LAPORAN PRAKTIKUM
STRUKTUR DATA DAN ALGORITME

MODUL 10

GRAPH II

Disusun Oleh :

Nama : Fatkhurrohman Purnomo

NIM : 21102125

Dosen Pengampu

Ipam Fuaddina Adam, S.T., M.Kom.

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

PURWOKERTO

2022

A. Dasar Teori

a. Algoritma Dijkstra

Algoritma Dijkstra merupakan algoritma dalam pencarian lintasan/rute terpendek. Algoritma ini ditemukan oleh Edger W. Dijkstra (1930 – 2002), merupakan tokoh yang paling berpengaruh dari generasi penemu ilmu komputer. Pada setiap langkah dalam algoritma Dijkstra, ambil sisi yang berbobot minimum yang menghubungkan sebuah simpul yang telah dikunjungi dengan sebuah simpul lain yang belum dikunjungi. Lintasan dari simpul asal ke simpul yang baru haruslah merupakan lintasan yang terpendek diantara semua lintasannya ke simpul-simpul yang belum dikunjungi.

Berikut langkah-langkah algoritma Dijkstra:

1. Berikan nilai bobot untuk setiap titik ke titik lainnya, berikan nilai 0 pada node awal dan nilai tak hingga terhadap node lain (yang belum terisi).
2. Atur semua node dengan label “Belum dikunjungi” dan atur node awal sebagai “Node keberangkatan”.
3. Dari node keberangkatan, pertimbangkan node tetangga yang belum terjamah dan hitung jaraknya dari titik keberangkatan. Sebagai contoh, jika titik keberangkatan X ke Y memiliki bobot jarak 7 dan dari Y ke node Z berjarak 13, maka jarak ke Z melewati Y menjadi 7+13=20. Jika jarak ini lebih kecil dari jarak sebelumnya (yang telah terekam sebelumnya) hapus data lama, simpan ulang data jarak dengan jarak yang baru.
4. Saat kita selesai mempertimbangkan setiap jarak terhadap node tetangga, tandai node yang telah terjamah sebagai “Node dikunjungi”. Node yang telah dikunjungi tidak akan pernah di cek kembali, jarak yang disimpan adalah jarak terakhir dan yang paling minimal bobotnya.

5. Set “node belum dikunjungi” dengan jarak terkecil (dari node asal) sebagai “node asal” selanjutnya, kemudian lanjutkan dengan mengulang langkah 3.

b. Algoritma Flyod - Warshall

Algoritma pencarian rute terpendek Flyod – Warshall melakukan perbandingan jarak/bobot semua simpul sehingga hasil dari perhitungan algoritma ini adalah lintasan terpendek untuk semua pasangan simpul yang terdapat pada graph. Kelebihan dari algoritma ini yakni dapat diterapkan meskipun nilai jarak/bobot suatu busur/edge negatif berbeda dengan algoritma Dijkstra yang hanya dapat digunakan pada graph yang bernilai positif untuk semua jarak/bobot busurnya.

Langkah awal dalam menerapkan algoritma ini untuk menemukan lintasan terpendek dari sebuah graph adalah dengan membuat representasi graph dalam bentuk matrik adjacency. Graph dalam bentuk matrik tersebut yang kemudian diproses. Algoritma Flyod – Warshall merupakan algoritma yang bersifat iteratif, jumlah proses iterasi yang mungkin terjadi sebanyak jumlah simpul pangkat 3 (n3), sehingga apabila terdapat 7 simpul maka akan terjadi perulangan sebanyak 343.

Algoritma ini juga bisa diterapkan pada sebuah aplikasi pencari rute jalan yang terdekat dari suatu daerah ke daerah lainnya. dengan metode ini hasil yang di dapat bisa lebih optimal namun memerlukan resource yang cukup besar jika dipakai untuk pencarian yang kompleks.

Ref:

Modul 10: GRAPH II

PENERAPAN ALGORITMA FLOYD WARSHALL UNTUK PENYELESAIAN KASUS MENGGUNAKAN PROGRAM C++

MAKALAH

B. Guided

1. Program Pencarian Lintasan Terpendek Algoritma Dijkstra

```
// Nama : Fatkhurrohman Purnomo
// NIM : 21102125

#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

/// PROGRAM PENCARIAN LINTASAN TERPENDEK (ALGORITMA
DIJKSTRA)
// Deklarasi Global
int jumlahSimpul;
string *dataSimpul;
int **dataBusur;
int *jarakDiketahui;
int *kunjungan;
int indeksPosisi, simpulSaatIni, simpulAsal,
simpulTujuan, jarakSaatIni, jarakLama, jarakBaru;
int dikunjungi = 1;
int belumDikunjungi = 0;
bool cekMatriks = false;

// Buat Graph
void buatMatriks()
{
    dataSimpul = new string[jumlahSimpul];
    dataBusur = new int *[jumlahSimpul];
    dataBusur[0] = new int[jumlahSimpul * jumlahSimpul];
    for (int i = 1; i < jumlahSimpul; i++)
    {
        dataBusur[i] = dataBusur[i - 1] + jumlahSimpul;
    }
    cout << " Silahkan masukkan nama simpul" << endl;
    for (int i = 0; i < jumlahSimpul; i++)
    {
        cout << " Simpul " << i + 1 << ": ";
        cin >> dataSimpul[i];
    }
    cout << endl;
    cout << " Silahkan masukkan bobot antar simpul" <<
endl;
    for (int baris = 0; baris < jumlahSimpul; baris++)
```

```
    {
        for (int kolom = 0; kolom < jumlahSimpul;
kolom++)
        {
            cout << " " << dataSimpul[baris] << " --> "
<< dataSimpul[kolom] << ": ";
            cin >> dataBusur[baris][kolom];
        }
    }
    cout << endl;
    cekMatriks = true;
}

// Hitung Jarak Terpendek
void jarakTerdekat()
{
    if (cekMatriks)
    {
        jarakDiketahui = new int[jumlahSimpul];
        kunjungan = new int[jumlahSimpul];
        for (int i = 0; i < jumlahSimpul; i++)
        {
            jarakDiketahui[i] = 999; // Nilai 999
dianggap sebagai nilai infinity (tak hingga)
            kunjungan[i] = belumDikunjungi;
        }
        kunjungan[simpulAsal] = dikunjungi;
        jarakDiketahui[simpulAsal] = 0;
        simpulSaatIni = simpulAsal;
        while (simpulSaatIni != simpulTujuan)
        {
            jarakLama = 999;
            jarakSaatIni = jarakDiketahui[simpulSaatIni];
            for (int i = 0; i < jumlahSimpul; i++)
            {
                if (kunjungan[i] == belumDikunjungi)
                {
                    jarakBaru = jarakSaatIni +
dataBusur[simpulSaatIni][i];
                    if (jarakBaru < jarakDiketahui[i])
                    {
                        jarakDiketahui[i] = jarakBaru;
                    }
                    if (jarakDiketahui[i] < jarakLama)
                    {
```

```
                        jarakLama = jarakDiketahui[i];
                        indeksPosisi = i;
                    }
                }
            }
            simpulSaatIni = indeksPosisi;
            kunjungan[simpulSaatIni] = dikunjungi;
        }
        cout << " Lintasan terpendek dari simpul indeks
ke-" << simpulAsal << " ke simpul indeks ke-" <<
simpulTujuan << " adalah " <<
jarakDiketahui[simpulTujuan] << endl;
        delete jarakDiketahui;
        delete kunjungan;
    }
}

// Tampil Data Graph
void tampilMatriks()
{
    if (cekMatriks)
    {
        cout << left << setw(4) << " ";
        for (int i = 0; i < jumlahSimpul; i++)
        {
            cout << left << setw(5) << dataSimpul[i] << "
";
        }
        cout << endl;
        for (int baris = 0; baris < jumlahSimpul;
baris++)
        {
            cout << " " << setw(5) << dataSimpul[baris]
<< left << setw(2) << " ";
            for (int kolom = 0; kolom < jumlahSimpul;
kolom++)
            {
                cout << left << setw(5) <<
dataBusur[baris][kolom] << " ";
            }
            cout << endl;
        }
        cout << endl;
    }
    else
```

```
    {
        cout << " Graph masih kosong!" << endl;
        cout << endl;
    }
}

int main()
{
    char keluar;
    cout << " Silahkan masukkan jumlah simpul: ";
    cin >> jumlahSimpul;
    buatMatriks();
    tampilMatriks();
    do
    {
        cout << " Silahkan masukkan indeks simpul asal [0
- " << jumlahSimpul - 1 << "]: ";
        cin >> simpulAsal;
        cout << " Silahkan masukkan indeks simpul tujuan
[0 - " << jumlahSimpul - 1 << "]: ";
        cin >> simpulTujuan;
        jarakTerdekat();
        cout << endl;
        cout << " Keluar? (y/t): ";
        cin >> keluar;
        cout << endl;
        if (tolower(keluar) != 'y')
        {
            system("cls");
            tampilMatriks();
        }
    } while (tolower(keluar) != 'y');
    return 0;
}
```

Deskripsi:

Yang pertama adalah melakukan Deklarasi variabel dan pointer yang nantinya akan digunakan. Ketika di run/dijalankan program akan mengeksekusi program utama, atau main program. Lalu dalam program user dapat memasukan jumlah simpul sesuai keinginan, setelah semuanya sudah di isi, maka akan memanggil fungsi untuk membuat matriks. Di fungsi buat matriks user disuruh memasukkan nama simpul

dan nilai simpul. Lalu kembali ke program utama, dan disitu memanggil fungsi untuk menampilkan matriks, untuk menampilkan dapat menggunakan pengulangan for. Kembali ke program utama, disitu user dapat memasukkan simpul awal, dan simpul tujuan, dan memanggil fungsi jarak terdekat, untuk mencari jarak terdekat antar simpul, dan akan menampilkan jarak terdekat antar simpul tersebut. setelah itu kembali ke fungsi utama, untuk menanyakan user ingin mengulang mencari simpul atau tidak, jika ingin mengulang maka akan mengulang fungsi jarak terdekat seperti tadi, jika tidak mengulang maka program akan berhenti.

Output:

```
dia --> dia: 5
dia --> mere: 4
mere --> aku: 6
mere --> kamu: 1
mere --> dia: 0
mere --> mere: 3

   aku   kamu  dia   mere
aku    0     2     5     6
kamu   4     3     1     2
dia    0     3     5     4
mere   6     1     0     3

Silahkan masukkan indeks simpul asal [0 - 3]: 0
Silahkan masukkan indeks simpul tujuan [0 - 3]: 2
Lintasan terpendek dari simpul indeks ke-0 ke simpul indeks ke-2 adalah 3

Keluar? (y/t): t

sh: 1: cls: not found
   aku   kamu  dia   mere
aku    0     2     5     6
kamu   4     3     1     2
dia    0     3     5     4
mere   6     1     0     3

Silahkan masukkan indeks simpul asal [0 - 3]: 0
Silahkan masukkan indeks simpul tujuan [0 - 3]: 3
Lintasan terpendek dari simpul indeks ke-0 ke simpul indeks ke-3 adalah 4
```

2. Program Pencarian Lintasan Terpendek Algoritma Floyd – Warshall

```
// Nama : Fatkhurrohman Purnomo
// NIM : 21102125

#include <iostream>
#include <string>
#include <iomanip>
using namespace std;
```

```
/// PROGRAM PENCARIAN LINTASAN TERPENDEK (ALGORITMA
FLOYD-WARSHALL)
// Deklarasi Global
int jumlahSimpul;
string *dataSimpul;
int **dataBusur;
int **jalurTerdekat;
bool cekMatriks = false;

// Buat Matriks Graph
void buatMatriks()
{
    dataSimpul = new string[jumlahSimpul];
    dataBusur = new int *[jumlahSimpul];
    dataBusur[0] = new int[jumlahSimpul * jumlahSimpul];

    for (int i = 1; i < jumlahSimpul; i++)
    {
        dataBusur[i] = dataBusur[i - 1] + jumlahSimpul;
    }
    cout << " Silahkan memasukkan nama simpul" << endl;

    for (int i = 0; i < jumlahSimpul; i++)
    {
        cout << " Simpul " << i + 1 << ": ";
        cin >> dataSimpul[i];
    }
    cout << endl;
    cout << " Silahkan masukkan bobot antar simpul" <<
endl;

    for (int baris = 0; baris < jumlahSimpul; baris++)
    {
        for (int kolom = 0; kolom < jumlahSimpul;
kolom++)
        {
            cout << " " << dataSimpul[baris] << " --> "
<< dataSimpul[kolom] << ": ";
            cin >> dataBusur[baris][kolom];
        }
    }
    cout << endl;
    cekMatriks = true;
}
```

```
// Hitung Lintasan Terpendek
void jarakTerdekat()
{
    if (cekMatriks)
    {
        // Membuat matriks yang sama dengan matriks
dataBusur
        jalurTerdekat = new int *[jumlahSimpul];
        jalurTerdekat[0] = new int[jumlahSimpul *
jumlahSimpul];
        for (int i = 1; i < jumlahSimpul; i++)
        {
            jalurTerdekat[i] = jalurTerdekat[i - 1] +
jumlahSimpul;
        }

        // Duplikasi isi matriks dataBusur ke dalam
matriks jalurTerdekat
        for (int baris = 0; baris < jumlahSimpul;
baris++)
        {
            for (int kolom = 0; kolom < jumlahSimpul;
kolom++)
            {
                jalurTerdekat[baris][kolom] =
dataBusur[baris][kolom];
            }
        }

        for (int k = 0; k < jumlahSimpul; k++)
        {
            for (int baris = 0; baris < jumlahSimpul;
baris++)
            {
                for (int kolom = 0; kolom < jumlahSimpul;
kolom++)
                {
                    if (jalurTerdekat[baris][k] +
jalurTerdekat[k][kolom] < jalurTerdekat[baris][kolom])
                    {
                        jalurTerdekat[baris][kolom] =
jalurTerdekat[baris][k] + jalurTerdekat[k][kolom];
                    }
                }
            }
```

```
        }

        // Tampilkan hasil
        cout << left << setw(4) << " ";
        for (int i = 0; i < jumlahSimpul; i++)
        {
            cout << left << setw(5) << dataSimpul[i] << "
";
        }
        cout << endl;

        for (int baris = 0; baris < jumlahSimpul;
baris++)
        {
            cout << " " << dataSimpul[baris] << left <<
setw(2) << " ";
            for (int kolom = 0; kolom < jumlahSimpul;
kolom++)
            {
                cout << left << setw(5) <<
jalurTerdekat[baris][kolom] << " ";
            }
            cout << endl;
        }
        cout << endl;
    }
}

// Tampil
void tampilMatriks()
{
    if (cekMatriks)
    {
        cout << left << setw(4) << " ";
        for (int i = 0; i < jumlahSimpul; i++)
        {
            cout << left << setw(5) << dataSimpul[i] << "
";
        }
        cout << endl;

        for (int baris = 0; baris < jumlahSimpul;
baris++)
        {
```

```
            cout << " " << dataSimpul[baris] << left <<
setw(2) << " ";
            for (int kolom = 0; kolom < jumlahSimpul;
kolom++)
            {
                cout << left << setw(5) <<
dataBusur[baris][kolom] << " ";
            }
            cout << endl;
        }
        cout << endl;
    }
    else
    {
        cout << " Matriks masih kosong!" << endl;
        cout << endl;
    }
}
int main()
{
    cout << " Silahkan masukkan jumlah simpul: ";
    cin >> jumlahSimpul;

    buatMatriks();
    tampilMatriks();
    jarakTerdekat();
    return 0;
}
```

Dekripsi:

Program hampir sama seperti latihan 1, hanya beda pada pencarian jarak terdekat. Pada pencarian jarak terdekat menggunakan algoritma Floyd-Warshall ini dengan melakukan duplikat array. Lalu dicari jalur yang terdekat dengan perulangan dan percabangan, setelah hasil ditemukan maka akan ditampilkan di layar.

Output:

```
Silahkan memasukkan nama simpul
Simpul 1: aku
Simpul 2: kmau
Simpul 3: dia
Simpul 4: mere

Silahkan masukkan bobot antar simpul
aku --> aku: 0
aku --> kmau: 2
aku --> dia: 5
aku --> mere: 6
kmau --> aku: 4
kmau --> kmau: 3
kmau --> dia: 1
kmau --> mere: 2
dia --> aku: 0
dia --> kmau: 3
dia --> dia: 5
dia --> mere: 4
mere --> aku: 6
mere --> kmau: 1
mere --> dia: 0
mere --> mere: 3

   aku   kmau  dia   mere
aku  0     2     5     6
kmau  4     3     1     2
dia  0     3     5     4
mere  6     1     0     3

   aku   kmau  dia   mere
aku  0     2     3     4
kmau  1     3     1     2
dia  0     2     3     4
mere  0     1     0     3
```

C. Tugas (Unguided)

Modifikasi program Guided I (Algoritma Dijkstra) dalam pencarian jalur terpendek agar menampilkan data nama simpulnya, dan buatlah menjadi program menu dengan menggabungkan algoritma Dijkstra dan Floyd – Warshall

```
// Nama : Fatkhurrohman Purnomo
// NIM : 21102125

#include <iostream>
```

```
#include <string>
#include <iomanip>
using namespace std;

/// PROGRAM PENCARIAN LINTASAN TERPENDEK (ALGORITMA
DIJKSTRA)
// Deklarasi Global
int jumlahSimpul;
string *dataSimpul;
int **dataBusur;
int **jalurTerdekat;
int *jarakDiketahui;
int *kunjungan;
int indeksPosisi, simpulSaatIni, simpulAsal, simpulTujuan,
jarakSaatIni, jarakLama, jarakBaru;
int dikunjungi = 1;
int belumDikunjungi = 0;
bool cekMatriks = false;

// Buat Graph
void buatMatriks()
{
    dataSimpul = new string[jumlahSimpul];
    dataBusur = new int *[jumlahSimpul];
    dataBusur[0] = new int[jumlahSimpul * jumlahSimpul];
    for (int i = 1; i < jumlahSimpul; i++)
    {
        dataBusur[i] = dataBusur[i - 1] + jumlahSimpul;
    }
    cout << " Silahkan masukkan nama simpul" << endl;

    for (int i = 0; i < jumlahSimpul; i++)
    {
        cout << " Simpul " << i + 1 << ": ";
        cin >> dataSimpul[i];
    }
    cout << endl;
    cout << " Silahkan masukkan bobot antar simpul" << endl;

    for (int baris = 0; baris < jumlahSimpul; baris++)
    {
        for (int kolom = 0; kolom < jumlahSimpul; kolom++)
        {
            cout << " " << dataSimpul[baris] << " --> " <<
dataSimpul[kolom] << ": ";
```

```
            cin >> dataBusur[baris][kolom];
        }
    }
    cout << endl;
    cekMatriks = true;
}

// Hitung Jarak Terpendek dengan Algoritma Dijkstra
void jarakTerdekatSatu()
{
    if (cekMatriks)
    {
        jarakDiketahui = new int[jumlahSimpul];
        kunjungan = new int[jumlahSimpul];

        for (int i = 0; i < jumlahSimpul; i++)
        {
            jarakDiketahui[i] = 999; // Nilai 999 dianggap
sebagai nilai infinity (tak hingga)
            kunjungan[i] = belumDikunjungi;
        }
        kunjungan[simpulAsal] = dikunjungi;
        jarakDiketahui[simpulAsal] = 0;
        simpulSaatIni = simpulAsal;

        while (simpulSaatIni != simpulTujuan)
        {
            jarakLama = 999;
            jarakSaatIni = jarakDiketahui[simpulSaatIni];
            for (int i = 0; i < jumlahSimpul; i++)
            {
                if (kunjungan[i] == belumDikunjungi)
                {
                    jarakBaru = jarakSaatIni +
dataBusur[simpulSaatIni][i];
                    if (jarakBaru < jarakDiketahui[i])
                    {
                        jarakDiketahui[i] = jarakBaru;
                    }
                    if (jarakDiketahui[i] < jarakLama)
                    {
                        jarakLama = jarakDiketahui[i];
                        indeksPosisi = i;
                    }
                }
```

```
            }
            simpulSaatIni = indeksPosisi;
            kunjungan[simpulSaatIni] = dikunjungi;
        }
        cout << " Hasil dari algoritma Dijkstra adalah: " <<
endl;
        cout << " Lintasan terpendek dari simpul indeks ke-"
<< simpulAsal << " (" << dataSimpul[simpulAsal] << ") " <<
"ke simpul indeks ke-" << simpulTujuan << " (" <<
dataSimpul[simpulTujuan] << ")" << " adalah " <<
jarakDiketahui[simpulTujuan] << endl;


        delete jarakDiketahui;
        delete kunjungan;
    }
}

// Hitung Lintasan Terpendek dengan menggabungkan algoritma
Dijkstra dan Floyd – Warshall
void jarakTerdekatDua()
{
    if (cekMatriks)
    {
        // Membuat matriks yang sama dengan matriks
dataBusur
        jalurTerdekat = new int *[jumlahSimpul];
        jalurTerdekat[0] = new int[jumlahSimpul *
jumlahSimpul];
        for (int i = 1; i < jumlahSimpul; i++)
        {
            jalurTerdekat[i] = jalurTerdekat[i - 1] +
jumlahSimpul;
        }

        // Duplikasi isi matriks dataBusur ke dalam matriks
jalurTerdekat
        for (int baris = 0; baris < jumlahSimpul; baris++)
        {
            for (int kolom = 0; kolom < jumlahSimpul;
kolom++)
            {
                jalurTerdekat[baris][kolom] =
dataBusur[baris][kolom];
            }
```

```
        }

        for (int k = 0; k < jumlahSimpul; k++)
        {
            for (int baris = 0; baris < jumlahSimpul;
baris++)
            {
                for (int kolom = 0; kolom < jumlahSimpul;
kolom++)
                {
                    if (jalurTerdekat[baris][k] +
jalurTerdekat[k][kolom] < jalurTerdekat[baris][kolom])
                    {
                        jalurTerdekat[baris][kolom] =
jalurTerdekat[baris][k] + jalurTerdekat[k][kolom];
                    }
                }
            }
        }

        // Tampilkan hasil
        cout << " Hasil dari algoritma Floyd-Warshall: " <<
endl;
        cout << left << setw(4) << " ";
        for (int i = 0; i < jumlahSimpul; i++)
        {
            cout << left << setw(5) << dataSimpul[i] << " ";
        }
        cout << endl;

        for (int baris = 0; baris < jumlahSimpul; baris++)
        {
            cout << " " << setw(5) <<  dataSimpul[baris] <<
left << setw(2) << " ";
            for (int kolom = 0; kolom < jumlahSimpul;
kolom++)
            {
                cout << left << setw(5) <<
jalurTerdekat[baris][kolom] << " ";
            }
            cout << endl;
        }
        cout << endl;
    }
}
```

```
// Tampil Data Graph
void tampilMatriks()
{
    if (cekMatriks)
    {
        cout << " Data Graph: " << endl;
        cout << left << setw(4) << " ";
        for (int i = 0; i < jumlahSimpul; i++)
        {
            cout << left << setw(5) << dataSimpul[i] << " ";
        }
        cout << endl;

        for (int baris = 0; baris < jumlahSimpul; baris++)
        {
            cout << " " << setw(5) << dataSimpul[baris] <<
left << setw(2) << " ";
            for (int kolom = 0; kolom < jumlahSimpul;
kolom++)
            {
                cout << left << setw(5) <<
dataBusur[baris][kolom] << " ";
            }
            cout << endl;
        }
        cout << endl;
    }
    else
    {
        cout << " Graph masih kosong!" << endl;
        cout << endl;
    }
}

int main()
{
    char keluar;
    int pilih;
    cout << " Silahkan masukkan jumlah simpul: ";
    cin >> jumlahSimpul;
    cout << endl;
    buatMatriks();
    tampilMatriks();
```

```
    do
    {
        cout << endl <<
"================================================" <<
endl;
        cout << "           Mencari lintasan terpendek" <<
endl;
        cout <<
"================================================" <<
endl;
        cout << " Silahkan pilih menu: " << endl;
        cout << " 1. Algortitma Dijkstra" << endl;
        cout << " 2. Algoritma Floyd-Warshall" << endl;
        cout << " 3. Dua Algoritma" << endl;
        cout << " 4. Keluar" << endl;
        cout << " Pilih: ";
        cin >> pilih;
        cout << endl;

        switch (pilih)
        {
        case 1:
            cout << " Silahkan masukkan indeks simpul asal
[0 - " << jumlahSimpul - 1 << "]: ";
            cin >> simpulAsal;
            cout << " Silahkan masukkan indeks simpul tujuan
[0 - " << jumlahSimpul - 1 << "]: ";
            cin >> simpulTujuan;
            cout << endl;
            jarakTerdekatSatu();
            break;
        case 2:
            jarakTerdekatDua();
            break;
        case 3:
            cout << " Silahkan masukkan indeks simpul asal
[0 - " << jumlahSimpul - 1 << "]: ";
            cin >> simpulAsal;
            cout << " Silahkan masukkan indeks simpul tujuan
[0 - " << jumlahSimpul - 1 << "]: ";
            cin >> simpulTujuan;
            cout << endl;
            jarakTerdekatSatu();
            cout << endl;
            jarakTerdekatDua();
```

```
            break;
        default:
            break;
        }
    } while (pilih != 4);   // Selama pilih bukan 4, maka
akan mengulangi program

    return 0;
}
```

Deskripsi:

Program sama seperti pada latihan 2 dan 3. Hanya ditambahkan dengan menu supaya user dapat lebih mudah memilih pencarian algoritma yang di inginkan, dan juga ada menu untuk menampilkan dari semua versi pencarian jalur terpendek.

Output:

a. Input yang dimasukan

```
Silahkan masukkan jumlah simpul: 4

Silahkan masukkan nama simpul
Simpul 1: aku
Simpul 2: kamu
Simpul 3: dia
Simpul 4: mere

Silahkan masukkan bobot antar simpul
aku --> aku: 0
aku --> kamu: 2
aku --> dia: 5
aku --> mere: 6
kamu --> aku: 4
kamu --> kamu: 3
kamu --> dia: 1
kamu --> mere: 2
dia --> aku: 0
dia --> kamu: 3
dia --> dia: 5
dia --> mere: 4
mere --> aku: 6
mere --> kamu: 1
mere --> dia: 0
mere --> mere: 3

Data Graph:
    aku    kamu   dia    mere
aku     0      2      5      6
kamu    4      3      1      2
dia     0      3      5      4
mere    6      1      0      3
```

b. Output dari Algoritma Dijkastra

```
==================================================
          Mencari lintasan terpendek
==================================================
Silahkan pilih menu:
1. Algortitma Dijkstra
2. Algoritma Floyd-Warshall
3. Dua Algoritma
4. Keluar
Pilih: 1

Silahkan masukkan indeks simpul asal [0 - 3]: 0
Silahkan masukkan indeks simpul tujuan [0 - 3]: 2

Hasil dari algoritma Dijkstra adalah:
Lintasan terpendek dari simpul indeks ke-0 (aku) ke simpul indeks ke-2 (dia) adalah 3
```

c. Output dari Algoritma Floyd-Warshall

```
==================================================
          Mencari lintasan terpendek
==================================================
Silahkan pilih menu:
1. Algortitma Dijkstra
2. Algoritma Floyd-Warshall
3. Dua Algoritma
4. Keluar
Pilih: 2

Hasil dari algoritma Floyd-Warshall:
    aku    kamu   dia    mere
aku     0      2      3      4
kamu    1      3      1      2
dia     0      2      3      4
mere    0      1      0      3
```

d. Output dari kedua Algoritma

```
==================================================
          Mencari lintasan terpendek
==================================================
Silahkan pilih menu:
1. Algortitma Dijkstra
2. Algoritma Floyd-Warshall
3. Dua Algoritma
4. Keluar
Pilih: 3

Silahkan masukkan indeks simpul asal [0 - 3]: 1
Silahkan masukkan indeks simpul tujuan [0 - 3]: 0

Hasil dari algoritma Dijkstra adalah:
Lintasan terpendek dari simpul indeks ke-1 (kamu) ke simpul indeks ke-0 (aku) adalah 1

Hasil dari algoritma Floyd-Warshall:
   aku    kamu   dia    mere
aku     0      2      3      4
kamu    1      3      1      2
dia     0      2      3      4
mere    0      1      0      3
```

D. Kesimpulan

1. Bisa membuat graph II (mencari lintasan terpendek)
2. Belajar lebih dalam pengaplikasian graph dengan lebih baik
3. Lebih mahir dalam menggunakan bahasa C++
4. Bisa melakukan problem solving bagi program yang error
5. Lebih paham dalam membuat program
6. Melatih daya pikir, imajinasi, dan langkah-langkah dalam membuat program
7. Belajar algoritma Dijkastra

8. Belajar algoritma Floyd-Warshall