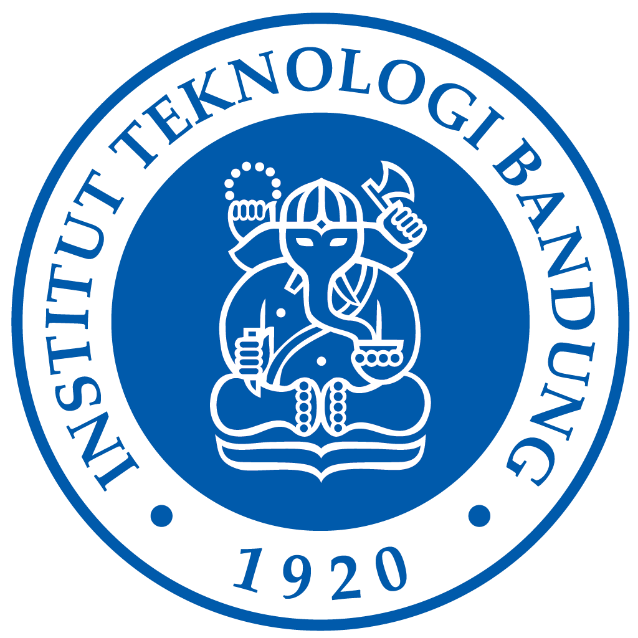
**IF2211 STRATEGI ALGORITMA**

LAPORAN TUGAS KECIL 1

Penyelesaian Word Search Puzzle dengan Algoritma Brute Force



Oleh:

**Tri Sulton Adila 13520033**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2022**

1. Langkah-Langkah Algoritma Brute Force
2. Telusuri setiap huruf yang terdapat pada puzzle mulai dari baris dan kolom pertama sampai baris dan kolom terakhir.
3. Periksa huruf yang terdapat pada puzzle dengan setiap huruf pertama pattern.
4. Apabila huruf puzzle sama dengan huruf awal suatu pattern, lakukan pemeriksaan huruf selanjutnya dalam delapan arah, yaitu atas, kanan atas, kanan, kanan bawah, bawah, kiri bawah, kiri, dan kiri atas.
5. Untuk setiap pemeriksaan dalam delapan arah, periksa terlebih dahulu apakah terdapat cukup sel dalam arah yang akan diperiksa untuk memuat pattern.
6. Apabila jumlah sel cukup, lakukan pemeriksaan untuk huruf berikutnya dari pattern dengan huruf pada sel selanjutnya dari arah yang diperiksa dalam puzzle.
7. Apabila semua huruf pada puzzle dalam arah yang diperiksa cocok dengan huruf pada pattern artinya pattern tersebut telah berhasil ditemukan
8. Apabila ada satu huruf yang tidak cocok, lakukan pemeriksaan pada arah selanjutnya.
9. Apabila tidak ada kecocokan pada setiap arah, kembali ke langkah dua untuk pattern selanjutnya.
10. Source Code Program
11. main.cpp

#include <iostream>

#include <fstream>

#include <string>

#include <chrono>

#include "point.cpp"

using namespace std;

using namespace std::chrono;

void printResult(char \*\*puzzle, int rowSize, int columnSize, POINT \*words, int wordNumber) {

for(int k = 0; k < wordNumber; k++) {

int l;

POINT p = words[k];

switch (DIRECTION(p))

{

case 1:

for(int i = 0; i < rowSize; i++) {

for(int j = 0; j < columnSize; j++) {

if(i <= ROW(p) && i > ROW(p) - WORDLENGTH(p) && j == COLUMN(p)) {

cout << puzzle[i][j];

} else {

cout << "-";

}

if(j == columnSize - 1) {

cout << endl;

} else {

cout << " ";

}

}

}

break;

case 2:

l = WORDLENGTH(p);

for(int i = 0; i < rowSize; i++) {

for(int j = 0; j < columnSize; j++) {

if(l > 0 && i == ROW(p) - l + 1 && j == COLUMN(p) + l - 1) {

l--;

cout << puzzle[i][j];

} else {

cout << "-";

}

if(j == columnSize - 1) {

cout << endl;

} else {

cout << " ";

}

}

}

break;

case 3:

for(int i = 0; i < rowSize; i++) {

for(int j = 0; j < columnSize; j++) {

if(i == ROW(p) && j >= COLUMN(p) && j < COLUMN(p) + WORDLENGTH(p)) {

cout << puzzle[i][j];

} else {

cout << "-";

}

if(j == columnSize - 1) {

cout << endl;

} else {

cout << " ";

}

}

}

break;

case 4:

l = 0;

for(int i = 0; i < rowSize; i++) {

for(int j = 0; j < columnSize; j++) {

if(l < WORDLENGTH(p) && i == ROW(p) + l && j == COLUMN(p) + l) {

l++;

cout << puzzle[i][j];

} else {

cout << "-";

}

if(j == columnSize - 1) {

cout << endl;

} else {

cout << " ";

}

}

}

break;

case 5:

for(int i = 0; i < rowSize; i++) {

for(int j = 0; j < columnSize; j++) {

if(i >= ROW(p) && i < ROW(p) + WORDLENGTH(p) && j == COLUMN(p)) {

cout << puzzle[i][j];

} else {

cout << "-";

}

if(j == columnSize - 1) {

cout << endl;

} else {

cout << " ";

}

}

}

break;

case 6:

l = 0;

for(int i = 0; i < rowSize; i++) {

for(int j = 0; j < columnSize; j++) {

if(l < WORDLENGTH(p) && i == ROW(p) + l && j == COLUMN(p) - l) {

l++;

cout << puzzle[i][j];

} else {

cout << "-";

}

if(j == columnSize - 1) {

cout << endl;

} else {

cout << " ";

}

}

}

break;

case 7:

for(int i = 0; i < rowSize; i++) {

for(int j = 0; j < columnSize; j++) {

if(i == ROW(p) && j <= COLUMN(p) && j > COLUMN(p) - WORDLENGTH(p)) {

cout << puzzle[i][j];

} else {

cout << "-";

}

if(j == columnSize - 1) {

cout << endl;

} else {

cout << " ";

}

}

}

break;

case 8:

l = WORDLENGTH(p);

for(int i = 0; i < rowSize; i++) {

for(int j = 0; j < columnSize; j++) {

if(l > 0 && i == ROW(p) - l + 1 && j == COLUMN(p) - l + 1) {

l--;

cout << puzzle[i][j];

} else {

cout << "-";

}

if(j == columnSize - 1) {

cout << endl;

} else {

cout << " ";

}

}

}

break;

default:

break;

}

cout << endl;

}

}

void loadData(string nameFile, char \*\*puzzle, int rowSize, int columnSize, string\* words) {

char ch;

int countEnter = 0;

string word = "";

int i = 0;

int j = 0;

int k = 0;

fstream inFile2("../test/"+nameFile, fstream::in);

while(inFile2 >> noskipws >> ch) {

if(countEnter != 2) {

if(ch == '\n') {

countEnter++;

i++;

j = 0;

} else {

if(ch != ' ') {

puzzle[i][j] = ch;

j++;

}

countEnter = 0;

}

} else {

if(ch != '\n') {

if(ch != ' ') {

word = word + ch;

}

} else {

words[k] = word;

k++;

word = "";

}

}

}

if(word != "") {

words[k] = word;

}

}

void getPuzzleSize(string nameFile, int &rowSize, int &columnSize, int &wordNumber) {

char ch;

int countEnter = 0;

bool stopCountColumn = false;

fstream inFile("../test/"+nameFile, fstream::in);

while(inFile >> noskipws >> ch) {

if(countEnter != 2) {

if(ch == '\n') {

countEnter++;

rowSize++;

stopCountColumn = true;

} else {

if(!stopCountColumn && ch != ' ') {

columnSize++;

}

countEnter = 0;

}

} else {

if(ch == '\n') {

wordNumber++;

}

}

}

}

int main() {

string nameFile;

cout << "Masukkan nama file: ";

cin >> nameFile;

int rowSize = -1;

int columnSize = 0;

int wordNumber = 1;

getPuzzleSize(nameFile, rowSize, columnSize, wordNumber);

char\*\* puzzle = new char\*[rowSize];

for(int i = 0; i < rowSize; i++) {

puzzle[i] = new char[columnSize];

}

string words[wordNumber];

loadData(nameFile, puzzle, rowSize, columnSize, words);

POINT coordinateDirection[wordNumber];

int wordFound = 0;

int compareCount = 0;

auto start = chrono::steady\_clock::now();

for(int i = 0; i < rowSize; i++) {

for(int j = 0; j < columnSize; j++) {

for(int k = 0; k < wordNumber; k++) {

compareCount++;

if(puzzle[i][j] == words[k][0] && words[k] != "") {

bool found = false;

bool finish = false;

while(!found && !finish) {

int wordLength = words[k].length();

// pemeriksaan ke atas

if(i >= wordLength - 1) {

int l = 1;

while(l < wordLength && words[k][l] == puzzle[i - l][j]) {

l++;

compareCount++;

}

if(l == wordLength) {

coordinateDirection[wordFound] = MakePOINT(i, j, 1, wordLength);

wordFound++;

words[k] = "";

found = true;

}

}

// pemeriksaan ke kanan atas

if(j <= columnSize - wordLength && i >= wordLength - 1) {

int l = 1;

while(l < wordLength && words[k][l] == puzzle[i - l][j + l]) {

l++;

compareCount++;

}

if(l == wordLength) {

coordinateDirection[wordFound] = MakePOINT(i, j, 2, wordLength);

wordFound++;

words[k] = "";

found = true;

}

}

// pemeriksaan ke kanan

if(j <= columnSize - wordLength) {

// cout << words[k] << " sedang diperiksa ke kanan" << endl;

int l = 1;

while(l < wordLength && words[k][l] == puzzle[i][j + l]) {

l++;

compareCount++;

}

if(l == wordLength) {

coordinateDirection[wordFound] = MakePOINT(i, j, 3, wordLength);

wordFound++;

words[k] = "";

found = true;

}

}

// pemeriksaan ke kanan bawah

if(j <= columnSize - wordLength && i <= rowSize - wordLength) {

int l = 1;

while(l < wordLength && words[k][l] == puzzle[i + l][j + l]) {

l++;

compareCount++;

}

if(l == wordLength) {

coordinateDirection[wordFound] = MakePOINT(i, j, 4, wordLength);

wordFound++;

words[k] = "";

found = true;

}

}

// pemeriksaan ke bawah

if(i <= rowSize - wordLength) {

int l = 1;

while(l < wordLength && words[k][l] == puzzle[i + l][j]) {

l++;

compareCount++;

}

if(l == wordLength) {

coordinateDirection[wordFound] = MakePOINT(i, j, 5, wordLength);

wordFound++;

words[k] = "";

found = true;

}

}

// pemeriksaan ke kiri bawah

if(i <= rowSize - wordLength && j >= wordLength - 1) {

int l = 1;

while(l < wordLength && words[k][l] == puzzle[i + l][j - l]) {

l++;

compareCount++;

}

if(l == wordLength) {

coordinateDirection[wordFound] = MakePOINT(i, j, 6, wordLength);

wordFound++;

words[k] = "";

found = true;

}

}

// pemeriksaan ke kiri

if(j >= wordLength - 1) {

int l = 1;

while(l < wordLength && words[k][l] == puzzle[i][j - l]) {

l++;

compareCount++;

}

if(l == wordLength) {

coordinateDirection[wordFound] = MakePOINT(i, j, 7, wordLength);

wordFound++;

words[k] = "";

found = true;

}

}

// pemeriksaan ke kiri atas

if(j >= wordLength - 1 && i >= wordLength - 1) {

int l = 1;

while(l < wordLength && words[k][l] == puzzle[i - l][j - l]) {

l++;

compareCount++;

}

if(l == wordLength) {

coordinateDirection[wordFound] = MakePOINT(i, j, 8, wordLength);

wordFound++;

words[k] = "";

found = true;

}

}

finish = true;

}

}

}

}

}

auto end = chrono::steady\_clock::now();

printResult(puzzle, rowSize, columnSize, coordinateDirection, wordNumber);

cout<<chrono::duration\_cast<chrono::nanoseconds>(end-start).count()<<" nanoseconds"<<endl;

cout<<chrono::duration\_cast<chrono::microseconds>(end-start).count()<< " microseconds" << endl;

cout << "Jumlah perbandingan huruf : " << compareCount << endl;

}

1. point.cpp

#include "point.h"

POINT MakePOINT (int i, int j, int direction, int length) {

POINT p;

ROW(p) = i;

COLUMN(p) = j;

DIRECTION(p) = direction;

WORDLENGTH(p) = length;

return p;

}

1. point.h

#ifndef POINT\_H

#define POINT\_H

typedef struct {

int i;

int j;

int direction;

int wordLength;

} POINT;

#define ROW(P) (P).i

#define COLUMN(P) (P).j

#define DIRECTION(P) (P).direction

#define WORDLENGTH(P) (P).wordLength

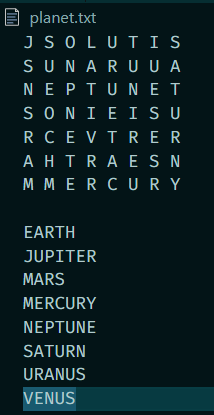
POINT MakePOINT (int i, int j, int direction, int length);

#endif

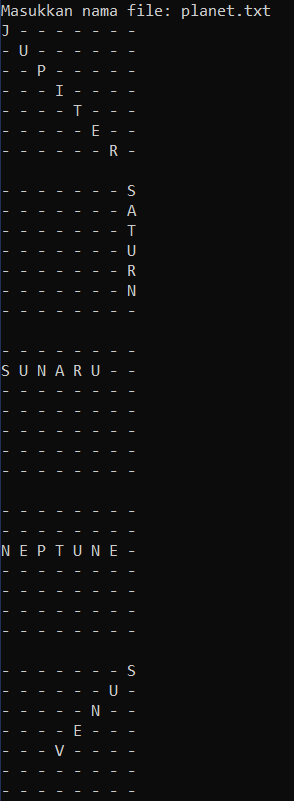
1. Screenshoot Hasil Uji

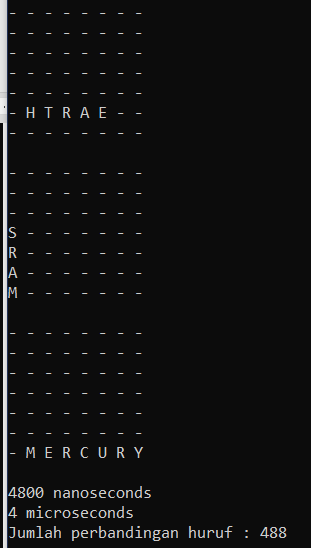
small 14x12 hingga 18x16, lalu medium 20x18 hingga 24x22, dan large 32x30 hingga 36x34.

1. Uji puzzle ukuran 8 x 8

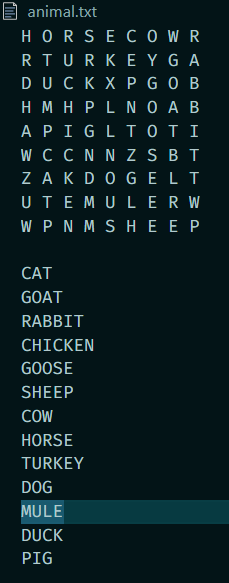


output:

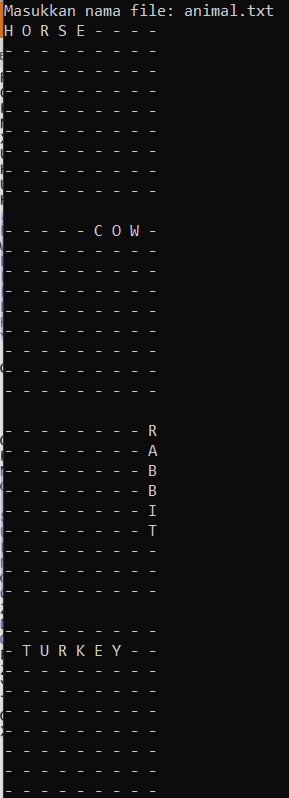


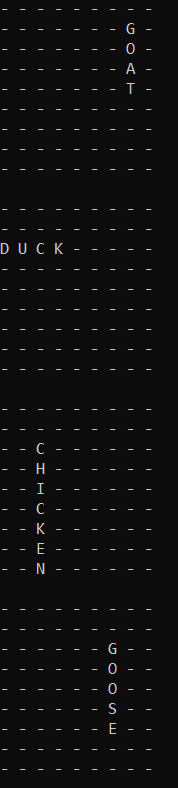


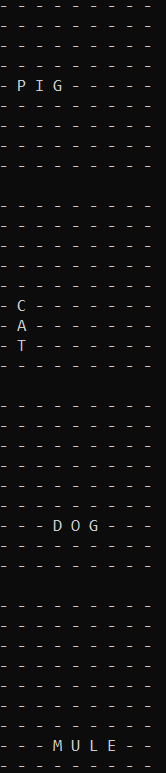
1. Uji puzzle ukuran 9 x 9

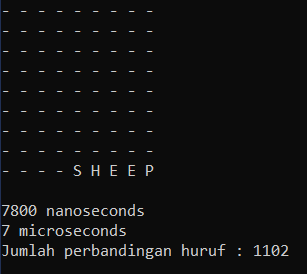


output:

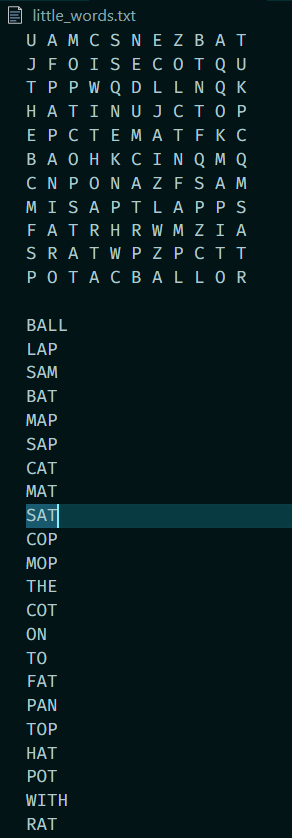




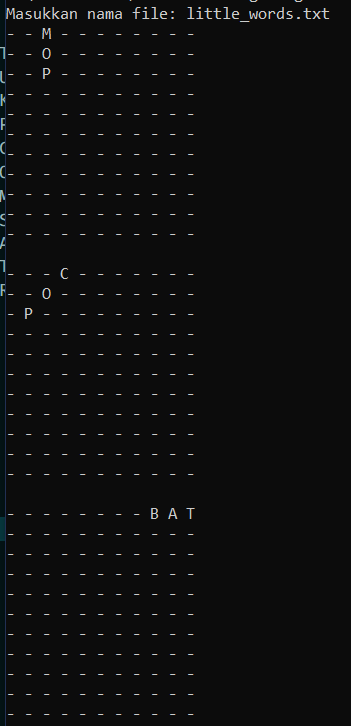




1. Uji puzzle ukuran 11 x 11



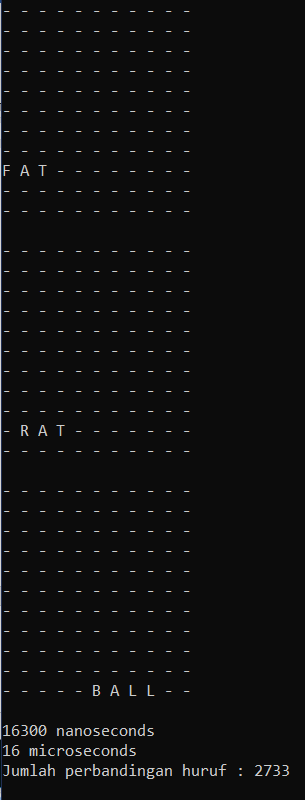
output:



.

.

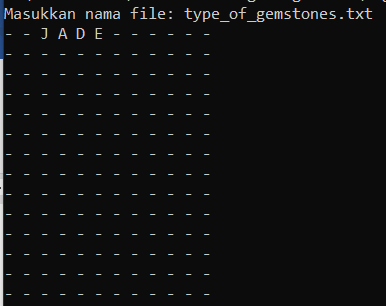
dst

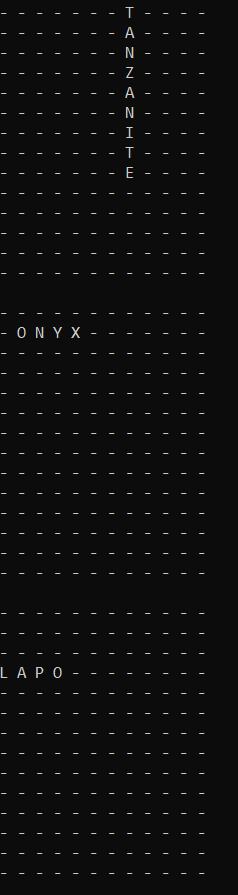


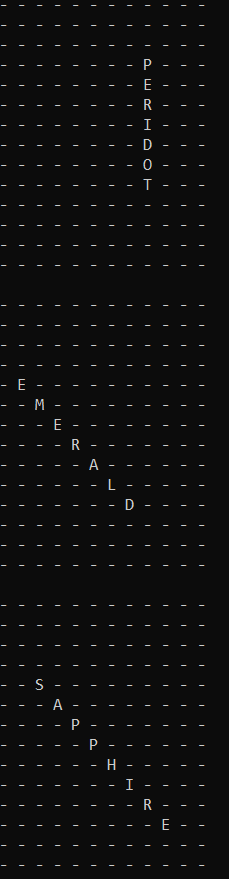
1. Uji puzzle ukuran 14 x 14

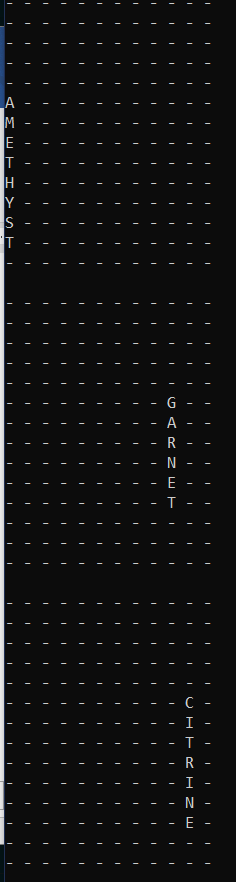


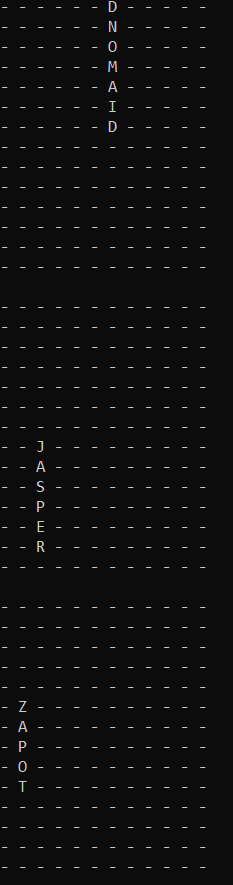
output

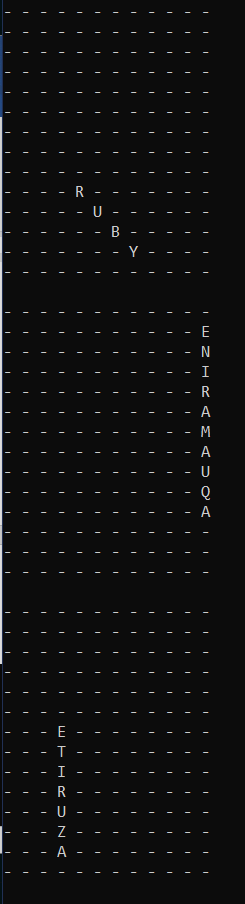


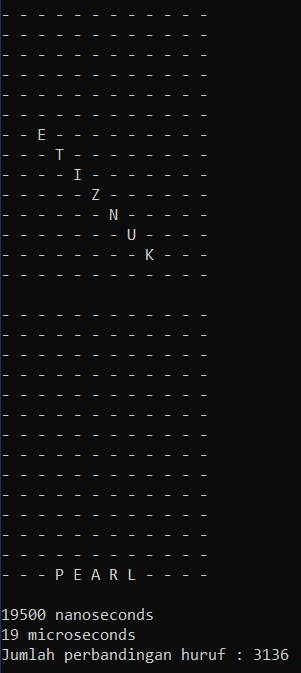






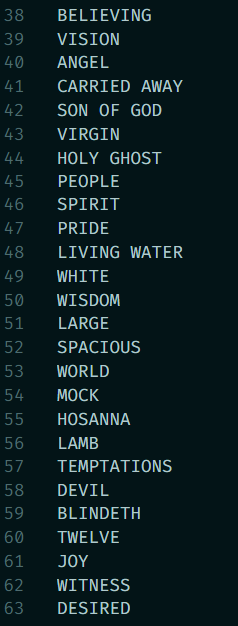






1. Uji puzzle ukuran 20 x 20

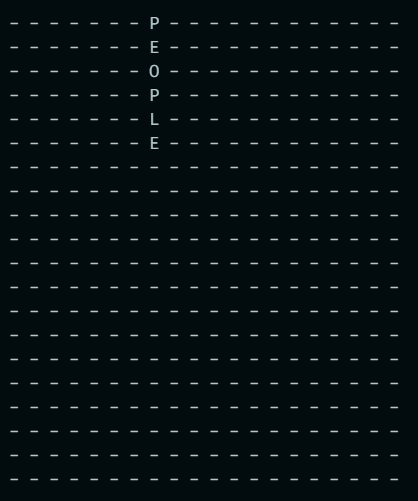




output:







.

.

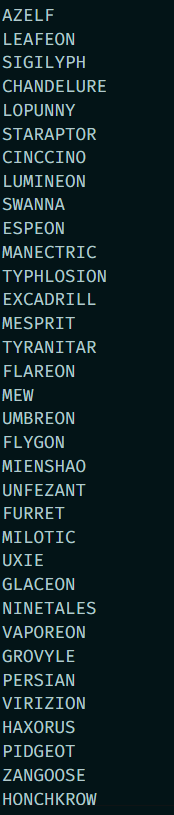
.

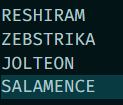
dst



1. Uji puzzle ukuran 21 x 21





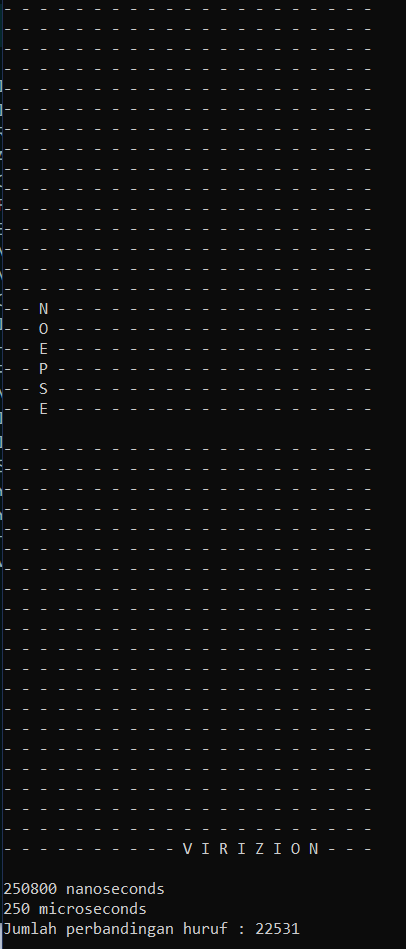




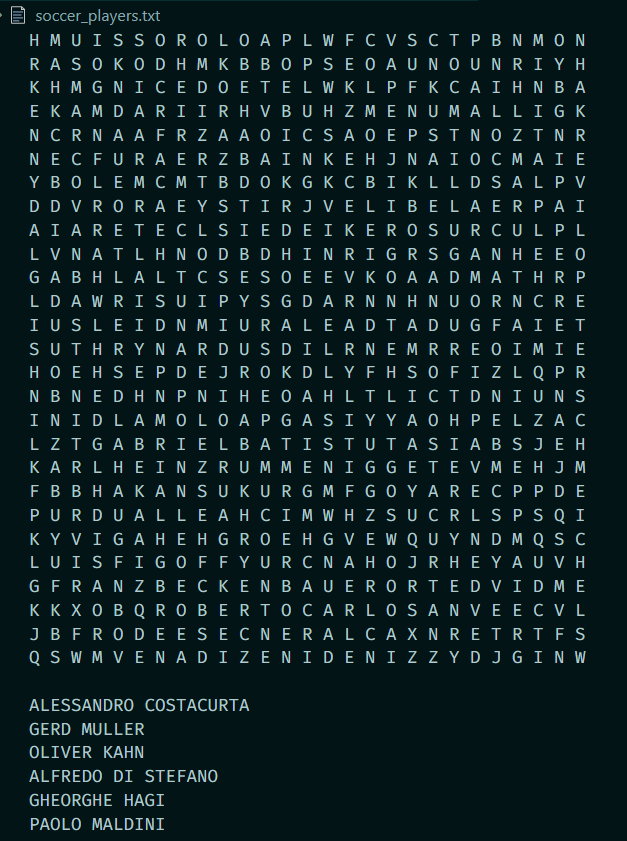
.

.

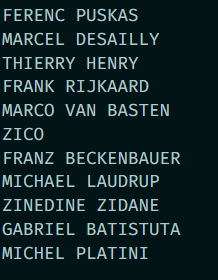
dst



1. Uji puzzle ukuran 27 x 27

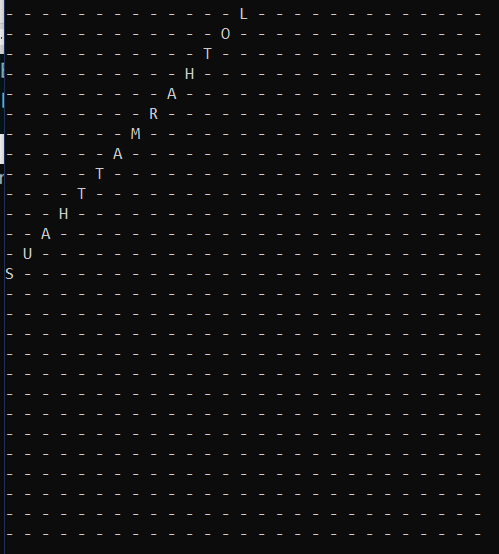






output:



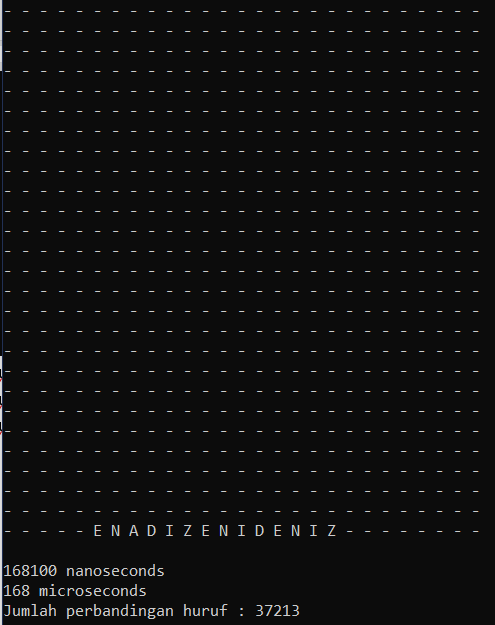


.

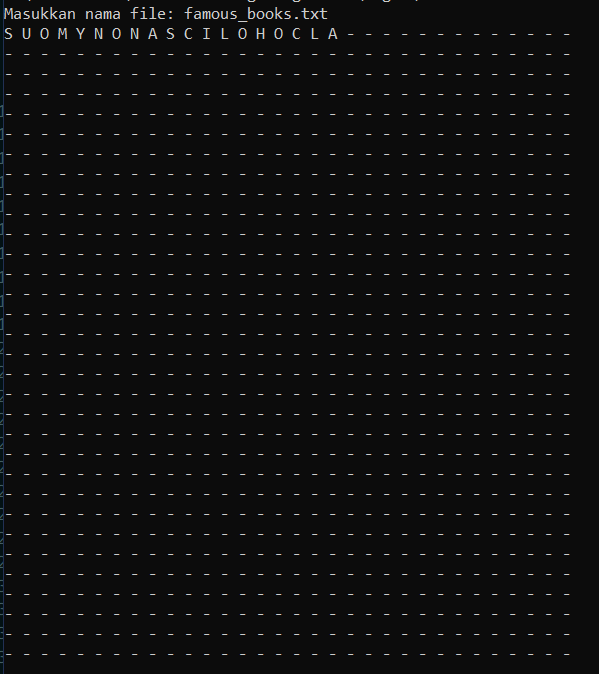
.

.

dst



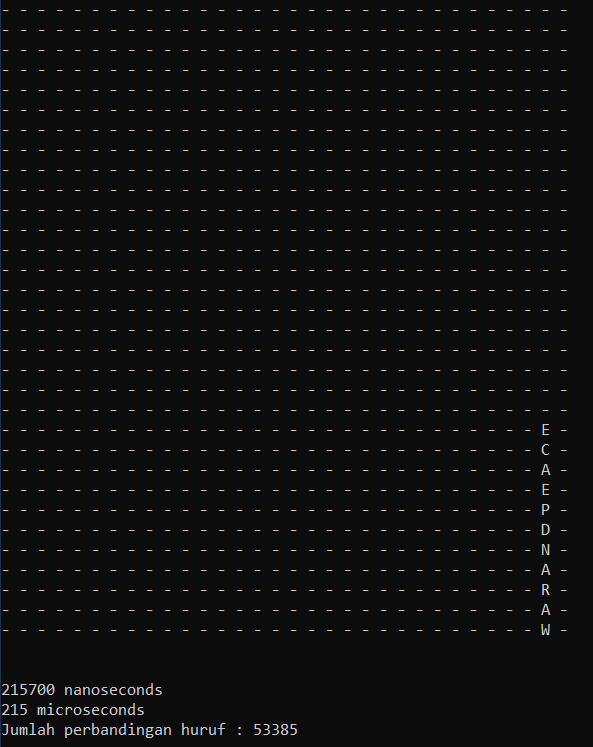
1. Uji puzzle ukuran 32 x 32



.

.

.dst



1. Alamat Drive
2. Tabel

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