LAPORAN TUGAS 02

DESAIN DAN ANALISIS ALGORITMA

DISUSUN OLEH

ILHAM NUR ROMDONI M0520038

PROGRAM INFORMATIKA

FAKULTAS MIPA

UNIVERSITAS SEBELAS MARET

2021

METODE *DIVIDE AND CONQUER*

1. **Deskripsi Masalah**

   Sebagai mahasiswa, *text editor* adalah salah satu *software* yang paling sering digunakan. *Text editor* digunakan dalam pengerjaan tugas, penyusunan laporan, pembuatan *source code* dan lain sebagainya. Untuk memudahkan pengerjaan, *text editor* menyediakan banyak *tools* pembantu salah satunya adalah *Replace*. *Tool* ini juga bisa digunakan untuk mengganti suatu nama variabel menjadi *string* lain yang dikehendaki pada suatu penyusunan *source code*.

   Pada mata Kuliah yang berhubungan dengan pemrograman, penyusunan *source code* adalah hal yang biasa. Terkadang karena alasan tertentu, sebagian mahasiswa menyalin *source code* milik teman mereka. *Source code* dapat diganti nama variabel sehingga program mereka tidak terlihat sama dengan sumber yang disalin.

2. **Ide Penyelesaian Masalah**

   Masalah dapat diselesaikan dengan membandingkan satu per satu baris dari *file source code* yang terlihat sama. Pengecekan satu per satu akan memakan banyak waktu. Untuk memudahkan hal tersebut, dibuatlah sebuah *source code* yang dapat membandingkan dua isi *source code* sehingga bisa diketahui apakah keduanya memiliki alur proses yang mirip atau tidak. Metode perbandingan akan dilakukan berdasarkan algoritma *divide and conquer*.

   Untuk membandingkan dua *file source code*, digunakan algoritma sebagai berikut :

   a. Membuat sebuah variabel penerjemah. Variabel ini adalah *Binary Search Tree* yang *node*-nya berisi pasangan nilai string1 dan string2. Penerjemah akan menerjemahkan setiap string1 menjadi string2.
   b. Buka *file* yang mana baris per barisnya akan dibaca.
   c. Setiap *string* yang dibaca per baris, lakukan iterasi dengan variabel j.
   d. Buat sebuah variabel, misalnya i yang diinisialisasi dengan 0.
   e. Jika posisi saat ini adalah tanda baca atau spasi, maka ambil *substring* posisi i ke j-1. Jika *substring* tersebut tidak ada di penerjemah, tambahkan

substring. Tuliskan terjemahan dari *substring* ke *file temporary*, lalu nilai i diubah menjadi j+1.

f. Jika posisi saat ini adalah angka dan i = j, lakukan penambahan nilai j hingga j mencapai akhir *string* atau karakter ke-j adalah tanda baca atau spasi. Ambil *substring* posisi i ke j-1 dan tuliskan ke *file temporary*.

g. Jika di akhir baris, nilai j > i, ambil *substring* posisi i .. j-1. Lakukan pengecekan seperti langkah e.

h. *File temporary* pertama dan kedua akan dibandingkan.

3. *Source Code*

```
#include <iostream>
#include <fstream>
#include <map>
#include <vector>
#include <cstring>

using namespace std;

#define MAX 1000

char name1[MAX], name2[MAX];
FILE *f1, *f2;

bool match (char c) {
        return ((c == ' ') || (c == '\t') || (c == '+') || (c == '-') || (c == '*') ||
                (c == '/') || (c == '\\') || (c == ':') || (c == ',') || (c == ';') ||
                (c == '"') || (c == '\'') || (c == '>') || (c == '<') || (c == '=') ||
                (c == '.') || (c == '&') || (c == '!') || (c == '(') || (c == ')') ||
                (c == '[') || (c == ']') || (c == '{') || (c == '}') || (c== '#') ||
                (c == '?') || (c == '%') || (c == '$') || (c == '^'));
}

void matching (char const* name1, char const* name2) {
        char temp[MAX], temp2[MAX];
        bool match = true;
        f1 = fopen (name1, "r");
        f2 = fopen (name2, "r");

        while ((fscanf (f1, "%s", temp) != EOF) && match)
```

```
        {
                if (fscanf (f2,"%s",temp2) == EOF)
                        match = false;
                else
                        match = strcmp(temp, temp2) == 0;
        }
        if (match)
        {
                if (fscanf (f2,"%s",temp2) == EOF)
                        cout << "Kedua file memiliki kemiripan\n";
                else
                        cout << "Kedua file berbeda\n";
        }
        else cout<< "Kedua file berbeda\n";
        fclose(f2);
        fclose(f1);
}

void convert (char const * name, char const * name2) {
        string temp, temp2;
        map <string, int> a;
        vector <string> s;
        char tempc[100];
        f1 = fopen (name2, "w");
        ifstream input (name , ifstream::in);
        int i, j, n;
        char c;

        while (getline (input, temp)) {
                n = temp.length();
                j = 0;
                for (i=0; i < n; ++i) {
                        c = temp[i];
                        if (match(c)) {
                                if (j < i) {
                                        temp2 = temp.substr(j, i-j);
                                        j = a[temp2];
                                        if (!j) {
                                                j = s.size();
                                                a[temp2] = j;
```

```
                                        if (j > 0) {
                                                itoa(j, tempc, 10);
                                                s.push_back ("a" +
(string)tempc);
                                        }
                                        else s.push_back ("a0");
                                }
                                fprintf(f1, "%s ", s[j].c_str());
                        }
                        if ((c != ' ') && (c != '\t'))
                                fprintf (f1, "%c", c);
                        j = i+1;
                }
                else if ((c <= '9') && (c>='0') && (i==j)) {
                        do
                                ++i;
                        while ((i < n) && (!match(temp[i])));
                        temp2 = temp.substr(j,i-j).c_str();
                        fprintf(f1, "%s ", temp2.c_str());
                        j = i+1;
                }
        }
        if (j < i) {
                fprintf(f1, "%s ", temp.substr(j, i-j).c_str());
                cout << temp.substr(j, i-j) << "...\n";
        }
    }
    input.close();
    fclose(f1);
}

int main() {
    cout << "Masukkan nama file pertama : ";
    cin >> name1;

    cout << "Masukkan nama file kedua : ";
    cin >> name2;

    convert (name1, "temp1.txt");
    convert (name2, "temp2.txt");
```

```
    matching ("temp1.txt", "temp2.txt");
    return 0;
    }
```

4. Hasil *Running* program

Program akan menghasilkan *output* di bawah ini jika memiliki alur yang mirip.

```
Masukkan nama file pertama : M0520038_Ilham.cpp
Masukkan nama file kedua : Ilham.cpp
Kedua file memiliki kemiripan
```

Program akan menghasilkan *output* di bawah ini jika memiliki alur yang berbeda.

```
Masukkan nama file pertama : M0520038_Ilham.cpp
Masukkan nama file kedua : Mahasiswa #1.cpp
Kedua file berbeda
```

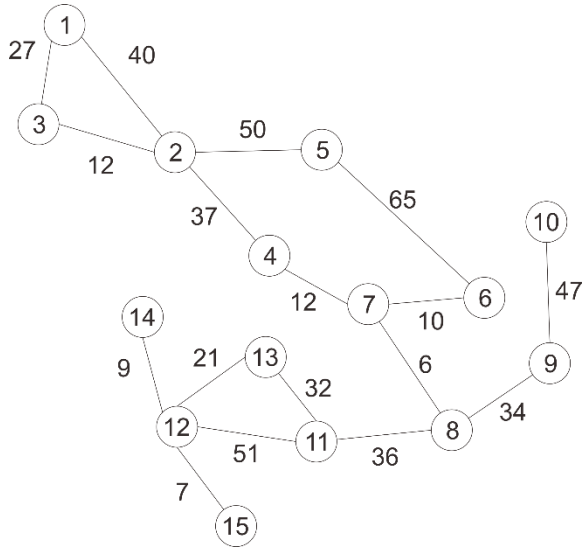
5. Kesimpulan algoritma yang dipilih

Program yang dibuat sederhananya hanya mengganti suatu kata (selain operator, tanda baca, dan angka) dengan sebuah *string*. Setiap kata secara berurutan akan berubah menjadi *string* dimulai dari a1 dan seterusnya. Jika kedua *source code* memang sama, walaupun memiliki nama variabel yang berbeda, maka kedua *file temporary* berisi sama. Kelemahan *source code* ini adalah akan tetap membaca *comment* sehingga dengan menambahkan isi *comment* yang berbeda saja sudah mengganggu program dalam membandingkan dua *file*. Sehingga harus diasumsikan bahwa tidak ada *comment* pada *source code*. Kompleksitas algoritma ini adalah O(n log k + s). Dengan kompleksitas tersebut, algoritma ini tergolong algoritma *Polynomial*. Diperlukan algoritma yang lebih umum lagi sehingga bisa mengurangi asumsi yang dibutuhkan.

ROUTE PLANNING PROBLEM

1. Deskripsi Masalah

Disajikan sebuah *graph* perjalanan antar kota dari provinsi X. Setiap kota dilambangkan dengan angka. Jarak antar kota bernilai km.



Mencari jalur terdekat perjalanan dari kota 1 ke kota 15.

2. Ide Penyelesaian Masalah

Untuk mengetahui jalur terdekat dari kota 1 ke kota 15 akan digunakan algoritma Dijstra. Algoritma Dijstra adalah sebuah algoritma rakus (*greedy algorithm*) yang dipakai dalam memecahkan permasalahan jarak terpendek (*shortest path problem*) untuk sebuah *graph* berarah (*directed graph*) dengan bobot-bobot garis (*edge weights*) yang bernilai non negatif, [0, ∞].

Cara kerja algoritma Dijktra memakai strategi *greedy*, di mana pada setiap langkah dipilih sisi dengan bobot terkecil yang menghubungkan sebuah *vertex* yang sudah terpilih dengan *vertex* lain yang belum terpilih. Algoritma Dijktra membutuhkan parameter berupa tempat asal dan tempat tujuan. Pembuatan *source code* dilakukan dengan langkah-langkah sebagai berikut.

a. Mendefinisikan variabel global yaitu *infinity*.
b. Menginisialisasi variabel yang dibutuhkan.
c. Memasukkan jumlah *vertex* yang digunakan.
d. Menginisialisasi matriks dari bobot, *buffer*, *path* dll.
e. Menginisialisasi nilai awal bobot, *buferr*, dll.
f. Memasukkan nilai bobot.

g. Memasukkan *vertex* awal dan tujuan.

h. Proses pencarian bobot terkecil dengan tetap mengecek *vertex* yang sudah dikunjungi.

i. Pencarian jalur yang dilalui dengan *backtracking*.

j. Menampilkan bobot terkecil yang dilalui.

k. Menampilkan *vertex* yang dikunjungi.

l. *Vertex* / *node* yang tidak terhubung secara langsung dianggap berjarak *infinity*.

3. *Source Code*

```
#include <iostream>

using namespace std;

#define INF 999

int main() {
    int i, j, source, target, start, minimum, m, update, ver, min_weight;
    cout << "Masukkan Jumlah Vertex / Node yang diinginkan : ";
    cin >> ver;
    ver++;
    int weight[ver][ver], buff[ver], path[ver], prev[ver], visited[ver]={0};

    for(i = 1; i < ver; i++) {
        buff[i] = INF;
        prev[i] = -1;
        path[i] = 0;
        for(int j = 1;j < ver; j++) {
            weight[i][j] = INF;
        }
    }

    for(i = 1;i < ver; i++) {
        for(j = i+1; j < ver; j++)
        {
            cout << "Masukkan bobot " << i << " ke " << j << " (masukkan 999 untuk
representasi Infinity) : ";
            cin >> weight[i][j];
            weight [j][i] = weight[i][j];
```

```
    }
    cout << "\n";
}

cout << "Masukkan Vertex / Node Awal : ";
cin >> source;
cout << "Masukkan Vertex / Node Tujuan : ";
cin >> target;

start = source;
visited[start]=1;
buff[start] = 0;

while(visited[target] == 0) {
    minimum = INF;
    m = 0;
    for(i=1;i< ver;i++) {
        update = buff[start] + weight[start][i];
        if(update < buff[i] && visited[i]==0) {
            buff[i] = update;
            prev[i] = start;
        }
        if(minimum > buff[i] && visited[i]==0) {
            minimum = buff[i];
            m = i;
        }
    }
    start = m;
    visited[start] = 1;
}

min_weight = buff[target];

start = target;
j = 0;

while(start != -1) {
    path[j] = start;
    start = prev[start];
    j++;
```

```
    }

    cout << "\nBobot terkecil yang dilalui adalah " << min_weight << "\n";
    cout << "Jalur yang ditempuh adalah : ";

    for (int i = ver-1; i >= 0; i--) {
        if (path[i] != 0) {
            cout << path[i] << " ";
        }
    }
    cout << "\n";
}
```

4. Hasil *Running* program

```
Masukkan Jumlah Vertex / Node yang diinginkan : 15
Masukkan bobot 1 ke 2 (masukkan 999 untuk representasi Infinity) : 40
Masukkan bobot 1 ke 3 (masukkan 999 untuk representasi Infinity) : 27
Masukkan bobot 1 ke 4 (masukkan 999 untuk representasi Infinity) : 999
Masukkan bobot 1 ke 5 (masukkan 999 untuk representasi Infinity) : 999
Masukkan bobot 1 ke 6 (masukkan 999 untuk representasi Infinity) : 999
Masukkan bobot 1 ke 7 (masukkan 999 untuk representasi Infinity) : 999
Masukkan bobot 1 ke 8 (masukkan 999 untuk representasi Infinity) : 999
Masukkan bobot 1 ke 9 (masukkan 999 untuk representasi Infinity) : 999
Masukkan bobot 1 ke 10 (masukkan 999 untuk representasi Infinity) : 999
Masukkan bobot 1 ke 11 (masukkan 999 untuk representasi Infinity) : 999
Masukkan bobot 1 ke 12 (masukkan 999 untuk representasi Infinity) : 999
Masukkan bobot 1 ke 13 (masukkan 999 untuk representasi Infinity) : 999
Masukkan bobot 1 ke 14 (masukkan 999 untuk representasi Infinity) : 999
Masukkan bobot 1 ke 15 (masukkan 999 untuk representasi Infinity) : 999
```

Input dilakukan hingga bobot 14 ke 15.

```
Masukkan bobot 13 ke 14 (masukkan 999 untuk representasi Infinity) : 999
Masukkan bobot 13 ke 15 (masukkan 999 untuk representasi Infinity) : 999

Masukkan bobot 14 ke 15 (masukkan 999 untuk representasi Infinity) : 999


Masukkan Vertex / Node Awal : 1
Masukkan Vertex / Node Tujuan : 15

Bobot terkecil yang dilalui adalah 188
Jalur yang ditempuh adalah : 1 3 2 4 7 8 11 12 15
```

Seperti yang ditunjukkan pada hasil di atas bahwa jalur terdekat yang harus ditempuh dari kota 1 ke kota 15 adalah 118 km. dengan harus melewati kota 1, 3, 2, 4, 7, 8, 11, 12, dan lalu kota tujuan yaitu 15.