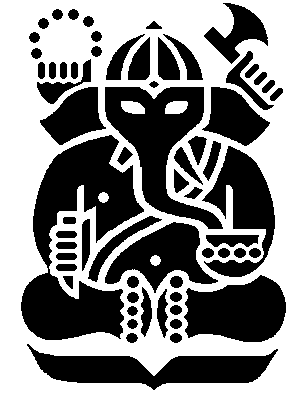
**Penyelesaian *Word Search Puzzle* dengan Algoritma *Brute Force***

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 adalahpersoalan 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:
   1. Semua karakter yang dibandingkan cocok (pencarian berhasil), atau
   2. 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 \*l)

{

  for (int i = 0; i < MAX; i++)

  {

    ListElem(\*l, i) = "\*";

  }

}

/\*\*\* FUNGSI/PROSEDUR PENDUKUNG \*\*\*/

int getLength(List l)

{

  int count = 0;

  for (int i = 0; i < MAX; i++)

  {

    if (ListElem(l, i) != "\*")

    {

      count++;

    }

  }

  return count;

}

void displayList(List l)

{

  for (int i = 0; i < getLength(l); i++)

  {

    cout << ListElem(l, i) << endl;

  }

}

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) << " ";

    }

    cout << endl;

  }

}

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++)

  {

    for (int j = 0; j < text[i].size(); j++)

    {

      Elem(\*m, i, j) = text[i][j];

    }

  }

}

void readFile(string filename, List \*l)

{

  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++)

  {

    for (int j = 0; j < text[i].size(); j++)

    {

      ListElem(\*l, i) = text[i][0];

    }

  }

}

List fixList(List l) {

  List lf;

  createList(&lf);

  int idx;

  for (int i = 0; i < getLength(l); i++) {

    if (ListElem(l,i) == "\*") {

      idx = i;

    }

  }

  idx = idx + 1;

  int k = 0;

  for (int i = idx; i < getLength(l) + 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)) {

          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 >= 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)) {

          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) {

      locRow = 0;

      while (locRow <= row - m && !found) {

        iter = 0;

        while (iter < m && P[iter] == Elem(T, locRow + iter, locCol)) {

          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 >= 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)) {

          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) {

                  Elem(T,i,j) = '-';

                }

              }

            }

          }

        } else {

          locRow--;

          \*perbandinganHuruf += 1;

        }

      }

      locCol++;

    }

  }

  // Row Major Diagonal Matching

  if (!found) {

    locRow = 0;

    locCol = 0;

    while (locRow < row) {

      locCol = 0;

      while (locCol <= col - m && !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 = 0; j < col; j++) {

              if (i < locRow) {

                Elem(T,i,j) = '-';

              }

              if (j < locCol) {

                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 <= 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) {

      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) {

                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) {

        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 = 0; j < col; j++) {

              if (i < locRow) {

                Elem(T,i,j) = '-';

              }

              if (j < locCol) {

                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 <= 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) {

      locRow = row - 1;

      while (locRow >= 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 (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) {

                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 - 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) {

                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;

    cout << "-----------------------ENDLINE-----------------------" << endl << endl;

  }

}

/\*\*\* MAIN FUNCTION \*\*\*/

int main()

{

  Matrix m;

  List l;

  int perbandinganHuruf = 0;

  createList(&l);

  createMat(&m);

  string namaFile;

  cout << "Masukkan nama file (\*.txt) : ";

  cin >> namaFile;

  readFile("../test/" + namaFile + ".txt", &l);

  readFileBox("../test/" + namaFile + ".txt", &m);

  l = fixList(l);

  cout << "-----------------------WORD BOX-----------------------" << endl;

  displayMat(m);

  cout << "-----------------------WORD CHOICES-----------------------" << endl;

  displayList(l);

  cout << "-----------------------RESULT-----------------------" << endl << endl;

  clock\_t start = clock();

  for (int i = 0; i < getLength(l); i++) {

    PencocokanString(m, ListElem(l,i), &perbandinganHuruf);

  }

  cout << "-----------------------CONCLUSION-----------------------" << endl << endl;

  cout << "Total number of letter comparison: " << perbandinganHuruf << " times" << endl;

  cout << "Execution Time of the program with Result Printing : ";

  cout.precision(10);

  cout << fixed << float(clock() - start) / CLOCKS\_PER\_SEC << "s" << endl;

  return 0;

}

**C. Hasil Program**

|  |  |  |
| --- | --- | --- |
| **No** | **Input** | **Output** |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  |  |
| 9 |  |  |

**D. Link Source Code**

<https://github.com/jasonk19/STIMA-Tucil1.git>

**E. Tabel Cek List**

|  |  |  |
| --- | --- | --- |
| 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. | √ |  |