# LAPORAN TUGAS KECIL 1 IF2211 STRATEGI ALGORITMA

# PENYELESAIAN *WORD SEARCH PUZZLE* DENGAN ALGORITMA *BRUTE FORCE*

# Logo Description automatically generated

Oleh

Firizky Ardiansyah

13520095

K02

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2022**

# BAB 1

# ALGORITMA *BRUTE FORCE*

Algoritma *brute force* adalah suatu strategi penyelesaian permasalahan komputasi dengan meninjau semua kasus yang mungkin dari permasalahan tersebut. Pencarian solusi permasalahan menggunakan algoritma *brute force* dengan tepat menjamin solusi permasalahan tersebut ditemukan untuk setiap kemungkinan. Namun, penggunaan algoritma ini cukup mahal. Permasalahan dengan kasus yang perlu ditinjau cukup banyak akan memerlukan memori yang besar dan waktu yang lama. Oleh sebab itu, algoritma ini cocok dipakai untuk batasan banyaknya kasus yang perlu ditinjau tidak terlalu banyak.

Ada banyak sekali cara untuk mengoptimalisasi algoritma *brute force*. Sudut pandang pengambilan kasus yang akan ditinjau saja tidak jarang mengoptimalisasi program secara signifikan. Umumnya, optimalisasi algoritma *brute force* dilakukan dengan mengurangi kemungkinan kasus yang sudah dipastikan tidak akan menjadi solusi persoalan, sehingga kasus yang perlu ditinjau menjadi lebih sedikit.

Pada penyelesaian *word search puzzle* ini, penulis mengimplementasikan algoritma *brute force* dengan beberapa optimalisasi. Secara garis besar optimalisasi yang dilakukan adalah pengecekan kesamaan huruf pertama dengan petak pada *puzzle,* pengecekan panjang maksimal yang bisa dimuat oleh petak pada arah tertentu, serta men-*terminate loop* saat suatu kata sudah ditemukan di petak-petak tertentu.

Langkah pertama yang dilakukan adalah menerima masukan dari pengguna fail teka-teki yang akan diselesaikan. Jika fail yang pengguna masukkan tidak valid, program akan melakukan validasi berulang hingga fail teka-teki yang dimasukkan pengguna valid. Alamat fail yang valid akan disimpan pada sebuah string *addrIn*, selanjutnya alamat direktori untuk menyimpan fail keluaran juga akan disimpan pada direktori fail masukan, tetapi dengan nama yang diberi awalan “out\_” tanpa tanda petik. Alamat keluaran akan disimpan pada string *addrOut*.

Selanjutnya, yang dilakukan adalah membaca fail masukan setiap baris dan memasukkannya dalam senarai dinamis. Bagian teka-teki dan kata yang akan dicari pada awalnya masih tergabung dalam array dinamis, sehingga perlu dipisahkan menjadi sebuah matriks berisi karakter-karakter pada teka-teki dan sebuah senarai dari kata-kata yang perlu dicari. Masing-masing hasil pemisahan disimpan dalam matriks *input* dan senarai *words*. Matriks *output* pada prosedur ini nantinya akan digunakan untuk menampilkan solusi pada terminal, sehingga perlu diinisialisasi persis seperti matriks *input*, tetapi tipenya string (tipe data string diperlukan agar nantinya elemen dalam matriks dapat diwarnai).

Pemrosesan pencarian solusi dilakukan dengan satu persatu kata dicari pada petak-petak teka-teki dari mulai kiri-atas hingga kanan-bawah. Setiap petak yang ditinjau, akan dilakukan pengecekan kesamaan huruf pada petak dengan huruf pertama kata yang dicari. Jika petak tersebut menyimpan huruf yang sama dengan huruf pertama kata yang dicari, dilakukan pengecekan jumlah huruf pada kata yang dicari. Jika jumlah hurufnya hanya satu, solusi ditemukan (sebab petak tersebut sudah sama dengan kata yang dicari), sedangkan jika jumlah hurufnya lebih dari satu, dilakukan iterasi ke setiap arah mata angin relatif terhadap petak yang hurufnya sesuai dengan huruf pertama kata yang dicari.

Pada setiap arah, dilakukan pengecekan batas maksimal kata yang dapat dimuat dari petak pertama hingga ujung-ujung matriks. Jika panjang kata yang dapat dimuat dalam arah tersebut tidak lebih besar atau tidak sama dengan panjang katanya, arah tersebut tidak akan dicek lebih lanjut. Pada arah dengan panjang yang memenuhi, pencocokan string kata yang dicari dengan petak pada teka-teki dilakukan dengan mengiterasi huruf-perhuruf pada arah tersebut dan berhenti jika ditemukan huruf yang tidak sesuai.

Kasus ketika string sudah ditemukan, koordinat petak-petak solusi disimpan pada senarai *coor*. Keluaran akan disajikan dalam dua buah format, fail keluaran dan antarmuka terminal. Pada fail keluaran, akan tersedia detail letak kata-kata yang perlu dicari dengan banyaknya komparasi dan waktu yang dibutuhkan. Pada keluaran yang disajikan dalam terminal, akan ditampilkan matriks teka-teki dengan solusi semua kata yang telah diwarnai.

Proses pembuatan fail keluaran adalah dengan memberikan keterangan kata yang dicari kemudian menulis matriks yang hanya menampilkan petak-petak solusi dan sisanya diberi simbol ‘-‘. Banyaknya komparasi dihitung berdasarkan banyaknya huruf yang dibandingkan pada iterasi sebuah kata. Waktu yang diperlukan adalah lamanya waktu eksekusi untuk menemukan solusi dari kata tersebut. Di akhir, fail ini juga menampilkan banyaknya komparasi keseluruhan dan waktu yang dibutuhkan untuk menemukan sebuah kata. Adapun keluaran yang ditampilkan dalam terminal pada dasarnya hanyalah mewarnai petak-petak yang merupakan solusi.

# BAB 2

# *SOURCE PROGRAM*

|  |
| --- |
| /\*  Nama : Firizky Ardiansyah/13520095  Tanggal : 24/01/2022  Kelas : K02  Tugas Kecil 1 Strategi Algoritma  Penyelesaian Word Search Problem Menggunakan Algoritma  \*/  #include <bits/stdc++.h>  #include <fstream>  using namespace std;  // Color Dictionary  const string EOS = "\u001b[0m"; // to reset color  string dict[] = {"\u001b[31m", "\u001b[32m", "\u001b[33m",  "\u001b[34m", "\u001b[35m", "\u001b[36m", "\u001b[36;1m",  "\u001b[31;1m", "\u001b[32;1m", "\u001b[33;1m", "\u001b[34;1m", "\u001b[35;1m"};  // Global Directiory  int dx[] = {1,1,1,0,0,-1,-1,-1};  int dy[] = {1,0,-1,1,-1,1,0,-1};  int n, m, l, cntWord;  long long allcomp = 0;  string line;  vector<string> lines, words;  vector<vector<char>> input;  vector<vector<string>> output;  clock\_t S, E;  // Interface  void startProgram(){  cout << dict[2]+"WELCOME!\n"+EOS;  cout << "============================================================\n";  cout << "This is a program to solve the well-known puzzle called word-search puzzle\n";  cout << "Let's Solve Some Problems!\n\n";  }  void inputFile(string& addrIn, string &addrOut){  cout << "Type Puzzle File You Want to Solve (without .txt): ";  cin >> addrIn;  addrOut = "./test/out\_"+addrIn+".txt";  addrIn = "./test/"+addrIn+".txt";  cout << "Opening " << addrIn << "...\n\n";  ifstream curIn(addrIn);  if(!curIn.good()){  cout << dict[0]+"The puzzle you requested was not found, please try again\n"+EOS;  // recursively open the address entered, to avoid infinite loop  inputFile(addrIn, addrOut);  }  }  // Inisialisasi  void init(string addrIn, string addrOut){  ifstream in(addrIn);  // read line by line  while(getline(in, line)){  lines.push\_back(line);  }  // parse it to puzzle and words list  for(auto s: lines){  if(s.size()!=lines[0].size()){  // if the corresponding string has size less than the  // one on the first line,  // it must be words from words-list  if(s[1]!=' '&&s[1]!='\t'){  // we also need to check whether the word  // contain blank character or not  // if it doesn't, it definitely a word from words-list  while(s.size()>0&&(s.back()==' '||s.back()=='\t'||s.back()=='\n')){  s.pop\_back();  }  if(s.size()!=0){  words.push\_back(s);  }  continue;  }  }  vector<char> cur;  vector<string> curS;  // if it's part of puzzle matrix, put it on input matrix.  for(auto c: s){  if(c!=' '&&c!='\n'&&c!='\t'){  cur.push\_back(c);  // output must have string type  string sc = "";  sc += c;  curS.push\_back(sc);  }  }  input.push\_back(cur);  output.push\_back(curS);  }  n = input.size();  m = input[0].size();  l = words.size();  in.close();  }  // Create Analysis For One Word  void createSummary(vector<pair<int, int>> coor, ofstream& to, long comp, string word, double duration){  to << "Searched Word: " << word << "\n";  // initialize vector as matrix of '-'  vector<vector<char>> res(n, vector<char> (m, '-'));  // for every solution cells, change it to character written in the puzzle  for(auto p: coor){  int x = p.first;  int y = p.second;  res[x][y] = input[x][y];  }  for(int x=0; x<n; x++){  string s = "";  for(int y=0; y<m; y++){  s += res[x][y];  if(y==m-1){  s += '\n';  }  else{  s += ' ';  }  }  to << s;  }  to << "Comparisons Count: " << comp << "\n";  to << fixed;  to << "Duration: " << setprecision(10) << duration << " seconds\n\n";  }  // Colorize Summarized Solution  void colorize(vector<pair<int, int>> coor, int id){  // id is color id  for(auto p: coor){  int x = p.first;  int y = p.second;  output[x][y] = dict[id]+output[x][y]+EOS;  }  }  // Main Program  int main() {  string addrIn, addrOut;  startProgram();  inputFile(addrIn, addrOut);  init(addrIn, addrOut);  ofstream out(addrOut);  S = clock();  for(int i=0; i<l; i++){  bool found = false;  long comp = 0;  clock\_t curS, curE;  curS = clock();  for(int j=0; j<n&&!found; j++){  for(int k=0; k<m&&!found; k++){  // this comparison count for one if below  allcomp++; comp++;  if(input[j][k] == toupper(words[i][0])){  if(words[i].size()==1){  vector<pair<int, int>> coor(1, {j, k});  curE = clock();  double dur = (double) (curE-curS)/(CLOCKS\_PER\_SEC);  createSummary(coor, out, comp, words[i], dur);  colorize(coor, i%12);  cntWord++;  // if it just one word, it's done  found = true;  }  for(int d=0; d<8&&!found; d++){  int sz = words[i].size();  int boundj = j+(sz-1)\*dx[d];  int boundk = k+(sz-1)\*dy[d];  vector<pair<int, int>> coor (1, {j, k});  int nj = j+dx[d];  int nk = k+dy[d];  bool match = boundj<n&&boundj>=0&&boundk<m&&boundk>=0;  int cnt = 1;  while(match&&cnt<sz){  if(input[nj][nk]!=toupper(words[i][cnt])){  match = false;  }  else{  coor.push\_back(make\_pair(nj, nk));  cnt++;  nj += dx[d];  nk += dy[d];  }  // every match or not match comparison in  // this while loop count as one  allcomp++;  comp++;  }  if(match&&cnt==sz){  curE = clock();  double dur = (double) (curE-curS)/(CLOCKS\_PER\_SEC);  createSummary(coor, out, comp, words[i], dur);  colorize(coor, i%12);  cntWord++;  found = true;  }  }  }  }  }  }  E = clock();  // Final line on detailed solution  out << "Overall Comparisons: " << allcomp << "\n";  out << "Overall Excecution Time : ";  out << setprecision(10) << (double) (E-S)/CLOCKS\_PER\_SEC << " seconds \n";  out.close();  // printing summarized solution  cout << dict[1] << "This is the solution of " << addrIn << "\n" << EOS;  cout << "Puzzle size: " << n << " x " << m << "\n\n";  for(int i=0; i<n; i++){  for(int j=0; j<m; j++){  cout << output[i][j] << " \n"[j==m-1];  }  }  cout << "\nTotal Word(s) Found: " << cntWord << "\n";  cout << "Overall Comparisons: " << allcomp << "\n";  cout << "Overall Excecution Time : ";  cout << fixed;  cout << setprecision(10) << (double) (E-S)/CLOCKS\_PER\_SEC << " seconds \n";  cout << dict[2] << "\nYou can also see more detailed solution in " << addrOut << "\n" << EOS;  return 0;  } |

# BAB 3

# TANGKAPAN LAYAR PENGETESAN

Tangkapan layar masukan-keluaran yang akan ditampilkan disini hanyalah keluaran yang ditampilkan pada terminal, keluaran pada fail bisa mencapai ribuan baris, sehingga tidak ditampilkan. Berikut adalah interface awal program.

Text

Description automatically generated

## Ukuran kecil (14x12)

|  |  |
| --- | --- |
|  |  |

Solution:

Text

Description automatically generated

## Ukuran kecil (16x14)

|  |  |
| --- | --- |
|  |  |

Solution:

Text

Description automatically generated with medium confidence

## Ukuran kecil (18x16)

|  |  |
| --- | --- |
|  |  |

Solution:

A picture containing text

Description automatically generated

## Ukuran sedang (20x18)

|  |  |
| --- | --- |
|  |  |

Solution:

A picture containing text

Description automatically generated

## Ukuran sedang (22x20)

|  |  |
| --- | --- |
|  |  |

Solution:

A picture containing arrow

Description automatically generated

## Ukuran sedang (24x22)

|  |  |
| --- | --- |
|  |  |

Solution:

A picture containing arrow

Description automatically generated

## Ukuran besar (32x30)

|  |  |
| --- | --- |
|  |  |

Solution:

A picture containing shape

Description automatically generated

## Ukuran besar (34x32)

|  |  |
| --- | --- |
|  |  |

Solution:

Background pattern

Description automatically generated

## Ukuran besar (36x34)

|  |  |
| --- | --- |
|  |  |

Solution:

Background pattern

Description automatically generated

# BAB 4

# DOKUMENTASI

Program penyelesaian *word search puzzle* menggunakan algoritma *brute force* lebih lengkap dapat diakses pada <https://github.com/firizky29/word-search-problem-solver>.

# BAB 5

# TABEL *CHECK 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* | √ |  |