**PRAKTIKUM 14**

**GRAPH**

****

**Dosen Pengampu :**

**Entin Martiana Kusumaningtyas,S.Kom,M.Kom**

**Oleh:**

**Zahrotul Hidayah**

**D3 IT A**

**3122500004**

**Politeknik Elektronika Negeri Surabaya**

**Tahun Akademik**

**2022/2023**

# 

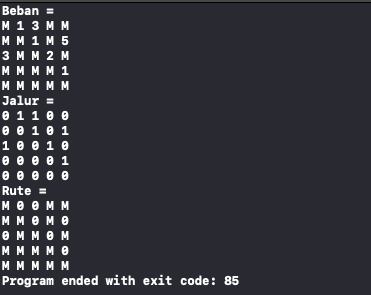
# PERCOBAAN

**Percobaan 1 : Mendeklarasikan matriks Beban, Jalur dan Rute**

**Source code**

|  |
| --- |
| #include<stdio.h>  #define N 5  #define M 1000  void Tampil(int data[N][N], char \*judul)  {  printf("%s = \n",judul);  for(int i=0; i<N; i++) {  for(int j=0; j<N; j++)  if(data[i][j] >= M)  printf("M ");  else  printf("%d ", data[i][j]);  printf("\n");  }  }  void main()  {  int Beban[N][N] = {  {M,1,3,M,M},  {M,M,1,M,5},  {3,M,M,2,M},  {M,M,M,M,1},  {M,M,M,M,M}};  int Jalur[N][N] = {  {0,1,1,0,0},  {0,0,1,0,1},  {1,0,0,1,0},  {0,0,0,0,1},  {0,0,0,0,0}};  int Rute[N][N] = {  {M,0,0,M,M},  {M,M,0,M,0},  {0,M,M,0,M},  {M,M,M,M,0},  {M,M,M,M,M}};  Tampil(Beban, "Beban");  Tampil(Jalur, "Jalur");  Tampil(Rute, "Rute");  } |

## Output



## Analisa

Matriks Beban merepresentasikan beban pada setiap simpul dalam suatu jaringan, dengan nilai M menunjukkan keterbatasan atau ketidak hubungan antara simpul-simpul. Matriks Jalur menunjukkan ketersediaan jalur antara simpul-simpul dalam jaringan, dengan angka 1 menunjukkan jalur yang tersedia dan angka 0 menunjukkan ketiadaan jalur. Matriks Rute merepresentasikan rute terpendek antara dua simpul, dengan nilai M menunjukkan ketiadaan rute dan angka 0 menunjukkan simpul itu sendiri.

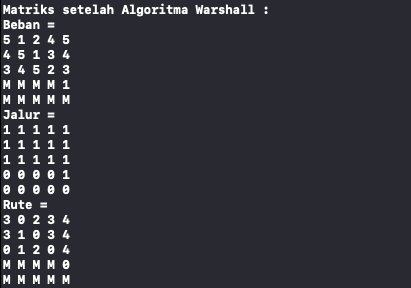
Fungsi Tampil digunakan untuk menampilkan isi matriks dengan format tertentu. Program ini tidak menampilkan algoritma atau perhitungan yang menggunakan matriks-matriks tersebut. Dalam contoh ini, matriks-matriks tersebut diinisialisasi dengan nilai-nilai awal sesuai diagram graph.

## Percobaan 2 : Algoritma Warshall untuk pencarian jalur terpendek multipath

## Listing code

|  |
| --- |
| #include<stdio.h>  #define N 5  #define M 1000  void Tampil(int data[N][N], char \*judul)  {  printf("%s = \n",judul);  for(int i=0; i<N; i++) {  for(int j=0; j<N; j++)  if(data[i][j] >= M)  printf("M ");  else  printf("%d ", data[i][j]);  printf("\n");  }  }  void Warshall(int Q[N][N], int P[N][N], int R[N][N])  {  for(int k=0; k<N; k++)  for (int i=0; i<N; i++)  for (int j=0; j<N; j++){  P[i][j] = P[i][j] | (P[i][k] & P[k][j]);  if ((Q[i][k] + Q[k][j]) < Q[i][j]) {  Q[i][j] = Q[i][k] + Q[k][j];  if (R[k][j] == 0)  R[i][j] = k+1;  else  R[i][j] = R[k][j];  }  }    }  void main()  {  int Beban[N][N] = {  {M,1,3,M,M},  {M,M,1,M,5},  {3,M,M,2,M},  {M,M,M,M,1},  {M,M,M,M,M}};  int Jalur[N][N] = {  {0,1,1,0,0},  {0,0,1,0,1},  {1,0,0,1,0},  {0,0,0,0,1},  {0,0,0,0,0}};  int Rute[N][N] = {  {M,0,0,M,M},  {M,M,0,M,0},  {0,M,M,0,M},  {M,M,M,M,0},  {M,M,M,M,M}};    Tampil(Beban, "Beban");  Tampil(Jalur, "Jalur");  Tampil(Rute, "Rute");  Warshall(Beban, Jalur, Rute);  printf("Matriks setelah Algoritma Warshall : \n");  Tampil(Beban, "Beban");  Tampil(Jalur, "Jalur");  Tampil(Rute, "Rute");  } |

## Output



## Analisa

Fungsi Warshall dalam program di atas adalah implementasi algoritma Warshall yang digunakan untuk mencari jalur terpendek antara simpul-simpul dalam suatu jaringan. Fungsi ini menggunakan tiga matriks sebagai parameter: Q untuk menyimpan beban atau jarak antara simpul-simpul, P untuk menyimpan informasi jalur yang tersedia, dan R untuk menyimpan informasi simpul pertama yang harus dilalui dalam rute terpendek. Dalam setiap iterasi, algoritma memeriksa kemungkinan adanya jalur yang lebih pendek atau jalur alternatif menggunakan operasi bitwise dan pembandingan. Akhirnya, matriks Q dan R diperbarui sesuai dengan jalur terpendek yang ditemukan.

# LATIHAN

1. Dari percobaan 2 diatas, tambahkan fungsi untuk menampilkan rute dari satu titik ke titik berikutnya berdasarkan matriks rute. Input berupa titik awal dan titik akhir. Gunakan struktur data Stack untuk mencari rute seperti langkah-langkah di bawah ini.

1. Rute 1-5?

2. Ambil nilai di baris 1, kolom 5 = 4 → push

3. Ambil nilai di baris 1, kolom 4 = 3 → push

4. Ambil nilai di baris 1, kolom 3 = 2 → push

5. Ambil nilai di baris 1, kolom 2 = 0 (stop) → pop sampai stack kosong

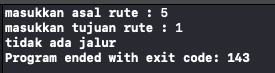
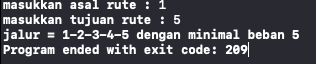
6. Sehingga rutenya menjadi 1-2-3-4-5 dengan beban minimal 5.

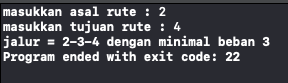
Beban minimal diperoleh dari matriks beban dengan nilai baris, kolom sama dengan 1,5. Lakukan hal yang sama untuk input titik awal dan titik akhir yang lain.

## Listing code

|  |
| --- |
| ​​#include<stdio.h>  #include <stdlib.h>  #define N 5  #define M 1000  typedef struct simpul Node;  struct simpul {  int nilai;  Node \*next;  };  void Tampil(int data[N][N], char \*judul)  {  printf("%s = \n",judul);  for(int i=0; i<N; i++) {  for(int j=0; j<N; j++)  if(data[i][j] >= M)  printf("M ");  else  printf("%d ", data[i][j]);  printf("\n");  }  }  void Warshall(int Q[N][N], int P[N][N], int R[N][N])  {  for(int k=0; k<N; k++)  for (int i=0; i<N; i++)  for (int j=0; j<N; j++){  P[i][j] = P[i][j] | (P[i][k] & P[k][j]);  if ((Q[i][k] + Q[k][j]) < Q[i][j]) {  Q[i][j] = Q[i][k] + Q[k][j];  if (R[k][j] == 0)  R[i][j] = k+1;  else  R[i][j] = R[k][j];  }  }    }  void showRute(int Q[N][N], int P[N][N], int R[N][N]){  int beban, start, end, push=1;    printf("masukkan asal rute : ");  scanf("%d", &start);  printf("masukkan tujuan rute : ");  scanf("%d", &end);  start--;  end--;  beban = Q[start][end];    Node \*rute = (Node \*) malloc(sizeof(Node));  rute->nilai = end;  rute->next = NULL;    while (push == 1) {  if (R[start][end] == M) {  printf("tidak ada jalur\n");  break;  } else if (R[start][end] == 0){  printf("jalur = %d", ++start);  while (rute->next != NULL) {  printf("-%d", rute->nilai);  rute = rute->next;  }  printf("-%d dengan minimal beban %d\n", ++rute->nilai, beban);  break;  }    Node \*baru = (Node \*) malloc(sizeof(Node));  baru->nilai = R[start][end];    baru->next = rute;  rute = baru;    end = R[start][end]-1;  }  }  void main()  {  int Beban[N][N] = {  {M,1,3,M,M},  {M,M,1,M,5},  {3,M,M,2,M},  {M,M,M,M,1},  {M,M,M,M,M}};  int Jalur[N][N] = {  {0,1,1,0,0},  {0,0,1,0,1},  {1,0,0,1,0},  {0,0,0,0,1},  {0,0,0,0,0}};  int Rute[N][N] = {  {M,0,0,M,M},  {M,M,0,M,0},  {0,M,M,0,M},  {M,M,M,M,0},  {M,M,M,M,M}};    Warshall(Beban, Jalur, Rute);  showRute(Beban, Jalur, Rute);  } |

## Output

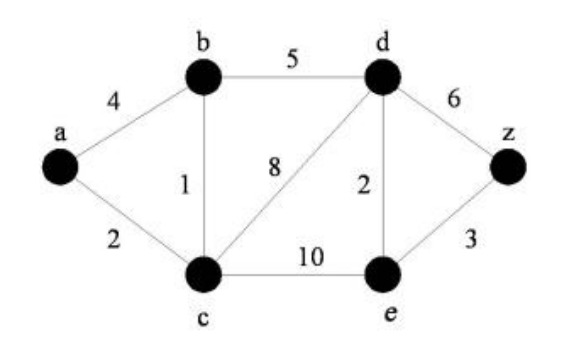




## Analisa

Fungsi showRute dalam program di atas bertujuan untuk menampilkan jalur terpendek antara dua simpul dalam suatu jaringan berdasarkan hasil dari algoritma Warshall. Fungsi ini meminta input dari pengguna berupa simpul asal dan simpul tujuan, kemudian menggunakan informasi dari matriks Rute untuk melacak jalur terpendek yang ada. Selama proses pelacakan, fungsi ini membangun sebuah linked list yang merepresentasikan jalur dari simpul asal ke simpul tujuan. Setelah selesai, jalur terpendek beserta bebannya ditampilkan kepada pengguna. Jika tidak ada jalur yang tersedia antara dua simpul tersebut, akan ditampilkan pesan tidak ada jalur.

1. Berdasarkan graph di bawah ini, representasikan matriks, gunakan algoritma warshall untuk mencari rute terpendek dan rute seperti pada Latihan 1.



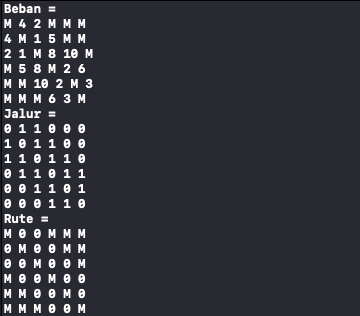
**Source code**

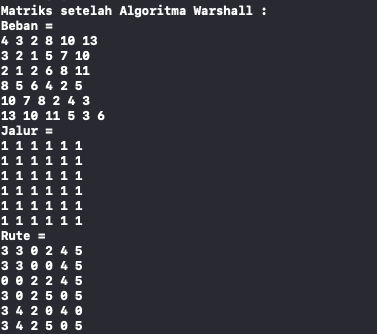
|  |
| --- |
| #include<stdio.h>  #include <stdlib.h>  #define N 6  #define M 1000  typedef struct simpul Node;  struct simpul {  int nilai;  Node \*next;  };  void Tampil(int data[N][N], char \*judul)  {  printf("%s = \n",judul);  for(int i=0; i<N; i++) {  for(int j=0; j<N; j++)  if(data[i][j] >= M)  printf("M ");  else  printf("%d ", data[i][j]);  printf("\n");  }  }  void Warshall(int Q[N][N], int P[N][N], int R[N][N])  {  for(int k=0; k<N; k++)  for (int i=0; i<N; i++)  for (int j=0; j<N; j++){  P[i][j] = P[i][j] | (P[i][k] & P[k][j]);  if ((Q[i][k] + Q[k][j]) < Q[i][j]) {  Q[i][j] = Q[i][k] + Q[k][j];  if (R[k][j] == 0)  R[i][j] = k+1;  else  R[i][j] = R[k][j];  }  }    }  void showRute(int Q[N][N], int P[N][N], int R[N][N]){  int beban, start, end, push=1;    printf("masukkan asal rute : ");  scanf("%d", &start);  printf("masukkan tujuan rute : ");  scanf("%d", &end);  start--;  end--;  beban = Q[start][end];    Node \*rute = (Node \*) malloc(sizeof(Node));  rute->nilai = end;  rute->next = NULL;    while (push == 1) {  if (R[start][end] == M) {  printf("tidak ada jalur\n");  break;  } else if (R[start][end] == 0){  printf("jalur = %d", ++start);  while (rute->next != NULL) {  printf("-%d", rute->nilai);  rute = rute->next;  }  printf("-%d dengan minimal beban %d\n", ++rute->nilai, beban);  break;  }    Node \*baru = (Node \*) malloc(sizeof(Node));  baru->nilai = R[start][end];    baru->next = rute;  rute = baru;    end = R[start][end]-1;  }  }  void main()  {  int Beban[N][N] = {  {M,4,2,M,M,M},  {4,M,1,5,M,M},  {2,1,M,8,10,M},  {M,5,8,M,2,6},  {M,M,10,2,M,3},  {M,M,M,6,3,M}  };  int Jalur[N][N] = {  {0,1,1,0,0,0},  {1,0,1,1,0,0},  {1,1,0,1,1,0},  {0,1,1,0,1,1},  {0,0,1,1,0,1},  {0,0,0,1,1,0}  };  int Rute[N][N] = {  {M,0,0,M,M,M},  {0,M,0,0,M,M},  {0,0,M,0,0,M},  {M,0,0,M,0,0},  {M,M,0,0,M,0},  {M,M,M,0,0,M}  };    // Tampil(Beban, "Beban");  // Tampil(Jalur, "Jalur");  // Tampil(Rute, "Rute");  // Warshall(Beban, Jalur, Rute);  // printf("Matriks setelah Algoritma Warshall : \n");  // Tampil(Beban, "Beban");  // Tampil(Jalur, "Jalur");  // Tampil(Rute, "Rute");    Warshall(Beban, Jalur, Rute);  showRute(Beban, Jalur, Rute);    } |

## 

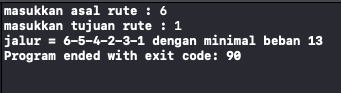
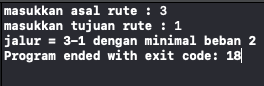
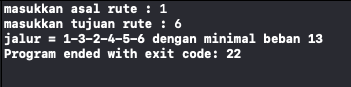
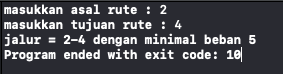
## Output

### Marshall





### Cari Rute



## Analisa

Hanya mengubah matriks menjadi 6 x 6 dengan mengganti nilai N serta menyesuaikan isi matriks dengan diagram graph yang diberikan. Graph yang tidak memiliki arah atau tanda panah diartikan sebagai jalur dua arah.

**KESIMPULAN**

* Representasi dan pemrosesan data: Graph dapat digunakan untuk merepresentasikan hubungan dan koneksi antara elemen-elemen data. Dalam bahasa C, graf dapat diimplementasikan menggunakan struktur data seperti matriks ketetanggaan atau daftar adjacency.
* Algoritma graph: Graph memberikan dasar untuk berbagai algoritma yang penting dalam pemrosesan graf. Misalnya, algoritma pencarian graf seperti Depth-First Search (DFS) dan Breadth-First Search (BFS) digunakan untuk mencari jalur atau elemen tertentu dalam graph. Algoritma Dijkstra digunakan untuk mencari jalur terpendek antara dua simpul dalam graf yang terhubung.
* Pohon dan jaringan: Pohon merupakan jenis graph khusus yang memiliki struktur hierarkis tanpa siklus. Dalam bahasa C, pohon dapat diimplementasikan menggunakan struktur data seperti struktur node dengan pointer ke simpul anak. Algoritma seperti traversal pre-order, in-order, dan post-order dapat digunakan untuk memanipulasi dan memproses pohon.
* Graph berbobot: Graph berbobot adalah graph di mana setiap sisi atau koneksi antara simpul memiliki bobot atau nilai tertentu. Dalam bahasa C, graf berbobot dapat diimplementasikan dengan menggunakan matriks ketetanggaan dengan nilai bobot yang sesuai pada setiap elemen matriks. Algoritma-algoritma seperti algoritma Dijkstra dan algoritma aliran maksimum (seperti algoritma Ford-Fulkerson) berguna dalam memproses graf berbobot.
* Analisis kompleksitas: Graph memainkan peran penting dalam menganalisis kompleksitas algoritma. Melalui representasi graph dan pemrosesan graph yang efisien, kita dapat memperkirakan kompleksitas waktu dan ruang dari algoritma yang menggunakan struktur graph.

Jadi kesimpulannya adalah graph menyediakan kerangka kerja yang penting untuk merepresentasikan, memanipulasi, dan memproses hubungan dan koneksi antara elemen data. Dengan memahami konsep dan algoritma graph, kita dapat mengembangkan solusi yang efisien dan efektif untuk berbagai masalah yang melibatkan struktur data graph.