  int rowEff;
  int colEff;
} Matrix;
typedef struct
 string patterns[MAX];
 int length;
} List;
#define LENGTH(L) (L).length
#define ListElem(L, i) (L).patterns[i]
#define ROWS(M) (M).rowEff
#define COLS(M) (M).colEff
#define Elem(M, i, j) (M).contents[(i)][(j)]
/*** KONSTRUKTOR ***/
void createMat(Matrix *m)
  for (int i = 0; i < MAX; i++)
    for (int j = 0; j < MAX; j++)
      Elem(*m, i, j) = '#';
void createList(List *1)
  for (int i = 0; i < MAX; i++)
   ListElem(*1, i) = "*";
  }
```

```
/*** FUNGSI/PROSEDUR PENDUKUNG ***/
int getLength(List 1)
 int count = 0;
 for (int i = 0; i < MAX; i++)
    if (ListElem(1, i) != "*")
      count++;
  return count;
void displayList(List 1)
 for (int i = 0; i < getLength(1); i++)</pre>
    cout << ListElem(1, i) << endl;</pre>
int getRow(Matrix m)
  int count = 0;
  for (int i = 0; i < MAX; i++)
    if (Elem(m, i, 0) == '#')
      break;
    count++;
  return count;
int getCol(Matrix m)
```

```
int count = 0;
  for (int j = 0; j < MAX; j++)
    if (Elem(m, 0, j) != '#')
      count++;
  return count;
void displayMat(Matrix m)
  for (int i = 0; i < getRow(m); i++)
    for (int j = 0; j < getCol(m); j++)
      cout << Elem(m, i, j) << " ";</pre>
    cout << endl;</pre>
  }
void readFileBox(string filename, Matrix *m) {
  vector<vector<char> > text;
  ifstream myfile;
  myfile.open(filename.c_str());
  string line;
  while (getline(myfile, line))
    vector<char> textData;
    stringstream lineStream(line);
    char value;
    while (lineStream >> value)
      textData.push_back(value);
```

```
}
    text.push_back(textData);
  int index;
  for (int i = 0; i < text.size(); i++)</pre>
    for (int j = 0; j < text[i].size(); j++)</pre>
      Elem(*m, i, j) = text[i][j];
  }
void readFile(string filename, List *1)
  vector<vector<string> > text;
  ifstream myfile;
  myfile.open(filename.c_str());
  string line;
  while (getline(myfile, line))
    vector<string> textData;
    stringstream lineStream(line);
    string value;
    while (lineStream >> value)
      textData.push_back(value);
    text.push_back(textData);
  int index;
  for (int i = 0; i < text.size(); i++)</pre>
```

```
for (int j = 0; j < text[i].size(); j++)</pre>
      ListElem(*1, i) = text[i][0];
List fixList(List 1) {
  List 1f;
  createList(&lf);
  int idx;
  for (int i = 0; i < getLength(1); i++) {
    if (ListElem(l,i) == "*") {
      idx = i;
    }
  idx = idx + 1;
  int k = 0;
  for (int i = idx; i < getLength(1) + 1; i++) {
   ListElem(lf,k) = ListElem(l, i);
    k++;
  return lf;
/*** PROSEDUR/FUNGSI STRING MATCHING ***/
void PencocokanString(Matrix T, string P, int *perbandinganHuruf)
  int col = getCol(T);
  int row = getRow(T);
  int m = P.length();
  int locRow, locCol;
  int iter;
  bool found;
```

```
found = false;
// Horizontal Matching
if (!found) {
 locRow = 0;
  locCol = 0;
  for (int i = 0; i < row; i++) {
    locCol = 0;
    while (locCol <= col - m && !found) {
      iter = 0;
      while (iter < m && P[iter] == Elem(T, i, locCol + iter)) {</pre>
        iter++;
        *perbandinganHuruf += 1;
      if (iter == m) {
        found = true;
        locRow = i;
        for (int i = 0; i < row; i++) {
          for (int j = 0; j < col; j++) {
            if (i != locRow) {
              Elem(T,i,j) = '-';
            if (i == locRow) {
              if (j < locCol) {</pre>
                Elem(T,i,j) = '-';
              if (j \ge m + locCol) {
                Elem(T, i, j) = '-';
              }
            }
        }
      } else {
        locCol++;
        *perbandinganHuruf += 1;
```

```
// Alternate Horizonal Matching
if (!found) {
  locRow = 0;
  locCol = col - 1;
  for (int i = 0; i < row; i++) {
    locCol = col - 1;
    while (locCol >= m - 1 && !found) {
      iter = 0;
      while (iter < m && P[iter] == Elem(T, i, locCol - iter)) {</pre>
        iter++;
        *perbandinganHuruf += 1;
      if (iter == m) {
        found = true;
        locRow = i;
        for (int i = 0; i < row; i++) {
          for (int j = 0; j < col; j++) {
            if (i != locRow) {
              Elem(T,i,j) = '-';
            if (i == locRow) {
              if (j > locCol) {
                Elem(T,i,j) = '-';
              if (j <= locCol - m) {
                Elem(T,i,j) = '-';
              }
      } else {
        locCol--;
        *perbandinganHuruf += 1;
```

```
// Vertical Matching
if (!found) {
  locCol = 0;
  while (locCol < col) {</pre>
    locRow = 0;
    while (locRow <= row - m && !found) {</pre>
      iter = 0;
      while (iter < m && P[iter] == Elem(T, locRow + iter, locCol)) {</pre>
        iter++;
        *perbandinganHuruf += 1;
      if (iter == m) {
        found = true;
        for (int i = 0; i < row; i++) {
          for (int j = 0; j < col; j++) {
            if (j != locCol) {
               Elem(T,i,j) = '-';
            if (j == locCol) {
              if (i < locRow) {</pre>
                 Elem(T,i,j) = '-';
              if (i >= m + locRow) {
                 Elem(T,i,j) = '-';
      } else {
        locRow++;
```

```
*perbandinganHuruf += 1;
      }
    locCol++;
  }
// Alternate Vertical Matching
if (!found) {
  locCol = 0;
  locRow = row - 1;
  while (locCol < col) {
    locRow = row - 1;
    while (locRow >= m - 1 && !found) {
      iter = 0;
      while (iter < m && P[iter] == Elem(T, locRow - iter, locCol)) {</pre>
        iter++;
        *perbandinganHuruf += 1;
      if (iter == m) {
        found = true;
        for (int i = 0; i < row; i++) {
          for (int j = 0; j < col; j++) {
            if (j != locCol) {
              Elem(T,i,j) = '-';
            if (j == locCol) {
              if (i > locRow) {
                Elem(T,i,j) = '-';
              if (i <= locRow - m) {</pre>
                Elem(T,i,j) = '-';
```

```
} else {
          locRow--;
          *perbandinganHuruf += 1;
      locCol++;
    }
  // Row Major Diagonal Matching
  if (!found) {
    locRow = 0;
    locCol = 0;
    while (locRow < row) {</pre>
      locCol = 0;
      while (locCol <= col - m && !found) {
        iter = 0;
        while (iter < m && P[iter] == Elem(T, locRow + iter, locCol +</pre>
iter)) {
          iter++;
          *perbandinganHuruf += 1;
        if (iter == m) {
          found = true;
          for (int i = 0; i < row; i++) {
             for (int j = 0; j < col; j++) {
              if (i < locRow) {</pre>
                 Elem(T,i,j) = '-';
               if (j < locCol) {</pre>
                 Elem(T,i,j) = '-';
              if (i >= locRow + m) {
```

```
Elem(T,i,j) = '-';
              if (j >= locCol + m) {
                Elem(T,i,j) = '-';
              for (int k = 0; k < m; k++) {
                if (i > locRow + k & j <= locCol + k) {
                  Elem(T,i,j) = '-';
                if (i \leftarrow locRow + k && j > locCol + k) {
                   Elem(T,i,j) = '-';
        } else {
          locCol++;
          *perbandinganHuruf += 1;
      locRow++;
  // Alternate Row Major Diagonal Matching
  if (!found) {
    locRow = 0;
    locCol = col - 1;
    while (locRow < row) {</pre>
      locCol = col - 1;
      while (locCol >= m - 1 && !found) {
        iter = 0;
        while (iter < m && P[iter] == Elem(T, locRow + iter, locCol -
iter)) {
          iter++;
          *perbandinganHuruf += 1;
```

```
if (iter == m) {
    found = true;
    for (int i = 0; i < row; i++) {
      for (int j = col - 1; j >= 0; j--) {
        if (i < locRow) {
          Elem(T,i,j) = '-';
        if (j > locCol) {
          Elem(T,i,j) = '-';
        if (j <= locCol - m) {</pre>
          Elem(T,i,j) = '-';
        if (i >= locRow + m) {
          Elem(T,i,j) = '-';
        for (int k = 0; k < m; k++) {
          if (i > locRow + k \&\& j == locCol - k) {
            Elem(T,i,j) = '-';
          if (i == locRow + k \&\& j < locCol - k) {
            Elem(T,i,j) = '-';
          }
    }
  } else {
    locCol--;
    *perbandinganHuruf += 1;
locRow++;
```

```
// Column Major Diagonal Matching
  if (!found) {
    locRow = 0;
    locCol = 0;
    while (locCol < col) {
      locRow = 0;
      while (locRow <= row - m && !found) {</pre>
        iter = 0;
        while (iter < m && P[iter] == Elem(T, locRow + iter, locCol +</pre>
iter)) {
          iter++;
          *perbandinganHuruf += 1;
        }
        if (iter == m) {
          found = true;
           for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
              if (i < locRow) {</pre>
                 Elem(T,i,j) = '-';
               if (j < locCol) {</pre>
                 Elem(T,i,j) = '-';
               if (i >= locRow + m) {
                 Elem(T,i,j) = '-';
               if (j \ge locCol + m) {
                 Elem(T,i,j) = '-';
               for (int k = 0; k < m; k++) {
                 if (i > locRow + k \&\& j <= locCol + k) {
                   Elem(T,i,j) = '-';
                 }
                 if (i \leftarrow locRow + k && j > locCol + k) {
                   Elem(T,i,j) = '-';
```

```
}
        } else {
          locRow++;
          *perbandinganHuruf += 1;
      locCol++;
    }
  // Alternate Column Major Diagonal Matching
  if (!found) {
    locRow = row - 1;
    locCol = 0;
    while (locCol < col) {</pre>
      locRow = row - 1;
      while (locRow >= m - 1 && !found) {
        iter = 0;
        while (iter < m && P[iter] == Elem(T, locRow - iter, locCol +</pre>
iter)) {
          iter++;
          *perbandinganHuruf += 1;
        if (iter == m) {
          found = true;
          for (int i = 0; i < row; i++) {
            for (int j = col - 1; j >= 0; j--) {
              if (j < locCol) {</pre>
                 Elem(T,i,j) = '-';
              if (i > locRow) {
                Elem(T,i,j) = '-';
              if (j >= locCol + m) {
                Elem(T,i,j) = '-';
```

```
if (i <= locRow - m) {
                Elem(T,i,j) = '-';
              for (int k = 0; k < m; k++) {
                if (i < locRow - k && j <= locCol + k) {
                  Elem(T,i,j) = '-';
                if (i == locRow - k && j > locCol + k) {
                  Elem(T,i,j) = '-';
        } else {
          locRow--;
          *perbandinganHuruf += 1;
      locCol++;
    }
  // Diagonal checking from bottom right
  if (!found) {
    locRow = row - 1;
    locCol = col - 1;
    while (locCol >= 0) {
      locRow = row - 1;
      while (locRow >= m - 1 && !found) {
        iter = 0;
        while (iter < m && P[iter] == Elem(T, locRow - iter, locCol -</pre>
iter)) {
          iter++;
          *perbandinganHuruf += 1;
        if (iter == m) {
          found = true;
```

```
for (int i = 0; i < row; i++) {
           for (int j = col - 1; j >= 0; j--) {
             if (j > locCol) {
               Elem(T,i,j) = '-';
             if (i > locRow) {
               Elem(T,i,j) = '-';
             if (j <= locCol - m) {
               Elem(T,i,j) = '-';
             if (i <= locRow - m) {</pre>
               Elem(T,i,j) = '-';
             for (int k = 0; k < m; k++) {
               if (i < locRow - k \&\& j == locCol - k) {
                 Elem(T,i,j) = '-';
               if (i == locRow - k \&\& j < locCol - k) {
                 Elem(T,i,j) = '-';
               }
          }
       } else {
         locRow--;
         *perbandinganHuruf += 1;
     locCol--;
    }
  }
 if (found) {
   displayMat(T);
    cout << endl;</pre>
                  -----" <<
    cout << "----
endl << endl;</pre>
```

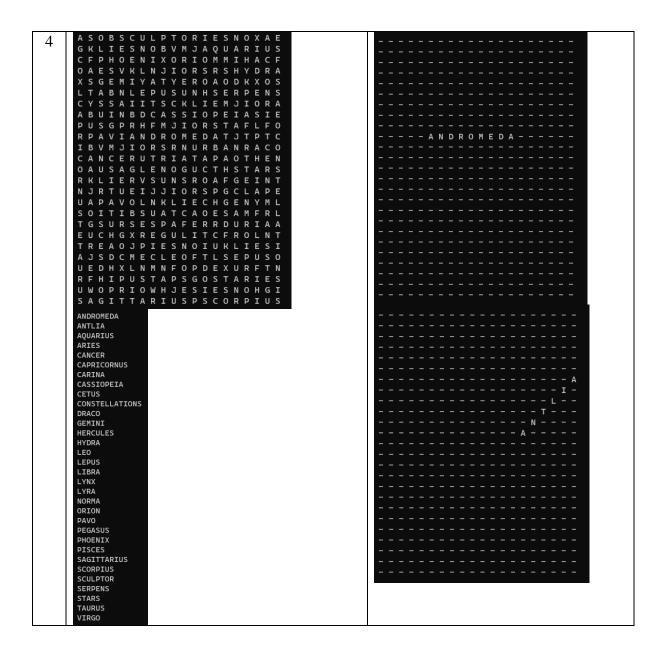
```
/*** MAIN FUNCTION ***/
int main()
 Matrix m;
 List 1;
 int perbandinganHuruf = 0;
 createList(&1);
 createMat(&m);
 string namaFile;
 cout << "Masukkan nama file (*.txt) : ";</pre>
 cin >> namaFile;
 readFile("../test/" + namaFile + ".txt", &1);
 readFileBox("../test/" + namaFile + ".txt", &m);
 1 = fixList(1);
 cout << "-----" <<
end1;
 displayMat(m);
 cout << "-----" <<
endl;
 displayList(1);
 cout << "-----" << endl
<< endl;
 clock_t start = clock();
```

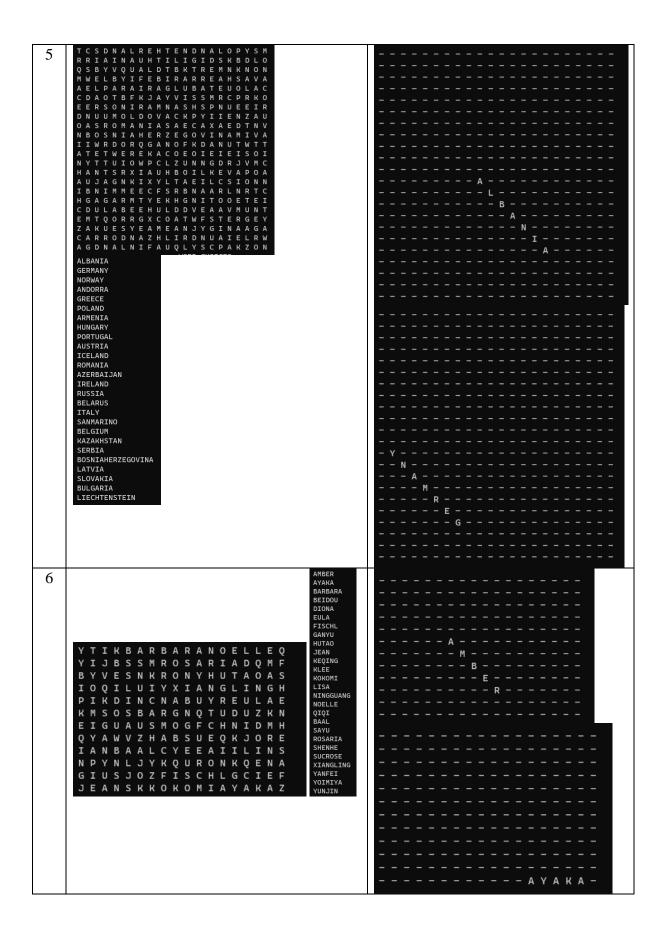
C. Hasil Program

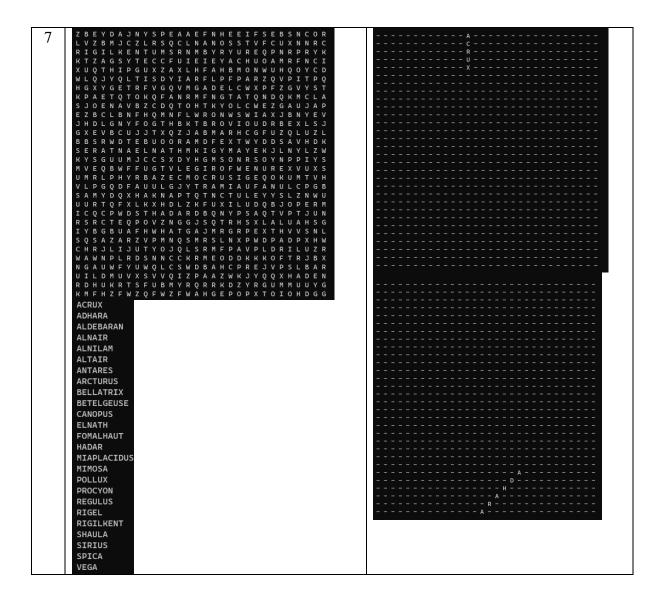
N	Input	Output	
0			
1	>> D\k\s\s\t\src on main ./a Masukkan nama file (*.txt) : small2		

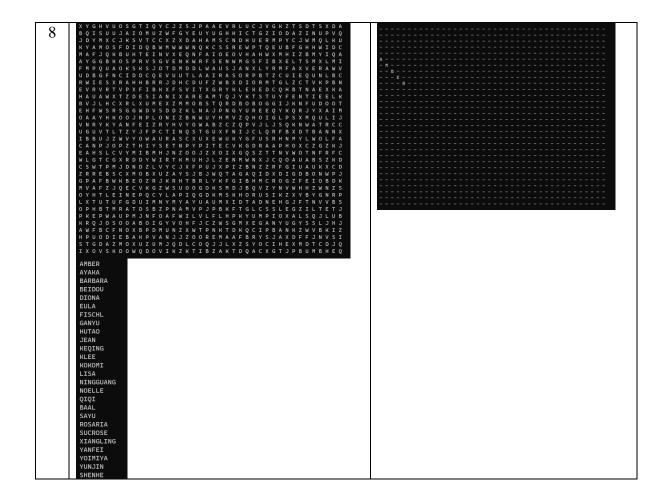
		J
		T
2	Masukkan nama file (*.txt): small1	Total number of letter comparison: 1972 times Execution Time of the program with Result Printing: 0.1809999035
		Total number of letter comparison: 2442 times Execution Time of the program with Result Printing : 0.2930000126s

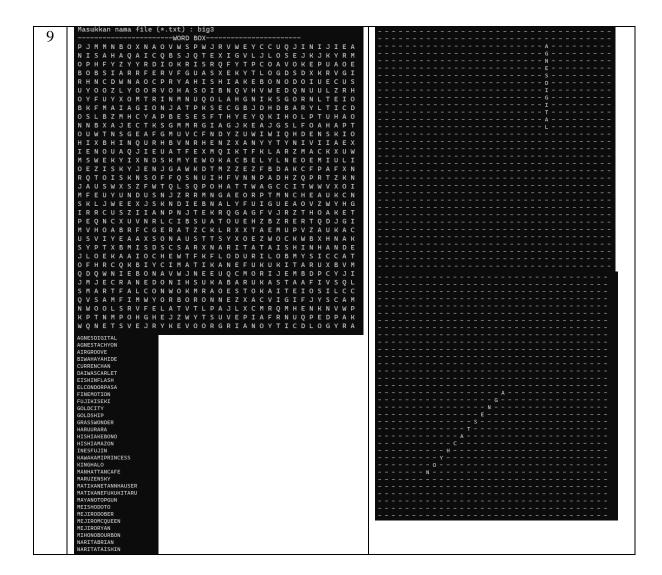
3	A A A A A A A A A A D		
	A A A A A A A A A A L		
	AABADAADAAR	BAD	
	A A A A A A A A A A O A A A A A A A A A		
	AAAHELLOAAAA		
	T A A A A A A A A A A A A A A A A A A A		
	AHAAAAGAAAOA		
	N A G A A A O A A A R A I A A I A A O A A A N A		
	G A A A R A D A A A I A		
H	H A S L E E P A A A N A		
	T A A A A A A A G A		
	BAD		
	HELLO		
	GOOD		
	MORNING		
	NIGHT DAY		
	WORLD		
F	RIGHT		
9	SLEEP		
		G	
		0	
		0	
		D	
		N	
		ï	
		_	
		G	
		н	
		T	











D. Link Source Code

 $\underline{https://drive.google.com/drive/folders/1NE5kXoWwCTtJFjtdc6me_HjMPpOn_bsd?usp=sharing}$

E. Tabel Cek List

Poin		Ya	Tidak
1.	Program berhasil dikompilasi tanpa kesalahan (no syntax error).	V	
2.	Program berhasil running.	V	
3.	Program dapat membaca file masukan dan menuliskan luaran.	V	
4.	Program berhasil menemukan semua kata di dalam puzzle.	V	