**Tugas Kecil IF2211 Strategi Algoritma**

*Penyelesaian Word Search Puzzle dengan Algoritma Brute Force*

*Logo

Description automatically generated*

**Ilham Pratama  
13520041  
Kelas K-02**

**PROGRAM STUDI SARJANA TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG**

**DAFTAR ISI**

[**Algoritma Brute Force** 3](#_Toc94043002)

[**Source Code Program** 3](#_Toc94043003)

[**Screen Shoot Input dan Output** 8](#_Toc94043004)

[**Input dan Output 1** 8](#_Toc94043005)

[**Input dan Output 2** 8](#_Toc94043006)

[**Input dan Output 3** 9](#_Toc94043007)

[**Input dan Output 4** 9](#_Toc94043008)

[**Input dan Output 5** 10](#_Toc94043009)

[**Input dan Output 6** 11](#_Toc94043010)

[**Input dan Output 7** 11](#_Toc94043011)

[**Input dan Output 8** 12](#_Toc94043012)

[**Input dan Output 9** 14](#_Toc94043013)

[**Alamat Source Code Program** 16](#_Toc94043014)

# **Algoritma Brute Force**

Penyelesaian masalah tersebut secara garis besar mengikuti langkah-langkah sebagai berikut

1. Program akan memuat file, bagian file akan dipisah menjadi 2 bagian, bagian yang pertama merupakan puzzle, dan bagian kedua adalah kata yang akan dicari. Bagian puzzle akan disimpan program dalam bentuk array of array(matriks) dengan nama matriks *puzzle* dan bagian yang akan dicari akan disimpan juga dalam array dengan nama *words*
2. Setelah disimpan ke dalam bentuk matriks, selanjutnya program akan melakukan pencocokan, program akan mulai *looping*, jika ditemukan ada elemen *puzzle* yang bernilai sama dengan elemen yang ada di *words*, maka elemen pada *puzzle* akan di cek pada sebelah kanan, kiri, atas, bawah, diagonal kanan atas, diagonal kanan bawah, diagonal kiri atas dan diagonal kanan bawahnya.
3. Setelah dilakukan pengecekan dan ternyata elemennya sama dengan elemen yang ada di *words*, maka lokasi dari elemennya yang sama akan disimpan dalam suatu matriks yang bernama *match* dengan nilai 1. Selanjutnya pengecekan akan terus dilanjutkan sampai elemen pada *words* berada pada kolom terakhir dari baris yang dicari.
4. Jika semua nilainya sama, maka program akan pengecek apakah ada kemungkinan lain
5. Jika tidak ada, maka program akan lanjut ke elemen baris selanjutnya yang ada di *words*
6. Hal tersebut akan terus berulang hingga semua baris pada *words* habis.
7. Jika baris pada *words* sudah habis maka program akan menampilkan elemen pada *puzzle* yang mempunyai match 1, dan jika elemen mempunyai match selain 1 maka elemen akan diubah menjadi “ - ”.
8. Selanjutnya program akan menampilkan waktu dan jumlah perbandingan kata yang dilakukan

# **Source Code Program**

|  |
| --- |
| #include<stdio.h>  #include<string.h>  #include<ctype.h>  #include<unistd.h>  #include<time.h>  #define ROWS 100  #define COLS 100  #define MAXWORD 100  **void** **InitMatched**(**int** match[ROWS][COLS]){  /\*digunakan untuk inisialisasi matrik untuk matched, elemen diisi dengan 0 \*/  **int** i, j;  **for**(i = **0** ; i < ROWS ; i++){  **for**(j = **0** ; j < COLS; j++){  match[i][j] = **0**;  }  }  }  **void** **Max**(**int** current, **int** \*max){  // menentukan bilangan yang terbesar antara 2 bilangan  **if**(current > \*max){  \*max = current;  }  }  **int** **WordLen**(**char** \*word){  // mencari panjang dari Word  **int** i;  **for** (i = **0** ; word[i] != '\0'; i++){}  i--;  **return** i;  }  **void** **Strip**(**char**\* word){  // mengupas string di word  **int** i;  **for**(i = **0** ; word[i] != '\0'; i++){  **if**(word[i] == '\n'){  word[i] = '\0';  }  }  }  **int** **main**(){  **char** puzzle[ROWS][COLS];  **int** match[ROWS][COLS];  **char** words[MAXWORD][COLS];  **char** cc;  **int** row = **0**;  **int** col = **0**;  **int** maxcol = **0**;  **int** numword = **0**;  **int** i,j,k,l;  **FILE** \*puzzlefile;  **FILE** \*keysearch;  **double** time\_spend = **0.0**;  **int** jumlah\_banding\_kata = **0**;  **char** filename[**50**], keyname[**50**];    /\*inisialisasi matcher\*/  InitMatched(match);  printf("masukkan nama file puzzle (e.x. puzzle1.txt) : **\n**");  scanf("%s", &filename);  printf("masukkan nama file kata yang dicari (e.x. key1.txt) : **\n**");  scanf("%s", &keyname);  puzzlefile = fopen(filename, "r");  **if** (puzzlefile == NULL){  printf("gagal membuka file, periksa dulu apakah file yang anda masukkan benar **\n**");  **return** **0**;  }  **while**(!feof(puzzlefile)){  cc = fgetc(puzzlefile);  **if**(cc == '\n'){  row++;  col = **0**;  }**else** **if**(isalnum(cc)){  puzzle[col][row] = toupper(cc);  col++;  Max(col,&maxcol);  }  }  keysearch = fopen(keyname, "r");  **if**(keysearch == NULL){  printf("tidak ada kata yang akan dicari**\n**");  **return** **0**;  }  **while**(!feof(keysearch)){  fgets(words[numword], **100**, keysearch);  Strip(words[numword]);  numword++;  }  numword--;  //perhitungan waktu dimulai  **clock\_t** begin = clock();  /\*algoritma matching \*/  **for**(i = **0** ; i < numword; i++){  **for**(k = **0** ; k < row; k++){  **for**(j = **0**; j < maxcol; j++){  **if**(puzzle[j][k] == words[i][**0**]){  **if**(maxcol > (WordLen(words[i])+ j)){  **for**(l = **0**; words[i][l] == puzzle[j+l][k]; l++){  **if**(words[i][l+**1**] == '\0'){  **for**(l = **0**; words[i][l] == puzzle[j+l][k]; l++){  match[j+l][k] = **1**;  jumlah\_banding\_kata++;  }  **break**;  }**else**{  jumlah\_banding\_kata++;  }  }  **if** (row > (WordLen(words[i]) + k)) {  **for**(l=**0**; words[i][l] == puzzle[j+l][k+l]; l++) {  **if**(words[i][l+**1**]=='\0') {  **for**(l=**0**; words[i][l] == puzzle[j+l][k+l]; l++)  match[j+l][k+l] = **1**;  jumlah\_banding\_kata++;  **break**;  }**else**{  jumlah\_banding\_kata++;  }  }  }  **if** (**0** <= (k - WordLen(words[i]))) {  **for**(l=**0**; words[i][l] == puzzle[j+l][k-l]; l++){  **if**(words[i][l+**1**]=='\0') {  **for**(l=**0**; words[i][l] == puzzle[j+l][k-l]; l++)  match[j+l][k-l] = **1**;  jumlah\_banding\_kata++;  **break**;  }**else**{  jumlah\_banding\_kata++;  }  }  }  }  **if** (**0** <= (j - WordLen(words[i]))){  **for**(l=**0**; words[i][l] == puzzle[j-l][k]; l++) {  **if**(words[i][l+**1**]=='\0') {  **for**(l=**0**; words[i][l] == puzzle[j-l][k]; l++)  match[j-l][k] = **1**;  jumlah\_banding\_kata++;  **break**;  }**else**{  jumlah\_banding\_kata++;  }  }  **if** (row > (WordLen(words[i]) + k)) {  **for**(l=**0**; words[i][l] == puzzle[j-l][k+l]; l++) {  **if**(words[i][l+**1**]=='\0') {  **for**(l=**0**; words[i][l] == puzzle[j-l][k+l]; l++)  match[j-l][k+l] = **1**;  jumlah\_banding\_kata++;  **break**;  }**else**{  jumlah\_banding\_kata++;  }  }  }  **if** (**0** <= (k - WordLen(words[i]))) {  **for**(l=**0**; words[i][l] == puzzle[j-l][k-l]; l++) {  **if**(words[i][l+**1**]=='\0') {  **for**(l=**0**; words[i][l] == puzzle[j-l][k-l]; l++)  match[j-l][k-l] = **1**;  jumlah\_banding\_kata++;  **break**;  }**else**{  jumlah\_banding\_kata++;  }  }  }  }  **if** (row > (WordLen(words[i]) + k)) {  **for**(l=**0**; words[i][l] == puzzle[j][k+l]; l++) {  **if**(words[i][l+**1**]=='\0') {  **for**(l=**0**; words[i][l] == puzzle[j][k+l]; l++)  match[j][k+l] = **1**;  jumlah\_banding\_kata++;  **break**;  }**else**{  jumlah\_banding\_kata++;  }  }  }  **if** (**0** <= (k - WordLen(words[i]))){  **for**(l=**0**; words[i][l] == puzzle[j][k-l]; l++) {  **if**(words[i][l+**1**]=='\0') {  **for**(l=**0**; words[i][l] == puzzle[j][k-l]; l++)  match[j][k-l] = **1**;  jumlah\_banding\_kata++;  **break**;  }**else**{  jumlah\_banding\_kata++;  }  }  }  }  }  }  }  **clock\_t** end = clock();  **for** (k = **0**; k < row; k++) {  **for** (j = **0**; j < maxcol; j++){  **if** (match[j][k] == **1**){  printf("%c ", puzzle[j][k]);  }**else** {  printf("- ");  }  }  printf("**\n**");  }  time\_spend = (**double**)(end-begin)/CLOCKS\_PER\_SEC;  printf("waktu yang dibutuhkan untuk pencarian kata adalah:**\n**%f detik**\n**", time\_spend );  printf("jumlah kata yang dibandingkan adalah : %d", jumlah\_banding\_kata);  **return** **0**;  } |

# **Screen Shoot Input dan Output**

## **Input dan Output 1**

|  |  |
| --- | --- |
| Input | output |
| Kata yang dicari : |  |

## **Input dan Output 2**

|  |  |
| --- | --- |
| Input | output |
| Kata yang dicari |  |

## **Input dan Output 3**

|  |  |
| --- | --- |
| input | output |
| Kata yang dicari: |  |

## **Input dan Output 4(medium)**

|  |  |
| --- | --- |
| Input | Output |
| Kata yang dicari  Text  Description automatically generated |  |

## **Input dan Output 5**

|  |  |
| --- | --- |
| input | output |
| Yang dicari : |  |

## **Input dan Output 6**

|  |  |
| --- | --- |
| input | output |
| Yang dicari |  |

## **Input dan Output 7**

|  |
| --- |
| input |
| Yang dicari : |
| output |
|  |

## **Input dan Output 8**

|  |
| --- |
| input |
| Yang dicari |
| output |
|  |

## **Input dan Output 9**

|  |
| --- |
| input |
| Yang dicari: |
| output |
|  |

# **Alamat Source Code Program**

Program dapat ditemukan pada link berikut.

<https://github.com/ilhampratama2109/word-seach-puzzle-solver>

|  |  |  |
| --- | --- | --- |
| Point | Ya | Tidak |
| Program berhasil dikompilasi tanpa kesalahan (no syntax error) | V |  |
| Program berhasil *running* | V |  |
| Program dapat membaca file masukan dan menuliskan luaran. | V, agak berbeda dari spesifikasi, di program saya file input ada 2, yaitu file puzzle dan file kata yang akan dicari, saya belum bisa memadukannya kedalam satu file karena keterbasan saya |  |
| Program berhasil menemukan semua kata di dalam puzzle. | V |  |