LAPORAN MATEMATIKA DISKRIT
PENGIMPLEMENTASIAN ALGORITMA TEORI GRAF DAN TREE DALAM PEMROGRAMAN BAHASA C

Oleh :

Nama : Muhammad Firyanul Rizky

NIM : 1708561006

Kelas : A

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS UDAYANA
2018

BAB I
PENJELASAN ALGORITMA PADA GRAPH

1.1 PENJELASAN FUNGSI PEWARNAAN GRAPH (GRAPH COLORING)

1.1.1. SYNTAX FUNGSI PEWARNAAN

```
void pewarnaan(){
    printf("\nMenu 1 ( Algoritma Graph )\n\n");
   printf("-- PEWARNAAN GRAPH --\n\n");
    int n,i,j,k,temp,eror=0,warna=0;
    //Inisialisasi Variabel & Input
   printf("Masukkan jumlah vertek = ");scanf("%d",&n);
   int m[n][n],count[n],colored[n],sort[n],save[n];
   printf("Hubungan matriks = \n\n  ");
   for(i=0;i<n;i++){
         printf("%c ",i+65);
   }
   printf("\n");
   for(i=0;i<n;i++){
         printf("%c ",i+65);
         for(j=0;j<n;j++){
               scanf("%d",&m[i][j]);
         }
   }

//SYNTAX DIBAWAH INI ADALAH LANGKAH PENGISIAN ARRAY
COUNT (REPRESENTASI DERAJAT VERTEK)
//1. Bertujuan untuk mengisi array count(derajat) menjadi
0
//2. Sort(urutan)sesuai dengan urutan vertek dari 0
sampai n
//3. Dalam langkah ini semua vertek belum diwarnai
   for(i=0;i<n;i++){
         count[i]=0;
         sort[i]=i;
         colored[i]=0;
//4. Hitung derajat yang disimpan pada array count
         for(j=0;j<n;j++){
               count[i]+=m[i][j];
         }
   }
//<END, LANJUT KE SYNTAX & TAHAPAN ALGORITMA BERIKUTNYA>

//SYNTAX DIBAWAH INI ADALAH LANGKAH SORTING DERAJAT
VERTEK
//1. Jika derajat vertek setelahnya pada array sort lebih
besar dari vertek sekarang
//2. Maka, terlebih dahulu urutan vertek selanjutnya akan
dipindahkan ke urutan vertek yang sekarang
   for(i=0;i<n-1;i++){
         for(j=0;j<n-i-1;j++){
               if(count[j+1]>count[j]){
                     temp=count[j];
                     count[j]=count[j+1];
                     count[j+1]=temp;
```

```
                        temp=sort[j];
                        sort[j]=sort[j+1];
                        sort[j+1]=temp;
                  }
            }
      }
//<END, LANJUT KE SYNTAX & TAHAPAN ALGORITMA BERIKUTNYA>

//SYNTAX DIBAWAH INI ADALAH LANGKAH PEWARNAAN DENGAN
VERTEK SEBAGAI PERTIMBANGAN TITIK AWAL & SELANJUTNYA
   for(i=0;i<n;i++){
//1. Inisialisasi save(menyimpan vertek yang tidak saling
berhubungan)
            for(k=0;k<n;k++){
               save[k] = -1;
           }
//2. Jika vertek pada urutan ke i belum diwarnai maka
akan dilakukan proses pewarnaan
            if(colored[sort[i]]==0){
                  ++warna;
                  colored[sort[i]]=warna;
                  for(j=0;j<n;j++){
//3. Jika vertek urutan ke i dan urutan ke j tidak saling
berhubungan dan vertek j belum diwarnai, maka
    //syntax dibawah ini adalah langkah selanjutnya
if(m[sort[i]][sort[j]]==0&&colored[sort[j]]==0){
//4. Mengecek apakah vertek urutan ke j memiliki hubungan
terhadap vertek lain
    //dan tidak pula berhubungan dengan vertek i serta
sudah lebih dulu diwarnai
                        for(k=0;k<n;k++){
                              if(save[k]!=-1){
                  if(m[save[k]][sort[j]]!=0){
                                                eror=1;
                                    }
                              }
                        }
//5. Langkah dibawah adalah representasi Jika ada vertek
urutan ke j
    // tidak memiliki hubungan terhadap vertek lain dan
tidak pula berhubungan
    // dengan vertek i yang sudah diwarnai sebelumnya
//6. Selanjutnya Vertek urutan ke j akan diberi warna
yang sama dengan vertek urutan ke i
                        if(eror==0){
                              colored[sort[j]]=warna;
                              save[j]=sort[j];
//7. Menyimpan vertek urutan ke-j kedalam array save
untuk mengecek apakah vertek lain tidak saling terhubung
    //dengan vertek urutan ke i
//8. Jika memiliki hubungan, maka akan menimbulkan
konflik dengan vertek lain
    //dan representasi 0 adalah tanda bahwa vertek urutan
ke-j tidak memiliki hubungan dengan vertek urutan ke i
                              }
                        eror=0;
                        }
                  }
            }
```

```
   }
//<END, LANJUT KE SYNTAX & TAHAPAN ALGORITMA BERIKUTNYA>

//SYNTAX DIBAWAH INI ADALAH LANGKAH DALAM MENAMPILKAN
HASIL AKHIR
   for(i=0;i<n;i++){
    printf("\nwarna vertek %c = warna
%d",i+65,colored[i]);
   }
   printf("\n\ntotal bilangan kromatik = %d\n\n",warna);
}
//<END, PEWARNAAN GRAPH SUDAH SELESAI>
```

1.1.2. TAHAPAN ALGORITMA DARI FUNGSI PEWARNAAN

1. Pada fungsi ini kita membutuhkan variabel yang menyimpan jumlah vertek (n) , variabel untuk indicator jika terjadi konflik antarr vertek / eror (eror) , variabel untuk menghitung jumlah warna (bilangan kromatik) (warna) , dan variabel untuk menghitung jumlah vertek yang di warnai.

   ```
   int n,i,j,k,temp,eror=0,warna=0;
   ```

2. Setelah menginputkan jumlah vertek , kita akan memerlukan array 2 dimensi berukuran n x n untuk menyimpan matriks ketetanggaan (m[n][n]) , array untuk menyimpan derajat setiap vertek (count[n]) , array untuk menyimpan indicator bahwa vertek sudah diwarnai atau belum (colored[n]) , array untuk menyimpan urutan vertek berdasarkan derajat secara descending (sort [n]) , dan juga array untuk menyimpan sementara vertek yang tidak berhubungan dengan suatu vertek.

   ```
   int m[n][n],count[n],colored[n],sort[n],save[n];
   ```

3. Melakukan input matriks , matriks yang diinput disini adalah matriks ketetanggaan.

   ```
   printf("Hubungan matriks = \n  ");
   for(i=0;i<n;i++){
         printf("%c ",i+65);
   }
   printf("\n");
   for(i=0;i<n;i++){
         printf("%c ",i+65);
         for(j=0;j<n;j++){
               scanf("%d",&m[i][j]);
         }
   }
   ```

4. Menginisialisasi derajat awal semua vertek adalah 0 , semua vertek belum di warnai , dan urutan vertek sesuai dengan nilai vertek (0 – n). Setelah inisialisasi , kita akan menghitung derajat vertek , dikarenakan input matriks adalah matriks ketetanggaan yang menunjukan berhubungan (1) atau tidak (0) , jadi kita bisa menghitung derajat vertek I dengan cara menambahkan count[i] dengan matriks ketetanggaan baris I di setiap kolom (m[i][j]).

```
for(i=0;i<n;i++){
	count[i]=0;
	sort[i]=i;
	colored[i]=0;
	for(j=0;j<n;j++){
		count[i]+=m[i][j];
	}
}
```

5. Melakukan sorting terhadap semua vertek berdasarkan derajatnya secara descending. Jika derajat vertek j+1 (vertek selanjutnya dari vertek yang di cek) lebih besar dari vertek j (vertek yang di cek) , maka kita akan menukar posisi count dan sort kedua vertek tersebut. Dengan begini array sort akan berisi urutan vertek berdasarkan derajatnya secara descending.

```
	for(i=0;i<n-1;i++){
		for(j=0;j<n-i-1;j++){
			if(count[j+1]>count[j]){
				temp=count[j];
				count[j]=count[j+1];
				count[j+1]=temp;
				temp=sort[j];
				sort[j]=sort[j+1];
				sort[j+1]=temp;
			}
		}
	}
```

6. Melakukan coloring

- ✓ Melakukan inisialisasi terhadap array save , bahwa belum ada vertek yang disimpan , dimana veretek yang disimpan adalah vertek yang tidak memiliki hubungan pada vertek yang akan di cek
- ✓ Jika vertek urutan ke-I belum di warnai maka berlanjut pada proses pewarnaan. Warna (bilangan kromatik) akan bertambah , vertek urutan ke-I akan diberi warna sesuai dengan bilangn kromatik pada saat pewarnaan.
- ✓ Setelah itu program akan mencari vertek yang tidak berhubungan dengan vertek urutan ke-I dan juga belum di warnai. Setelah itu program akan cek vertek tersebut membandingkannya dengan array save pada saat pengecekan tersebut. Jika vertek tersebut memiliki hubungan terhadap vertek yang berada pada array save , maka vertek tersebut memiliki konflik , sehingga nilai indicator eror = 1.

- ✓ Jika vertek tersebut tidak memiliki konflik terhadap vertek yang ada di array save (eror = 0) maka vertek tersebut akan diwarnai dengan warna yang sama dengan vertek urutan ke- I (bilangan kromatik saat pengecekan) dan vertek tersebut akan disimpan di array save. Dan kita akan set nilai eror menjadi 0 kembali.
- ✓ Proses ini akan dilakukan sebanyak jumlah vertek , karena kita akan mewarnai semua vertek.

```
for(i=0;i<n;i++){
      for(k=0;k<n;k++){
      save[k] = -1;
  }
      if(colored[sort[i]]==0){
            ++warna;
            colored[sort[i]]=warna;
            for(j=0;j<n;j++){
                  if(m[sort[i]][sort[j]]==0&&colored[s
                  ort[j]]==0){
                  for(k=0;k<n;k++){
                        if(save[k]!=-1){
            if(m[save[k]][sort[j]]!=0){
                                          eror=1;
                              }
                        }
                  }
                  if(eror==0){
                        colored[sort[j]]=warna;
                        save[j]=sort[j];
                        }
                  eror=0;
                  }
            }
      }
}
```

7. Terakhir adalah menampilkan hasil akhir yang berupa warna setiap vertek dan juga jumlah bilangan kromatik (warna yang di gunakan).

```
for(i=0;i<n;i++){
      printf(" warna vertek %c = warna
      %d\n",i+65,colored[i]);
}
      printf("\n\ntotal bilangan kromatik =
%d",warna);
```

8. Proses Pewarnaan Sudah Selesai.

1.2 PENJELASAN FUNGSI ALGORITMA DJIKSTRA

1.2.1. SYNTAX FUNGSI DJIKSTRA

```
void djikstra(){
    printf("\nMenu 2 ( Algoritma Graph )\n\n");
      printf("-- ALGORITMA DJIKSTRA --\n\n");
      int n,count,start,min,before,i,j;
      printf("Masukkan jumlah Vertek = "); scanf("%d",&n);
      int m[n][n],visited[n],jarak[n],prev[n];
      //input
      printf("\nMasukkan matriks hubungan : \n\n  ");
      for(i=0;i<n;i++){
            printf("%c ",i+65);
      }
      printf("\n");
      for(i=0;i<n;i++){
            printf("%c ",i+65);
            for(j=0;j<n;j++){
                  scanf("%d",&m[i][j]);
            }
            for(j=0;j<n;j++){
      if(m[i][j]==0){
            m[i][j]=999;
                  }
            }
      }
//SYNTAX DIBAWAH INI ADALAH LANGKAH ALGORITMA DJIKSTRA DALAM
INISIALISASI VARIABEL & PEMROSESAN PROGRAM
//1. Inisialisasi jarak setiap vertek dari jarak awal = 999
//2. Dalam tahap ini semua vertek belum dikunjungi
    for(i=0;i<n;i++){
      visited[i]=0;
      jarak[i]=999;
      }
//3. Input vertek awal
      printf("\nMasukkan vertek awal = ");scanf("%d",&start);
      start-=1;
//4. Mengubah jarak vertek awal menjadi 0
      jarak[start]=0;
//5. Mencari sorthest path dan dilakukan sebanyak jumlah vertek
      while(count<n){

            min=999;
//6. Mencari jarak terkecil (lebih kecil dari min) yang ada pada
vertek belum terkunjungi
            for(i=0;i<n;i++){
                  if(jarak[i]<min && visited[i]==0){
    //maka jarak min menjadi jarak dari setiap vertek
//7. variabel Before akan menyimpan vertek yang memiliki jarak
minimum
                        min=jarak[i];
                        before=i;
                  }
            }
            visited[before]=1;
            //before diubah menjadi sudah terkunjungi
//8. Mengubah jarak vertek yang terhubung dengan vertek before
            for(i=0;i<n;i++){
```

```
        //jika vertek tersebut belum terkunjungi dan jika
            //jarak vertek before(min) ditambah dengan jarak
vertek before ke vertek i
                //maka jarak vertek tersebut akan diubah menjadi
                    //jarak vertek before ditambah dengan jarak
vertek before ke vertek i
                 if(visited[i]==0){
                       if(min+m[before][i]<jarak[i]){
                             jarak[i]=min+m[before][i];
                             prev[i]=before; //9. Prev menyimpan
vertek yang menuju vertek i yaitu vertek before
                       }
                 }
            }
            count++;
      }
//SYNTAX DIBAWAH INI ADALAH LANGKAH ALGORITMA DJIKSTRA DALAM
MENAMPILKAN HASIL AKHIR
      for(i=0;i<n;i++){
            if(i!=start){
        printf("\nJarak dari titik %c ke %c adalah = %d\nDengan
jalur %c ke %c adalah : ",start+65,i+65,jarak[i],start+65,i+65);
        j=i;
        printf("%c <-- ",j+65);
            while(j!=start){
                j=prev[j];
                printf("%c ",j+65);
                if(j!=start){
                    printf("<-- ");
                }
            } printf("\n");
            }
      }
      printf("\n");
}
//<END, PROGRAM ALGORITMA DJIKSTRA SUDAH SELESAI>
```

1.2.2. TAHAPAN ALGORITMA DARI FUNGSI DJIKSTRA

1. Pada fungsi ini kita memerlukan variabel untuk menyimpan jumlah vertek (n) , menghitung jumlah pengecekan (count) , vertek awal (start) , menyimpan vertek sementara (before) , dan menyimpan nilai minimum (min).

2. Juga memerlukan array 2 dimensi berukuran n untuk menyimpan matriks ketetanggaan (m[n][n]) , array yang menyimpan apakah vertek sudah di kunjungi berukuran n (visited[n]) , jarak setiap vertek yang berukuran n (jarak[n]) dan juga array menyimpan vertek sebelumnya dari setiap vertek yang berukuran n (prev[n]).

3. Melakukan Input yang berupa matriks ketetanggaan.

```
printf("Masukan matriks hubungan : \n  ");
for(i=0;i<n;i++){
      printf("%c\t",i+65);
}
printf("\n");
for(i=0;i<n;i++){
      printf("%c|",i+65);
      for(j=0;j<n;j++){
            scanf("%d",&m[i][j]);
      }
      for(j=0;j<n;j++){
if(m[i][j]==0){
      m[i][j]=999;
            }
      }
}
```

4. Setelah melakukan input matriks ketetanggaan , kita perlu set jarak untuk semua vertek adalah 999 dan semua vertek belum terkunjungi. Dan memilih titik awal atau start. Setelah menentukan titik awal / start maka jarak dari vertek awal adalah 0 , karena jarak dari vertek awal ke vertek awal adalah 0.

```
for(i=0;i<n;i++){
      visited[i]=0;
      jarak[i]=999;
}
//input vertek awal
printf("Masukan vertek awal = ");scanf("%d",&start);
start-=1;
//mengubah jarak vertek awal menjadi 0
jarak[start]=0;
```

5. Melakukan proses algoritma djikstra

   - ✓ Mencari vertek dengan jarak terkecil yang ada dan juga belum terkunjungi .

```
for(i=0;i<n;i++){
      min=999;
      if(jarak[i]<min && visited[i]==0){
            min=jarak[i];
            before=i;
      }
}
visited[before]=1;
```

   Program akan mencari nilai terkecil dari jarak vertek. Dan vertek dengan jarak terkecil akan disimpan pada variabel before, dan visited[before]=1 , berarti vertek tersebut statusnya dirubah menjadi sudah terkunjungi.

   - ✓ Mengisi jarak setiap vertek yang terhubung dengan vertek before.

```
for(i=0;i<n;i++){
      if(visited[i]==0){
            if(min+m[before][i]<jarak[i]){
                  jarak[i] = min +
            m[before][i];
            prev[i]=before;
            }
      }
}
```

   Program akan mengecek setiap vertek , jika vertek tersebut belum terkunjungi (visited[i]==0) program akan menghitung jarak dari vertek before ke vertek I dan menjumlahkannya dengan nilai min (jarak[before]) , jika lebih kecil dari jarak vertek i yang sekarang maka jarak vertek I akan dirubah menjadi hasil penjumlahan tersebut. Dan itu berarti vertek sebelumnya untuk menuju vertek I dengan jarak tersebut adalah vertek before.

   - ✓ Jika proses tersebut selesai maka jumlah pengecekan + 1 , (count++)
   - ✓ Proses ini dilakukan selama nilai count masih lebih kecil dari jumlah vertek.

6. Mencetak hasil dari algoritma djikstra.

```
for(i=0;i<n;i++){
        if(i!=start){
              printf("\nJarak dari titik %c ke %c adalah
              = %d\nDengan jalur %c ke %c adalah
              :",start+65,i+65,jarak[i],start+65,i+65);
              j=i;
              printf("%c <-- ",j+65);
                    while(j!=start){
                          j=prev[j];
                          printf("%c ",j+65);
                          if(j!=start){
                                printf("<-- ");
                          }
                    }
              }
        }
   }
```

Mencetak hasil dari proses perhitungan terhadap semua vertek yang ada. Jika I bukan titik awal , maka akan dicetak hasil dari jarak titik start ke veertek i. dan akan menampilkan rute dari titik start ke vertek i. Dimana mencetak rute tersebut menggunakan array prev , dan j akan menyimpan nilai dari prev[j] , yang artinya nilai j akan bergerak mundur dari vertek I , mengikuti jalannya yang ada pada array prev hingga nilai j bukan start.

1.3. PENJELASAN FUNGSI FLOYD WARSHALL

1.3.1. SYNTAX FUNGSI FLOYD

```
void floyd(){
    printf("\nMenu 3 ( Algoritma Graph )\n\n");
      printf("-- ALGORITMA FLOYD WARSHALL --\n");
    int n,k,i,j;
//SYNTAX DIBAWAH INI ADALAH LANGKAH ALGORITMA FLOYD WARSHALL DALAM
INISIALISASI VARIABEL & PEMROSESAN PROGRAM
//1. Input Vertek dengan Variabel n
      printf("\nMasukkan jumlah vertek : ");scanf("%d",&n);
      int m[n][n],before[n][n];
      printf("\nMasukkan matriks bobot = \n\n  ");
//2. Input pertama merupakan iterasi awal / iterasi 0
      for(i=0;i<n;i++){
            printf("%c ",i+65);
      }
      printf("\n");
      for(i=0;i<n;i++){
            printf("%c ",i+65);
            for(j=0;j<n;j++){
                  scanf("%d",&m[i][j]);
            }
            for(j=0;j<n;j++){
                  if(m[i][j]==0 && i!=j){
                        m[i][j]=999;
                  }
```

```
            }
      }
//3. Akan dilakukan Pengecekan perulangan iterasi, baris, dan
kolom sebanyak jumlah vertek
    //k = perulangan iterasi
    //i = perulangan baris
    //j = perulangan kolom
    for(k=0;k<n;k++){
//4. Menyimpan nilai matriks hasil dari iterasi sebelumnya ke
matriks before
      for(i=0;i<n;i++){
            for(j=0;j<n;j++){
                  before[i][j]=m[i][j];
                  }
            }
      printf("\n\nD(%d) [iterasi ke-%d] :",k+1,k+1);
      for(i=0;i<n;i++){
            printf("\n");
            for(j=0;j<n;j++){
//5. Kondisi - Kondisi
    //jika bobot dari vertek i ke j pada iterasi sebelumnya lebih
besar dari
    //penjumlahan bobot vertek i ke vertek k (iterasi ke - k) dan
bobot vertek k ke vertek j pada iterasi sebelumnya
    //maka nilai dari bobot vertek i ke j akan setara dengan
penjumlahan bobot vertek i ke vertek k
    //dan bobot vertek k ke vertek j
                  if(before[i][j] > before[i][k]+before[k][j] ){
                        m[i][j]=before[i][k]+before[k][j];
                        }
                        if(m[i][j]==999){
                              printf("~\t");
                        }
                        else{
                              printf("%d\t",m[i][j]);
                        }
                  }
            }
      }
//6. Mencetak hasil akhir
      printf("\n\nShortest Path (Algoritma Floyd Warshall) : ");
      for(i=0;i<n;i++){
            printf("\n");
            for(j=0;j<n;j++){
                  printf("%d\t",m[i][j]);
            }
      }
      printf("\n\n");
}
//<END, PROGRAM ALGORITMA FLOYD WARSHALL SUDAH SELESAI>
```

1.3.2. TAHAPAN ALGORITMA DARI FUNGSI FLOYD

1. Pada fungsi ini kita memerlukan variabel yang menyimpan jumlah vertek (n). Kita juga memerlukan array 2 dimensi berukuran n x n untuk menyimpan matriks ketetanggaan(m[n][n]), dan array 2 dimensi berukuran n x n untuk menyimpan matriks iterasi sebelumnya untuk setiap iterasi.

2. Melakukan input yaitu jumlah vertek , dan input matriks ketetanggaan.

```
printf("Masukan jumlah vertek : ");scanf("%d",&n);
int m[n][n],before[n][n];
printf("Masukan matriks bobot = \n  ");
//input pertama merupakan iterasi awal / iterasi 0
for(i=0;i<n;i++){
      printf("%c\t",i+65);
}
printf("\n");
for(i=0;i<n;i++){
      printf("%c|",i+65);
      for(j=0;j<n;j++){
            scanf("%d",&m[i][j]);
      }
      for(j=0;j<n;j++){
            if(m[i][j]==0 && i!=j){
                  m[i][j]=999;
            }
      }
}
```

   Matriks ketetanggaan yang diinputkan pertama merupakan shortest path iterasi ke -0 atau D(0). Jika m[i][j] = 0 dan I != j maka m[i][j] diubah menjadi 999 atau infinity , karena pada iterasi ke-0 atau matriks awal jarak antara vertex I ke j tidak di ketahui.

3. Melakukan Algoritma Floyd warshal

   - ✓ Pada algoritma ini kita menggunakan 3 variabel counter untuk melakukan perulangan yaitu k untuk perulangan iterasi , dimana perulangan k dimulai dari k=0 selama k < n, variabel i sebagai perulangan baris dimana perulangan I dimulai dari I = 0 selama I < n , dan j sebagai variabel counter untuk perulangan kolom dimana perulangan j dimulai dari j = 0 selama j < n.
   - ✓ Pertama kita harus menyimpan keadaan matriks yang sekarang ke dalam matriks before sebagai indicator dari matriks iterasi sebelumnya.
   - ✓ Jika jarak dari vertek I ke j pada matriks before lebih besar dari hasil penjumlahan jarak vertek I ke vertek k dan vertek k ke vertek j pada matriks before , maka m[i][j] akan diubah menjadi hasil dari penjumlahan tersebut, karena m[i][j] merupakan matriks hasil perhitungan Floyd warshal pada iterasi ke – k.
   - ✓ Proses ini akan terus dilakukan sebanyak jumlah vertek.

```
      for(k=0;k<n;k++){
             for(i=0;i<n;i++){
                   for(j=0;j<n;j++){
                         before[i][j]=m[i][j];
                   }
             }
             printf("\n\nD(%d) [iterasi ke-%d]
:",k+1,k+1);
             for(i=0;i<n;i++){
             printf("\n");
             for(j=0;j<n;j++){
             if(before[i][j] >
before[i][k]+before[k][j] ){

      m[i][j]=before[i][k]+before[k][j];
                         }
                         if(m[i][j]==999){
                               printf("~\t");
                         }
                         else{
                               printf("%d\t",m[i][j]);
                         }
                   }
             }
      }
```

4. Mencetak hasil akhir (iterasi ke-n) dari m[i][j] karena hasil akhir dari algoritma ini adalah berupa matriks.

```
printf("\n\nShortest path(algoritma floyd warshal) = ");
for(i=0;i<n;i++){
      printf("\n");
      for(j=0;j<n;j++){
            printf("%d\t",m[i][j]);
      }
}
```

5. Proses Algoritma Floyd Warshall Sudah Selesai.

1.4. PENJELASAN FUNGSI ALGORITMA BELLMAN FORD

1.4.1. SYNTAX FUNGSI BELLMAN

```
void bellman(){
    printf("\nMenu 4 ( Algoritma Graph )\n\n");
      printf("-- ALGORITMA BELLMAN FORD --\n\n");
      int n,count=0,start,i,j;
//SYNTAX DIBAWAH INI ADALAH LANGKAH ALGORITMA BELLMAN FORD DALAM
INISIALISASI VARIABEL & PEMROSESAN PROGRAM
//1. Input Vertek dan Matriks Bobot
      printf("Masukkan jumlah vertek : ");scanf("%d",&n);
      int m[n][n],d[n],pred[n],negative[n];
      printf("Masukkan matriks bobot = \n\n  ");
      for(i=0;i<n;i++){
            printf("%c ",i+65);
      }
      printf("\n");
      for(i=0;i<n;i++){
            printf("%c ",i+65);
            for(j=0;j<n;j++){
                  scanf("%d",&m[i][j]);
            }
            for(j=0;j<n;j++){
      if(m[i][j]==0){
            m[i][j]=999;
                  }
            }
      }
//2. Menginisialisasi jarak setiap vertek adalah 999, dan vertek
pembuat negatif tidak ada
    for(i=0;i<n;i++){
      d[i]=999;
      negative[i]=-1;
      }
//3. Menginput vertek awal dan membuat jarak dari titik awal
menjadi 0
    printf("\nMasukkan vertek awal : ");scanf("%d",&start);
    d[start]=0;
//4. Syntax dibawah ini akan menghitung sebanyak jumlah vertek-1
    while(count<n-1){
      for(i=0;i<n;i++){
            for(j=0;j<n;j++){
//5. Kondisi - Kondisi
    //jika jarak titik awal ke vertek i dijumlahkan dengan bobot(
jarak) vertek i ke vertek j
    //yang nilainya lebih kecil dari jarak titik awal ke vertek j.
    //maka jarak titik awal ke vertek j akan diubah menjadi
penjumlahan dari jarak titik awal ke vertek i
    //dengan bobot(jarak) vertek i ke vertek j
                  if(d[i]+m[i][j] < d[j]){
                        d[j]= d[i]+m[i][j];
pred[j]=i;
//variabel array predecessor diatas akan menyimpan vertek
sebelumnya yang menuju vertek i
                        }
                  }
            }
      count++;
```

```
      }
//6. Syntax dibawah ini adalah langkah pengecekan negatif cycle
pada suatu vertek
      for(i=0;i<n;i++){
            for(j=0;j<n;j++){
//7. Kondisi - Kondisi
    //jika jarak titik awal ke vertek j berubah lagi pada
pengecekan ini,
    //maka vertek tersebut merupakan vertek terhubung dengan
negatif cycle
                  if(d[i]+m[i][j] < d[j]){
d[j]= (-999); //Syntax disamping adalah jarak dari vertek awal ke
vertek j
              //yang kemudian akan diubah menjadi -999
              //karena akan selalu berkurang setiap dilakukan
pengecekan
negative[j]=i;//Syntax disamping akan menyimpan vertek
              //yang berhubungan dengan vertek j
              //yang membuat jaraknya menjadi negatif cycle
                  }
            }
      }
//8. Menampilkan hasil akhir
      for(i=0;i<n;i++){
            if(i!=start){
printf("\nJarak dari %c ke vertek %c adalah = ",start+65,i+65);
                  if(d[i]==(-999)){
printf("-%d dengan vertek pembuat negative %c
",d[i],negative[i]+65);
                  }
                  else{
                        printf("%d",d[i]);
                  }
            }
      }
printf("\n\n");
}
//<END, PROGRAM ALGORITMA BELLMAN FORD SUDAH SELESAI>
```

1.4.2. TAHAPAN ALGORITMA DARI FUNGSI BELLMAN

1. Pada fungsi ini kita memerlukan variabel untuk menyimpan jumlah vertek (n) , variabel untuk menghitung jumlah proses atau pengecekan yang dilakukan (count) , dan juga variabel yang menyimpan vertek awal (start)

2. Membuat array 2 dimensi yang menyimpan matriks hubungan / ketetanggaan berukuran n x n (m[n][n]) , array 1 dimensi yang berukuran n , yaitu d[n] untuk menyimpan jarak dari titik awal ke semua vertek , pred[n] yang menyimpan vertek sebelumnya untuk menuju vertek tersebut dari titik start , dan juga negative[n] yang akan menyimpan vertek penyebab negative cycle pada suatu vertek.

3. Melakukan input berupa matriks ketetanggaan

```
printf("Masukan matriks bobot = \n  ");
//input
for(i=0;i<n;i++){
      printf("%c ",i+65);
}
printf("\n");
for(i=0;i<n;i++){
      printf("%c ",i+65);
      for(j=0;j<n;j++){
            scanf("%d",&m[i][j]);
      }
      for(j=0;j<n;j++){
if(m[i][j]==0){
      m[i][j]=999;
            }
      }
}
```

4. Setelah melakukan input matriks, kita perlu menginisialisasi jarak untuk setiap vertek adalah 999 dan juga setiap vertek tidak memiliki vertek pembuat negative. Setelah itu kita memasukan vertek awal. Jarak ke vertek awal akan di set menjadi 0 karena , jarak dari vertek awal ke vertek awal adalah 0.

```
for(i=0;i<n;i++){
      d[i]=999;
      negative[i]=-1;
}
printf("\nMasukan vertek awal : ");scanf("%d",&start);
d[start]=0;
```

5. Melakukan proses algoritma bellmand ford

   - ✓ Mengisi jarak untuk setiap vertek. Dimulai dari vertek 0 akan di cek untuk setiap vertek apakah jika jarak dari vertek 0 + bobot vertek I ke vertek j memiliki nilai lebih kecil dari pada jarak untuk titik start ke titik j saat ini. Jika lebih kecil maka jarak dari vertek start ke vertek j akan diubah menjadi hasil penjumlahan tersebut dan juga untuk menuju vertek j akan melalui vertek I untuk mencapai jarak tersebut , sehingga vertek I disimpan pada pred[j], (pred adalah predecessor yaitu vertek sebelumnya/pendahulu untuk mencapai vertek tersebut). Pengecekan dilakukan hingaa vertek terakhir. Jika proses selesai maka jumlah pengecekan bertambah satu(count++). Proses ini dilakukan selama jumlah pengecekan kurang dari jumlah vertek -1.

```
while(count<n-1){
for(i=0;i<n;i++){
      for(j=0;j<n;j++){
            if(d[i]+m[i][j] < d[j]){
                  d[j]= d[i]+m[i][j];
                  pred[j]=i;
            }
      }
  }
  count++;
}
```

   - ✓ Setelah melakukan proses tersebut , akan dilakukan pengecekan 1 kali lagi untuk mengetahui apakah jarak dari titik start ke vertek j terus berubah ubah atau tidak. Jika jarak untuk ke vertek j berubah kembali pada proses ini maka jarak dari vertek start ke vertek j adalah -999 , karena jarak untuk mencapai vertek tersebut akan terus berkurang semakin dilakukan pengecekan. Jika terjadi hal tersebut berarti graph ini mengandung negative cycle yang disebabkan oleh bobot dari vertek I ke vertek j yang menyebabkan terjadinya perubahan tersebut. Sehingga negative[j] akan menyimpan vertek I yang merupakan vertek yang berhubungan dengan vertek j yang edge nya menyebabkan negative cycle.

```
for(i=0;i<n;i++){
      for(j=0;j<n;j++){
            if(d[i]+m[i][j] < d[j]){
                  d[j]= (-999);
                  negative[j]=i;
            }
      }
}
```

6. Mencetak Hasil akhir

```
for(i=0;i<n;i++){
      if(i!=start){
            printf("\nJarak dari %c ke vertek %c
            adalah = ",start+65,i+65);
            if(d[i]==(-999)){
                  printf("-%d dengan vertek pembuat
                  negative %c ",d[i],negative[i]+65);
            }
            else{
                  printf("%d",d[i]);
            }
      }
}
```

Program akan mencetak jarak dari titik start ke semua vertek lain kecuali start. Jika jarak ke suatu vertek adalah -999 program akan mencetak juga vertek pembuat negative untuk vertek tersebut.

7. Program Algoritma Bellman Ford Sudah Selesai.

BAB II
PENJELASAN ALGORITMA PADA TREE - MST

2.1. PENJELASAN FUNGSI KODE PRUFER

2.1.1. DEKONSTRUKSI PRUFER - ENCODE

2.1.1.1. SYNTAX FUNGSI ENCODE PRUF

```
void pruf_en(){
    printf("\nMenu 1 ( MST - Prufer )\n\n");
      printf("-- DEKONSTRUKSI PRUFER --\n\n");
      int i,j,k,vertek,a,b;
//SYNTAX DIBAWAH INI ADALAH LANGKAH MST ENKODING KODE PRUFER DALAM
INISIALISASI VARIABEL & PEMROSESAN PROGRAM
//1. Inisialisasi & Input
      printf("Masukkan jumlah vertek : ");scanf("%d",&vertek);
      int code[vertek-2]; //menyimpan code
      int M[vertek][vertek]; //menyimpan matriks bobot
      int drjt[vertek]; //menyimpan derajat
      //Syntax Dibawah ini adalah inisialisasi code kosong ,
derajat semua vertek 0 , dan matriks bobot kosong
      for(i=0;i<vertek;i++){
            code[i]=0;
            drjt[i]=0;
            for(j=0;j<vertek;j++){
                  M[i][j]=0;
            }
      }
      for(i=0;i<vertek-1;i++){
            printf("\nMasukkan edge = ");scanf("%d %d",&a,&b);
            M[(a-1)][(b-1)]=M[(b-1)][(a-1)]=1;
      }
//2. Syntax dibawah ini adalah langkah menghitung derajat
      for(i=0;i<vertek;i++){
            for(j=0;j<vertek;j++){
                  drjt[i]+=M[i][j];
            }
            printf("\nDerajat vertek %d = %d ",i+1,drjt[i]);
      }
//Syntax dibawah ini adalah langkah dalam mencari kode prufer
//3. kode prufer suatu tree berjumlah sebanyak jumlah vertek - 2
      for(k=0;k<vertek-2;k++){
//4. mencari vertek dengan angka terkecil dan derajat 1
//5. perulangan ini akan mendapatkan nilai vertek dengan angka
terkecil
//6. karena di mulai dari vertek awal dan jika sudah ketemu kode
untuk edge pada vertek tersebut
//7. maka perulangan akan dipaksa berhenti, dengan begitu akan
didapat vertek terkecil
            for(i=0;i<vertek;i++){
                  if(drjt[i]==1){
//8. Mencari vertek yang berhubungan dengan vertek i
                        for(j=0;j<vertek;j++){
                              if(M[i][j]==1){
```

```
                                code[k]=j+1;
                    //kode yang disimpan adalah vertek j
                                drjt[i]--;
                                drjt[j]--;
                                break;
                            }
                        }
                        M[i][code[k]]=0;
                        M[code[k]][i]=0;
                        break;
                    }
                }
            }
//9. Syntax Dibawah ini adalah langkah dalam mencetak hasil akhir
            printf("\n\nKode Prufer = ");
            for(i=0;i<vertek-2;i++){
                printf("%d ",code[i]);
            }
            printf("\n\n");
}
//<END, PROGRAM MST - DEKONSTRUKSI PRUFER SUDAH SELESAI>
```

2.1.1.2. TAHAPAN ALGORITMA DARI FUNGSI PRUFER - ENCODE

1. Pada fungsi ini kita memerlukan variabel untuk menyimpan jumlah vertek (vertek) , variabel untuk menyimpan vertek asal pada saat input (a) , dan vertek tujuan pada saat input (b).

```
int i,j,k,vertek,a,b;
```

2. Kita juga memerlukan array 2 dimensi berukuran vertek x vertek untuk menyimpan matriks ketetanggaan (M[vertek][vertek]) , array berukuran vertek-2 untuk menyimpan code , karena jumlah kode prufer adalah jumlah vertek-2, array untuk menyimpan derajat setiap vertek.

```
int code[vertek-2];
int M[vertek][vertek];
int drjt[vertek];
```

3. Menginisialisasi semua kode kosong , derajat setiap vertek 0 dan semua vertek tidak memiliki hubungan.

```
for(i=0;i<vertek;i++){
      code[i]=0;
      drjt[i]=0;
      for(j=0;j<vertek;j++){
            M[i][j]=0;
      }
}
```

4. Melakukan input vertek yang memiliki hubungan . Dan setelah melakukan input edge , program akan menghitung derajat dari setiap vertek dengan cara menambahkan derajat vertek I dengan bobot atau nilai dari vertek I ke semua vertek.

```
for(i=0;i<vertek-1;i++){
      printf("\nMasukan edge= ");scanf("%d %d",&a,&b);
      M[(a-1)][(b-1)]=M[(b-1)][(a-1)]=1;
}
for(i=0;i<vertek;i++){
      for(j=0;j<vertek;j++){
            drjt[i]+=M[i][j];
      }
      printf("\nDerajat vertek %d = %d ",i+1,drjt[i]);
}
```

5. Mencari kode prufer dari tree tersebut

- ✓ Mencari vertek dengan angka terkecil dan dengan deerajat 1. Program akan menemukan vertek dengan angka terkecil karena perulangan mencari vertek dengan derajat 1 dimulai dari vertek terkecil hingga vertek terakhir. Dan saat sudah ketemu vertek dengan derajat 1 dan di dapatkan kodenya , program akan memberhentikan perulangan I dan j, dengan begitu program dpt menemukan vertek terkecil
- ✓ Jika vertek tersebut memiliki derajat 1 maka akan di proses selanjutnya , program akan mencara vertek yang berhubungan dengan vertek I , dan program akan menyimpan vertek yang berhubungan dengan vertek I sebagai kode ke-k .
- ✓ Lalu derajat vertek I dan vertek yang berhubungan dengan vertek I akan di kurangi. Dan juga pada matriks ketetanggaan bobot mereka di set menjadi 0 .
- ✓ Proses dilakukan sebanyak jumlah vertek-2

```
for(k=0;k<vertek-2;k++){
      for(i=0;i<vertek;i++){
            if(drjt[i]==1){
                  for(j=0;j<vertek;j++){
                        if(M[i][j]==1){
                              code[k]=j+1;
                              drjt[i]--;
                              drjt[j]--;
                              break;
                        }
                  }
                  M[i][code[k]]=0;
                  M[code[k]][i]=0;
                  break;
            }
      }
}
```

6. Mencetak hasil akhir dari kode prufer.

```
printf("\nKode Prufer = ");
for(i=0;i<vertek-2;i++){
      printf("%d ",code[i]);
}
```

7. Program MST - Dekonstruksi Prufer Sudah Selesai.

2.1.2. REKONSTRUKSI PRUFER - DECODE

2.1.2.1. SYNTAX FUNGSI DECODE - PRUFER

```
void pruf_dec(){
      int jml_k,i,j,k,jml_v,temp;
    printf("\nMenu 2 ( MST - Prufer )\n\n");
      printf("-- REKONSTRUKSI PRUFER --\n\n");
//SYNTAX DIBAWAH INI ADALAH LANGKAH MST DEKODING KODE PRUFER DALAM
INISIALISASI VARIABEL & PEMROSESAN PROGRAM
//1. Inisialisasi Variabel & Input
      printf("Masukkan jumlah kode = ");scanf("%d",&jml_k);
      int kode[jml_k];
      printf("\nMasukkan kode = ");
      for(i=0;i<jml_k;i++){
            scanf("%d",&kode[i]);
      }
      jml_v=jml_k+2; //Syntax disamping menunjukkan jumlah vertek
sama dengan jumlah kode + 2
      int vertek[jml_v];
      int m[jml_v][jml_v];
      for(i=0;i<jml_v;i++){
vertek[i]=i+1;
//2. Mengisi array vertek dengan angka sesuai urutan dari 1 -
jumlah vertek
            for(j=0;j<jml_v;j++){
                  m[i][j]=0;
            }
      }
      printf("\nTree = ");
      for(i=0;i<jml_k;i++){
            for(j=0;j<jml_v;j++){
if(kode[i]!= vertek[j] && vertek[j]!=0 && kode[i]!=0 ){
                        for(k=0;k<jml_k;k++){
                              temp=0;
                              if(kode[k]!=vertek[j]){
                                    temp=1;
                              }
                              else{
                                    break;
                              }
                        }
                        if(temp==1){
printf("\n[%d] - [%d]",kode[i],vertek[j]);
m[kode[i]-1][vertek[j]-1]=m[vertek[j]-1][kode[i]-1] = 1;
                              kode[i]=0;
                              vertek[j]=0;
```

```
                        }
                  }
            }
      }
      for(i=0;i<jml_v;i++){
            if(vertek[i]!=0){
                  for(j=0;j<jml_v;j++){
                        if(vertek[i]!=vertek[j] && vertek[j]!=0 ){
printf("\n[%d] - [%d]",vertek[i],vertek[j]);
m[vertek[i]-1][vertek[j]-1]=m[vertek[j]-1][vertek[i]-1] = 1;
                              vertek[i]=0;
                              vertek[j]=0;
                        }
                  }
            }
      }
      printf("\n\nDengan matriks ketetanggaan  = \n\n  ");
      for(i=1;i<=jml_v;i++){
            printf("%d ",i);
      }
      for(i=0;i<jml_v;i++){
            printf("\n%d ",i+1);
            for(j=0;j<jml_v;j++){
                  printf("%d ",m[i][j]);
            }
      }
      printf("\n\n");
}
//<END, PROGRAM MST - REKONSTRUKSI PRUFER SUDAH SELESAI>
```

2.1.2.2. TAHAPAN ALGORITMA DARI FUNGSI PRUFER - DECODER

1. Pada fungsi ini kita memerlukan variabel untuk menyimpan jumlah kode (jml_k) , untuk menyimpan jumlah vertek (jm_v) dan variabel untuk menyimpan indicator apakah proses bisa dilanjutkan atau tidak (temp).

```
int jml_k,i,j,k,jml_v,temp;
```

2. Kita memerlukan array untuk menyimpan kode berukuran jml_k (kode[jml_k]) , array untuk menyimpan vertek berukuran jml_v (vertek[jml_v]) dan array untuk menyimpan matriks ketetanggaan.

```
int kode[jml_k];
int vertek[jml_v];
int m[jml_v][jml_v];
```

3. Menginisialisasi jumlah verteek yaitu jumlah kode + 2 , mengisi array vertek dengan nilai 1 sampai jumlah vertek , dan juga menginisialisasi semua vertek tidak memiliki hubungan (m[i][j]=0)

```
jml_v=jml_k+2;
for(i=0;i<jml_v;i++){
      vertek[i]=i+1;
      // mengisi array vertek dengan angka sesuai
      urutan //dari 1 - jumlah vertek
```

```
            for(j=0;j<jml_v;j++){
                  m[i][j]=0;
            }
      }
```

4. Melakukan proses pembentukan tree dari kode yang sudah di input
   - ✓ Melakukan pengecekan untuk setiap kode pada array kode ke setiap vertek yang ada di array vertek. Dimulai dari kode ke-0
   - ✓ Jika kode ke-I tidak sama dengan vertek ke-j dan vertek ke-j dan kode ke-i tidak 0 atau kosong , maka proses dilanjutkan. Karena untuk membentuk tree , kode tidak boleh sama dengan vertek yang di cek.
   - ✓ Selanjutnya , kita mengecek apakah vertek[j] terdapat pada array kode dengan cara membandingkan vertek[j] dengan semua array kode yang tidak kosong. Jika vertek[j] ada pada array kode , maka nilai temp tetap 0 dan pengecekan langsung di paksa berhenti untuk melanjutkan ke pengecekan selanjutnya. Jika tidak maka nilai temp adalah 1 , sehingga proses bisa dilanjutkan , jika tidak temp akan tetap 0.
   - ✓ Jika temp =1 , maka program akan mencetak edge yaitu vertek kode[i] dengan vertek [j] . Selanjutnya program akan mengisi matriks ketetanggan dan membuat vertek kode[i] dengan vertek[j] menjadi berhubungan / bernilai i.
   - ✓ Nilai dari kode[i] dan vertek[j] di set menjadi 0 agar tidak di cek kembali.

```
printf("Tree = ");
   for(i=0;i<jml_k;i++){
         for(j=0;j<jml_v;j++){
               if(kode[i]!= vertek[j] && vertek[j]!=0 &&
               kode[i]!=0 ){
                     for(k=0;k<jml_k;k++){
                           temp=0;
                           if(kode[k]!=vertek[j]){
                                 temp=1;
                           }
                           else{
                                 break;
                           }
                     }
                     if(temp==1){
                           printf("\n[%d] -
                           [%d]",kode[i],vertek[j]);
                           m[kode[i]-1][vertek[j]-
                           1]=m[vertek[j]-1][kode[i]-1] = 1;
                           kode[i]=0;
                           vertek[j]=0;

                     }
               }
         }
   }
```

- ✓ Setelah melakukan pengecekan terhadap semua kode , kita akan melakukan hal yang sama terhadap vertek yang masih tersisa pada array vertek.

```
for(i=0;i<jml_v;i++){
            if(vertek[i]!=0){
                  for(j=0;j<jml_v;j++){
                        if(vertek[i]!=vertek[j] &&
vertek[j]!=0 ){
                              printf("\n[%d] -
                              [%d]",vertek[i],vertek[j
                              ]);
                              m[vertek[i]-
                              1][vertek[j]-
                              1]=m[vertek[j]-
                              1][vertek[i]-1] = 1;
                              vertek[i]=0;
                              vertek[j]=0;
                        }
                  }
            }
      }
```

5. Mencetak matriks ketetanggaan dari tree tersebut.

```
printf("\nDengan matriks ketetanggaan  =\n  ");
for(i=1;i<=jml_v;i++){
      printf("%d",i);
}
for(i=0;i<jml_v;i++){
      printf("\n%d|",i+1);
      for(j=0;j<jml_v;j++){
            printf("%d ",m[i][j]);
      }
}
```

6. Program MST - Rekonstruksi Prufer Sudah Selesai.

2.2. PENJELASAN FUNGSI ALGORITMA KRUSKAL

2.2.1. SYNTAX FUNGSI KRUSKAL

```
void kruskal(){

    printf("\nMenu 6 ( Minimum Spanning Tree - MST )\n\n");
      printf("-- ALGORITMA KRUSKAL --\n\n");
//SYNTAX DIBAWAH INI ADALAH LANGKAH MST - ALGORITMA KRUSKAL DALAM
INISIALISASI VARIABEL & PEMROSESAN PROGRAM
//1. Input akan disimpan dalam bentuk struct yang menyimpan vertek
terhubung (edge) dan bobot dari edge tersebut
      struct data{
            int bobot[1];
            int edge[2];
      };
```

```
      int total=0;
      int i,j,vertek,edge,temp,cek,t,k=0;
//2. Inisialisasi Variabel & Input
      printf("Masukkan jumlah vertek : ");scanf("%d",&vertek);
      printf("Masukkan jumlah edge : ");scanf("%d",&edge);
      int graph[vertek][vertek];
      struct data hub[edge];
      int used[edge];

//3. Inisialisasi nilai awal bobot semua edge infinity
      for(i=0;i<vertek;i++){
            for(j=0;j<vertek;j++){
                  graph[i][j]=999;
            }
      }
//4. Mengosongkan array used
      for(i=0;i<edge;i++){
            used[i]=0;
      }
//5. Input bobot
      printf("\n");
      for(i=0;i<edge;i++){
printf("Masukkan edge :
");scanf("%d%d",&hub[i].edge[1],&hub[i].edge[2]);
printf("Bobot : ");scanf("%d",&hub[i].bobot[1]);
graph[hub[i].edge[1]-1][hub[i].edge[2]-1]=hub[i].bobot[1];
graph[hub[i].edge[2]-1][hub[i].edge[1]-1]=hub[i].bobot[1];
            printf("\n");
      }
//6. Cetak matriks - matriks ketetanggaan
      printf("Matriks ketetanggaan : \n\n");
      for(i=0;i<vertek;i++){
            for(j=0;j<vertek;j++){
                  if(graph[i][j]!=999){
                        printf("%d ",graph[i][j]);
                  }
                  else{
                        printf("~ ");
                  }
            }
            printf("\n");
      }
//7. Sorting berdasarkan bobot dari terkecil hingga terbesar
      for (i=0;i<edge-1;i++){
      for (j=0;j<edge-i-1;j++){
            if (hub[j+1].bobot[1] < hub[j].bobot[1]){
                temp = hub[j].edge[1];
                hub[j].edge[1]=hub[j+1].edge[1];
                hub[j+1].edge[1]=temp;
                temp = hub[j].edge[2];
                hub[j].edge[2]=hub[j+1].edge[2];
                hub[j+1].edge[2]=temp;
                temp = hub[j].bobot[1];
                hub[j].bobot[1] = hub[j+1].bobot[1];
                hub[j+1].bobot[1] = temp;
            }
        }
      }
```

```
//8. Syntax Dibawah menunjukkan hasil sorting
      printf("\n*============================*\n");
      printf("        hasil sorting        \n\n");
      for(i=0;i<edge;i++){
            printf("  edge %d - %d <=> bobot = %d
\n",hub[i].edge[1],hub[i].edge[2],hub[i].bobot[1]);
      }
        printf("*============================*\n");

//9. Syntax Dibawah menunjukkan minimum spanning tree
      printf("\n*============================*\n");
      printf("    Minimum Spanning Tree    \n\n");
      int from,to;
    //from menyimpan vertek awal
      //to menyimpan vertek tujuan
      for(i=0;i<edge;i++){
            from=hub[i].edge[1];
            to=hub[i].edge[2];
//10. Syntax Dibawah akan mengecek apakah edge tersebut membentuk
cycle atau tidak
      //dengan cara mencari union graph tersebut
      //jika from dan to merupakan union(gabungan) graph yang sama
      //maka mereka akan membentuk cycle
      //dan jika akhir dari union from dan to sama maka mereka
membentuk cycle
      //dengan kata lain jika from sama dengan to
      //maka vertek awal bertemu kembali dengan dirinya sendiri
dan akan membentuk cycle
            while(used[from]){
                  from=used[from];
            }
            while(used[to]){
                  to=used[to];
            }
            if(from!=to){
        printf("    edge %d - %d bobot = %d
\n",hub[i].edge[1],hub[i].edge[2],hub[i].bobot[1]);
                  total+=hub[i].bobot[1];
//11. Syntax Dibawah adalah langkah menambahkan total bobot
                  used[from]=to;
        //syntax diatas akan menyimpan vertek terakhir pada
union(gabungan)
        //yang dibuat oleh vertek from
            }

      }
        printf("*============================*\n");
//12. Mencetak bobot total
      printf("\nbobot total : %d\n\n",total);
}
//<END, PROGRAM MST - ALGORITMA KRUSKAL SUDAH SELESAI>
```

2.2.2. TAHAPAN ALGORITMA DARI FUNGSI KRUSKAL

1. Pada Fungsi ini Kita memerlukan variabel yang menyimpan jumlah vertek (vertek) , menyimpan jumlah edge (edge) , variabel untuk menyimpan suatu nilai sementara. Variabel total untuk menyimpan total bobot dari MST , from untuk menyimpan vertek asal pada saat pengecekan , to untuk menyimpan vertek tujuan pada saat pengecekan.

```
int total=0;
int i,j,vertek,edge,temp;
int from,to;
```

2. Pada fungsi ini kita menggunakan struct yang menyimpan edge(vertek yang berhubungan) serta bobot dari edge tersebut sejumlah banyak edge (struct data hub[edge]). Array 2 dimensi berukuran vertek x vertek untuk menyimpan matriks (graph[vertek][vertek]) dan array yang menyimpan union atau gabungan dari dua buah vertek (menyimpan ujung dari gabungan vertek / atau bisa dikatakan vertek ujung dari sutau trail yang titik awalnya vertek tertentu) yang berukuran sebanyak jumlah edge.

```
struct data{
      int bobot[1];
      int edge[2];
};
int graph[vertek][vertek];
struct data hub[edge];
int used[edge];
```

3. Menginisialisasi matriks bobot dengan 999 , dan array used menjadi 0 atau kosong.

```
for(i=0;i<vertek;i++){
      for(j=0;j<vertek;j++){
            graph[i][j]=999;
      }
}
for(i=0;i<edge;i++){
      used[i]=0;
}
```

4. Melakukan input edge dan bobot dari edge. Vertek asal dalam edge ke -i akan disimpan pada hub[i].edge[1] , dan vertek tujuan pada edge ke I akan disimpan pada hub[i].edge[2] , serta bobotnya disimpan pada hub[i].bobot[1]. Input vertek tersebut juga akan kita masukan ke dalam graph.

```
for(i=0;i<edge;i++){
      printf("Masukan edge :
      ");scanf("%d%d",&hub[i].edge[1],&hub[i].edge[2])
      ;
      printf("Bobot : ");scanf("%d",&hub[i].bobot[1]);
      graph[hub[i].edge[1]-1][hub[i].edge[2]-
```

```
            1]=hub[i].bobot[1];
            graph[hub[i].edge[2]-1][hub[i].edge[1]-
            1]=hub[i].bobot[1];
            printf("\n");
      }
```

5. Selanjutnya program akan mencetak matriks ketetanggaan

```
      printf("Matriks ketetanggaan:\n");
      for(i=0;i<vertek;i++){
            for(j=0;j<vertek;j++){
                  if(graph[i][j]!=999){
                        printf("%d ",graph[i][j]);
                  }
                  else{
                        printf("~ ");
                  }
            }
            printf("\n");
      }
```

6. Melakukan sorting berdasarkan bobot dari setiap edge. Jika bobot edge sselanjutnya pada array lebih besar dari bobot edge sekarang , maka semua data pada struct hub[j] akan ditukar dengan hub[j+1]. Dan selanjutnya akan mencetak hasil sortingan tersebut.

```
      for (i=0;i<edge-1;i++){
            for (j=0;j<edge-i-1;j++){
                  if (hub[j+1].bobot[1] < hub[j].bobot[1]){
                temp = hub[j].edge[1];
            hub[j].edge[1]=hub[j+1].edge[1];
      hub[j+1].edge[1]=temp;
      temp = hub[j].edge[2];
hub[j].edge[2]=hub[j+1].edge[2];
hub[j+1].edge[2]=temp;
temp = hub[j].bobot[1];
hub[j].bobot[1] = hub[j+1].bobot[1];
hub[j+1].bobot[1] = temp;
            }
            }
      }

      printf("\nhasil sorting: \n");
      for(i=0;i<edge;i++){
            printf("edge %d-
            %d|bobot=%d\n",hub[i].edge[1],hub[i].edge[2],hub
            [i].bobot[1]);
      }
```

7. Melakukan proses mencari MST dengan algoritma Kruskal

   - ✓ Karena edge sudah diurutkan dari yang terkecil ke yang terbesar , maka kita akan mengecek dari edge hub[0] sampai hub[edge].
   - ✓ Kita tidak perlu mencari jarak minimum lagi karena sudah di urutkan , jadi yang perlu di cek pada program ini adalah apakah edge tersebut akan membentuk cycle atau tidak.
   - ✓ From akan menyimpan vertek asal pada hub[i] , dan to akan menyimpan vertek tujuan dari hub[i].
   - ✓ Selanjutnya kita akan mencari union dari vertek from, dengan cara menggunakan vertek used , selama used[from] tidak kosong atau 0 maka from akan menyimpan nilai used[from] , dengan kata lain posisi vertek from akan pindah ke vertek used[from] untuk mencari ujung dari union atau trailnya. Hal yang sama dilakukan pada vertek to.
   - ✓ Selanjutnya kita cek , apakah from sama dengan to atau tidak , kalau mereka sama itu berarti from dan to merupakan union graph yang sama , atau bisa dikatakan trail berawal dan berakhir pada vertek yang sama , dengan begitu edge tersebut akan membentuk cycle. Jika from tidak sama dengan to , maka edge tersebut tidak membentuk cycle.
   - ✓ Jika tidak membentuk cycle , maka edge tersebut akan di print Bersama dengan bobotnya. Lalu total MST akan ditambahkan dengan bobot edge tersebut. Dan ujung dari vertek from adalah to (used[from] = to).

```
printf("\n======Minimum Spanning Tree======\n");
for(i=0;i<edge;i++){
      from=hub[i].edge[1];
      to=hub[i].edge[2];
      while(used[from]){
            from=used[from];
      }
      while(used[to]){
            to=used[to];
      }
      if(from!=to){
            printf("edge %d-%d
            bobot=%d\n",hub[i].edge[1],hub[i].edge[2],hub[i]
            .bobot[1]);
            total+=hub[i].bobot[1];
            used[from]=to;
      }
}
```

8. Mencetak hasil akhir dari MST

```
      printf("\nbobot total : %d",total);
```

9. Program Algoritma Kruskal Sudah Selesai.

2.3. PENJELASAN FUNGSI ALGORITMA PRIM

2.3.1. SYNTAX FUNGSI PRIM

```
void prim(){
    printf("\nMenu 7 ( Minimum Spanning Tree - MST )\n\n");
      printf("-- ALGORITMA PRIM --\n\n");
      int n,i,j,awal;
//SYNTAX DIBAWAH INI ADALAH LANGKAH MST - ALGORITMA PRIM DALAM
INISIALISASI VARIABEL & PEMROSESAN PROGRAM
//1. Inisialisasi Variabel & Input
      printf("Masukkan jumlah vertek = ");scanf("%d",&n);
      int M[n][n],visited[n];
      printf("\nMasukkan matriks bobot = \n\n  ");
      for(i=0;i<n;i++){
            printf("%c ",i+65);
      }
      printf("\n");
      for(i=0;i<n;i++){
            printf("%c ",i+65);
            for(j=0;j<n;j++){
                  scanf("%d",&M[i][j]);
            }
      }
    int jumlah=0,min,end,start,bobot=0;
//2. Keterangan - Keterangan
    //jumlah = jumlah pengecekan yg dilakukan
    //min = jarak minimum
    //end = menyimpan vertek tujuan
    //start = menyimpan vertek asal
    //bobot = total mst
    //inisialisasi semua vertek belum terkunjungi
    for(i=0;i<n;i++){
            visited[i]=0;
      }
//3. Input vertek awal, sehingga nantinya vertek awal menjadi
sudah terkunjungi
    printf("\nMasukkan vertex awal yang dipilih : ");
scanf("%d", &awal);
    visited[awal-1]=1;
//4. Pengecekan dilakukan sampai jumlah edge yang didapat sebanyak
jumlah vertek -1
    //karena vertek awal tidak dicek lagi
    while(jumlah<n-1){
      min=999;
      for(i=0;i<n;i++){
//5. Kondisi - Kondisi
    //jika vertek i sudah dikunjungi
    //dan bobot vertek i terhadap j lebih kecil dari min , dan
juga tidak 0
    //maka bobot min akan dubah menjadi bobot vertek i terhadap j
    //start akan menyimpan vertek i sebagai vertek awal
    //dan end akan menyimpan vertek j sebagai vertek tujuan
            if(visited[i]==1){
                  for(j=0;j<n;j++){
                        if(M[i][j]<min&&M[i][j]!=0){
                              min=M[i][j];
                              start=i;
```

```
                                end=j;
                                }
                           }
                     }
               }
//6. Syntax dibawah ini menunjukkan bobot vertek start dan end
akan diubah menjadi 0
               M[start][end]=0;
               M[end][start]=0;
//7. jika vertak start dan vertek end belum terkunjungi maka
mereka tidak membuat cycle
//8. jika keduanya sudah terkunjungi maka edge selanjutnya akan
membentuk cycle
               if(visited[start]==0||visited[end]==0){
//9. Mencetak edge, menambahkan bobot total
    //dan merubah status vertek end menjadi sudah terkunjungi
    //dan jumlah edge bertambah
                     printf("[%c] - [%c]: %d\n",start+65,end+65,min);
                     bobot=bobot+min;
                     visited[end]=1;
                     jumlah++;
               }
      }
//10. Mencetak bobot akhir
      printf("\nTotal bobot adalah = %d \n\n",bobot);
}
//<END, PROGRAM MST - ALGORITMA PRIM SUDAH SELESAI>
```

2.3.2. TAHAPAN ALGORITMA DARI FUNGSI PRIM

1. Pada Fungsi ini kita memerlukan variabel untuk menyimpan jumlah vertek (n) dan juga variabel yang menyimpan vertek awal kita memulai algoritma prim ini (awal). Variabel yang menghitung jumlah edge yang dibuat atau jumlah pengecekan (jumlah) , variabel yang menyimpan vertek asal pada saat pengecekan (start) , variabel yang menyimpan vertek tujuan pada saat pengecekan (end) , variabel yang menyimpan bobot minimum (min) dan bobot total (bobot).

   ```
   int n,i,j,awal;
   int jumlah=0,min,end,start,bobot=0;
   ```

2. Kita juga memerlukan array 2 dimensi berukuran n x n untuk menyimpan matriks bobot (m[n][n]) , dan juga arra berukuran n untuk menyimpan indicator apakah vertek sudah di kunjungi atau belum.

   ```
   int M[n][n],visited[n];
   ```

3. Melakukan input matriks bobot.

   ```
   printf("Masukan matriks bobot = \n  ");
   for(i=0;i<n;i++){
         printf("%c\t",i+65);
   }
   printf("\n");
   for(i=0;i<n;i++){
   ```

```
        printf("%c|",i+65);
        for(j=0;j<n;j++){
              scanf("%d",&M[i][j]);
        }
}
```

4. Menginisialisasi bahwa semua vertek belum terkunjungi , dan juga input vertek awal. Status Vertek awal akan diubah menjadi sudah terekunjungi.

```
for(i=0;i<n;i++){
      visited[i]=0;
}
printf("\nMasukkan vertex awal yang dipilih : \n");
scanf("%d", &awal);
visited[awal-1]=1;
```

5. Melakukan algoritma prim

   - ✓ Pada setiap kali pengecekan kita akan menginisialisasi nilai min dengan nilai yang sangat besar , pada program ini akan diinisialisasi dengan nilai 999
   - ✓ Selanjutnya program akan mengecek dari vertek 0 sampai n , jika vertek i sudah dikunjungi maka proses akan dilanjut
   - ✓ Setelah itu program akan mencari bobot terkecil dari vertek yang berhubungan terhadap vertek I dan juga bobot tersebut bukan 0. Jika bobot tersebut lebih kecil dari nilai min , maka nilai min adalah nilai bobot dari edge tersebut (I – j) , dan start akan menyimpan vertek I , dan end menyimpan vertek j. Bobot dari vertek start ke end , dan end ke start akan di set menjadi 0 agar tidak di cek kembali
   - ✓ Selanjutnya mengecek apakah edge (start - end) tersebut membentuk cycle pada tree yang akan dibuat atau tidak. Pengecekan tersebut dilakukan dengan cara jika salah satu dari vertek tersebut atau keduanya belum terkunjungi maka mereka tidak akan membentuk cycle , sedangkan jika keduanya sudah terkunjungi , jika di tambahkan edge tersebut pada tree maka akan membentuk cycle.
   - ✓ Jika edge tersebut tidak membentuk cycle program akan mencetak edge tersebut dan juga bobot dari edge tersebut. Total bobot akan ditambahkan dengan bobot edge tersebut . Dan status vertek end menjadi sudah terkunjungi. Dan juga jumlah edge bertambah 1
   - ✓ Proses ini dilakukan selama jumlah edge masih kurang dari jumlah vertek-1.

```
        while(jumlah<n-1){
              min=999;
              for(i=0;i<n;i++){
              if(visited[i]==1){
                    for(j=0;j<n;j++){
                          if(M[i][j]<min&&M[i][j]!=0){
                                min=M[i][j];
                                start=i;
```

```
                                    end=j;
                                    }
                              }
                        }
                  }
                  M[start][end]=0;
                  M[end][start]=0;
                  if(visited[start]==0||visited[end]==0){
                        printf("[%c]- [%c]:
%d\n",start+65,end+65,min);
                        bobot=bobot+min;
                        visited[end]=1;
                        jumlah++;
                  }
            }
```

6. Mencetak Bobot total pada MST

```
printf("Total bobot adalah = %d ",bobot);
```

7. Program Algoritma Prim Sudah Selesai.

LAMPIRAN
SYNTAX PROGRAM KESELURUHAN

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int pilih=0;

void menu(){
    printf("Nama : Muhammad Firyanul Rizky\n");;
    printf("NIM  : 1708561006\n\n");
    printf("\t\tTUGAS AKHIR MATEMATIKA DISKRIT II\n");
  printf("\tPROGRAM STUDI TEKNIK INFORMATIKA UNUD 2017/2018\n\n");
printf("**============================**=======================**\n");
printf("||     ALGORITMA PADA GRAPH    ||   ALGORITMA PADA TREE  ||\n");
printf("||----------------------------||-----------------------||\n");
printf("|| 1. Pewarnaan Graph         || 5. Kode Prufer         ||\n");
printf("|| 2. Algoritma Djikstra      || 6. Algoritma Kruskal   ||\n");
printf("|| 3. Algoritma Floyd Warshall || 7. Algoritma Prim     ||\n");
printf("|| 4. Algoritma Bellman Ford  ||                       ||\n");
printf("**============================**=======================**\n\n");
      printf("\t\t\tPilihan : 8 = Keluar\n\n");
      printf("Pilihan menu : ");scanf("%d",&pilih);
}

void pewarnaan(){
    printf("\nMenu 1 ( Algoritma Graph )\n\n");
      printf("-- PEWARNAAN GRAPH --\n\n");
    int n,i,j,k,temp,eror=0,warna=0;
    //Inisialisasi Variabel & Input
      printf("Masukkan jumlah vertek = ");scanf("%d",&n);
      int m[n][n],count[n],colored[n],sort[n],save[n];
      printf("Hubungan matriks = \n\n  ");
      for(i=0;i<n;i++){
            printf("%c ",i+65);
      }
      printf("\n");
      for(i=0;i<n;i++){
            printf("%c ",i+65);
            for(j=0;j<n;j++){
                  scanf("%d",&m[i][j]);
            }
      }

//SYNTAX DIBAWAH INI ADALAH LANGKAH PENGISIAN ARRAY COUNT
(REPRESENTASI DERAJAT VERTEK)
//1. Bertujuan untuk mengisi array count(derajat) menjadi 0
//2. Sort(urutan)sesuai dengan urutan vertek dari 0 sampai n
//3. Dalam langkah ini semua vertek belum diwarnai
      for(i=0;i<n;i++){
            count[i]=0;
            sort[i]=i;
            colored[i]=0;
//4. Hitung derajat yang disimpan pada array count
            for(j=0;j<n;j++){
                  count[i]+=m[i][j];
            }
      }
//<END, LANJUT KE SYNTAX & TAHAPAN ALGORITMA BERIKUTNYA>
```

```
//SYNTAX DIBAWAH INI ADALAH LANGKAH SORTING DERAJAT VERTEK
//1. Jika derajat vertek setelahnya pada array sort lebih
besar dari vertek sekarang
//2. Maka, terlebih dahulu urutan vertek selanjutnya akan
dipindahkan ke urutan vertek yang sekarang
      for(i=0;i<n-1;i++){
            for(j=0;j<n-i-1;j++){
                  if(count[j+1]>count[j]){
                        temp=count[j];
                        count[j]=count[j+1];
                        count[j+1]=temp;

                        temp=sort[j];
                        sort[j]=sort[j+1];
                        sort[j+1]=temp;
                  }
            }
      }
//<END, LANJUT KE SYNTAX & TAHAPAN ALGORITMA BERIKUTNYA>

//SYNTAX DIBAWAH INI ADALAH LANGKAH PEWARNAAN DENGAN VERTEK
SEBAGAI PERTIMBANGAN TITIK AWAL & SELANJUTNYA
      for(i=0;i<n;i++){
//1. Inisialisasi save(menyimpan vertek yang tidak saling
berhubungan)
            for(k=0;k<n;k++){
            save[k] = -1;
        }
//2. Jika vertek pada urutan ke i belum diwarnai maka akan
dilakukan proses pewarnaan
            if(colored[sort[i]]==0){
                  ++warna;
                  colored[sort[i]]=warna;
                  for(j=0;j<n;j++){
//3. Jika vertek urutan ke i dan urutan ke j tidak saling
berhubungan dan vertek j belum diwarnai, maka
    //syntax dibawah ini adalah langkah selanjutnya
if(m[sort[i]][sort[j]]==0&&colored[sort[j]]==0){
//4. Mengecek apakah vertek urutan ke j memiliki hubungan
terhadap vertek lain
    //dan tidak pula berhubungan dengan vertek i serta sudah
lebih dulu diwarnai
                        for(k=0;k<n;k++){
                              if(save[k]!=-1){
                  if(m[save[k]][sort[j]]!=0){
                                          eror=1;
                                    }
                              }
                        }
//5. Langkah dibawah adalah representasi Jika ada vertek
urutan ke j
    // tidak memiliki hubungan terhadap vertek lain dan
tidak pula berhubungan
    // dengan vertek i yang sudah diwarnai sebelumnya
//6. Selanjutnya Vertek urutan ke j akan diberi warna yang
sama dengan vertek urutan ke i
                        if(eror==0){
                              colored[sort[j]]=warna;
```

```
                                save[j]=sort[j];
//7. Menyimpan vertek urutan ke-j kedalam array save untuk
mengecek apakah vertek lain tidak saling terhubung
    //dengan vertek urutan ke i
//8. Jika memiliki hubungan, maka akan menimbulkan konflik
dengan vertek lain
    //dan representasi 0 adalah tanda bahwa vertek urutan
ke-j tidak memiliki hubungan dengan vertek urutan ke i
                                }
                           eror=0;
                           }
                     }
               }
      }
//<END, LANJUT KE SYNTAX & TAHAPAN ALGORITMA BERIKUTNYA>

//SYNTAX DIBAWAH INI ADALAH LANGKAH DALAM MENAMPILKAN HASIL
AKHIR
      for(i=0;i<n;i++){
    printf("\nwarna vertek %c = warna %d",i+65,colored[i]);
      }
      printf("\n\ntotal bilangan kromatik = %d\n\n",warna);
}
//<END, PEWARNAAN GRAPH SUDAH SELESAI>

void djikstra(){
    printf("\nMenu 2 ( Algoritma Graph )\n\n");
      printf("-- ALGORITMA DJIKSTRA --\n\n");
      int n,count,start,min,before,i,j;
      printf("Masukkan jumlah Vertek = "); scanf("%d",&n);
      int m[n][n],visited[n],jarak[n],prev[n];
      //input
      printf("\nMasukkan matriks hubungan : \n\n  ");
      for(i=0;i<n;i++){
            printf("%c ",i+65);
      }
      printf("\n");
      for(i=0;i<n;i++){
            printf("%c ",i+65);
            for(j=0;j<n;j++){
                  scanf("%d",&m[i][j]);
            }
            for(j=0;j<n;j++){
      if(m[i][j]==0){
            m[i][j]=999;
                  }
            }
      }
//SYNTAX DIBAWAH INI ADALAH LANGKAH ALGORITMA DJIKSTRA DALAM
INISIALISASI VARIABEL & PEMROSESAN PROGRAM
//1. Inisialisasi jarak setiap vertek dari jarak awal = 999
//2. Dalam tahap ini semua vertek belum dikunjungi
    for(i=0;i<n;i++){
      visited[i]=0;
      jarak[i]=999;
      }
//3. Input vertek awal
      printf("\nMasukkan vertek awal =
");scanf("%d",&start);
```

```
      start-=1;
//4. Mengubah jarak vertek awal menjadi 0
      jarak[start]=0;
//5. Mencari sorthest path dan dilakukan sebanyak jumlah
vertek
      while(count<n){

            min=999;
//6. Mencari jarak terkecil (lebih kecil dari min) yang ada
pada vertek belum terkunjungi
            for(i=0;i<n;i++){
                  if(jarak[i]<min && visited[i]==0){
    //maka jarak min menjadi jarak dari setiap vertek
//7. variabel Before akan menyimpan vertek yang memiliki
jarak minimum
                        min=jarak[i];
                        before=i;
                  }
            }
            visited[before]=1;
            //before diubah menjadi sudah terkunjungi
//8. Mengubah jarak vertek yang terhubung dengan vertek
before
            for(i=0;i<n;i++){
        //jika vertek tersebut belum terkunjungi dan jika
            //jarak vertek before(min) ditambah dengan jarak
vertek before ke vertek i
                //maka jarak vertek tersebut akan diubah
menjadi
                  //jarak vertek before ditambah dengan
jarak vertek before ke vertek i
                  if(visited[i]==0){
                        if(min+m[before][i]<jarak[i]){
                              jarak[i]=min+m[before][i];
                              prev[i]=before; //9. Prev
menyimpan vertek yang menuju vertek i yaitu vertek before
                        }
                  }
            }
            count++;
      }
//SYNTAX DIBAWAH INI ADALAH LANGKAH ALGORITMA DJIKSTRA DALAM
MENAMPILKAN HASIL AKHIR
      for(i=0;i<n;i++){
            if(i!=start){
        printf("\nJarak dari titik %c ke %c adalah =
%d\nDengan jalur %c ke %c adalah :
",start+65,i+65,jarak[i],start+65,i+65);
        j=i;
        printf("%c <-- ",j+65);
            while(j!=start){
                j=prev[j];
                printf("%c ",j+65);
                if(j!=start){
                    printf("<-- ");
                }
            } printf("\n");
            }
      }
```

```
      printf("\n");
}
//<END, PROGRAM ALGORITMA DJIKSTRA SUDAH SELESAI>

void floyd(){
    printf("\nMenu 3 ( Algoritma Graph )\n\n");
      printf("-- ALGORITMA FLOYD WARSHALL --\n");
    int n,k,i,j;
//SYNTAX DIBAWAH INI ADALAH LANGKAH ALGORITMA FLOYD WARSHALL
DALAM INISIALISASI VARIABEL & PEMROSESAN PROGRAM
//1. Input Vertek dengan Variabel n
      printf("\nMasukkan jumlah vertek : ");scanf("%d",&n);
      int m[n][n],before[n][n];
      printf("\nMasukkan matriks bobot = \n\n  ");
//2. Input pertama merupakan iterasi awal / iterasi 0
      for(i=0;i<n;i++){
            printf("%c ",i+65);
      }
      printf("\n");
      for(i=0;i<n;i++){
            printf("%c ",i+65);
            for(j=0;j<n;j++){
                  scanf("%d",&m[i][j]);
            }
            for(j=0;j<n;j++){
                  if(m[i][j]==0 && i!=j){
                        m[i][j]=999;
                  }
            }
      }
//3. Akan dilakukan Pengecekan perulangan iterasi, baris,
dan kolom sebanyak jumlah vertek
    //k = perulangan iterasi
    //i = perulangan baris
    //j = perulangan kolom
    for(k=0;k<n;k++){
//4. Menyimpan nilai matriks hasil dari iterasi sebelumnya
ke matriks before
      for(i=0;i<n;i++){
            for(j=0;j<n;j++){
                  before[i][j]=m[i][j];
                  }
            }
      printf("\n\nD(%d) [iterasi ke-%d] :",k+1,k+1);
      for(i=0;i<n;i++){
            printf("\n");
            for(j=0;j<n;j++){
//5. Kondisi - Kondisi
    //jika bobot dari vertek i ke j pada iterasi sebelumnya
lebih besar dari
    //penjumlahan bobot vertek i ke vertek k (iterasi ke -
k) dan bobot vertek k ke vertek j pada iterasi sebelumnya
    //maka nilai dari bobot vertek i ke j akan setara dengan
penjumlahan bobot vertek i ke vertek k
    //dan bobot vertek k ke vertek j
                  if(before[i][j] >
before[i][k]+before[k][j] ){
                        m[i][j]=before[i][k]+before[k][j];
                        }
```

```
                        if(m[i][j]==999){
                              printf("~\t");
                        }
                        else{
                              printf("%d\t",m[i][j]);
                        }
                  }
            }
      }
//6. Mencetak hasil akhir
      printf("\n\nShortest Path (Algoritma Floyd Warshall) :
");
      for(i=0;i<n;i++){
            printf("\n");
            for(j=0;j<n;j++){
                  printf("%d\t",m[i][j]);
            }
      }
      printf("\n\n");
}
//<END, PROGRAM ALGORITMA FLOYD WARSHALL SUDAH SELESAI>

void bellman(){
    printf("\nMenu 4 ( Algoritma Graph )\n\n");
      printf("-- ALGORITMA BELLMAN FORD --\n\n");
      int n,count=0,start,i,j;
//SYNTAX DIBAWAH INI ADALAH LANGKAH ALGORITMA BELLMAN FORD
DALAM INISIALISASI VARIABEL & PEMROSESAN PROGRAM
//1. Input Vertek dan Matriks Bobot
      printf("Masukkan jumlah vertek : ");scanf("%d",&n);
      int m[n][n],d[n],pred[n],negative[n];
      printf("Masukkan matriks bobot = \n\n  ");
      for(i=0;i<n;i++){
            printf("%c ",i+65);
      }
      printf("\n");
      for(i=0;i<n;i++){
            printf("%c ",i+65);
            for(j=0;j<n;j++){
                  scanf("%d",&m[i][j]);
            }
            for(j=0;j<n;j++){
      if(m[i][j]==0){
            m[i][j]=999;
                  }
            }
      }
//2. Menginisialisasi jarak setiap vertek adalah 999, dan
vertek pembuat negatif tidak ada
    for(i=0;i<n;i++){
      d[i]=999;
      negative[i]=-1;
      }
//3. Menginput vertek awal dan membuat jarak dari titik awal
menjadi 0
    printf("\nMasukkan vertek awal : ");scanf("%d",&start);
    d[start]=0;
//4. Syntax dibawah ini akan menghitung sebanyak jumlah
vertek-1
```

```
    while(count<n-1){
      for(i=0;i<n;i++){
            for(j=0;j<n;j++){
//5. Kondisi - Kondisi
    //jika jarak titik awal ke vertek i dijumlahkan dengan
bobot( jarak) vertek i ke vertek j
    //yang nilainya lebih kecil dari jarak titik awal ke
vertek j.
    //maka jarak titik awal ke vertek j akan diubah menjadi
penjumlahan dari jarak titik awal ke vertek i
    //dengan bobot(jarak) vertek i ke vertek j
                  if(d[i]+m[i][j] < d[j]){
                        d[j]= d[i]+m[i][j];
pred[j]=i;
//variabel array predecessor diatas akan menyimpan vertek
sebelumnya yang menuju vertek i
                        }
                  }
            }
      count++;
      }
//6. Syntax dibawah ini adalah langkah pengecekan negatif
cycle pada suatu vertek
      for(i=0;i<n;i++){
            for(j=0;j<n;j++){
//7. Kondisi - Kondisi
    //jika jarak titik awal ke vertek j berubah lagi pada
pengecekan ini,
    //maka vertek tersebut merupakan vertek terhubung dengan
negatif cycle
                  if(d[i]+m[i][j] < d[j]){
d[j]= (-999); //Syntax disamping adalah jarak dari vertek
awal ke vertek j
              //yang kemudian akan diubah menjadi -999
              //karena akan selalu berkurang setiap
dilakukan pengecekan
negative[j]=i;//Syntax disamping akan menyimpan vertek
              //yang berhubungan dengan vertek j
              //yang membuat jaraknya menjadi negatif cycle
                  }
            }
      }
//8. Menampilkan hasil akhir
      for(i=0;i<n;i++){
            if(i!=start){
printf("\nJarak dari %c ke vertek %c adalah =
",start+65,i+65);
                  if(d[i]==(-999)){
printf("-%d dengan vertek pembuat negative %c
",d[i],negative[i]+65);
                  }
                  else{
                        printf("%d",d[i]);
                  }
            }
      }
printf("\n\n");
}
//<END, PROGRAM ALGORITMA BELLMAN FORD SUDAH SELESAI>
```

```
void pruf_en(){
    printf("\nMenu 1 ( MST - Prufer )\n\n");
      printf("-- DEKONSTRUKSI PRUFER --\n\n");
      int i,j,k,vertek,a,b;
//SYNTAX DIBAWAH INI ADALAH LANGKAH MST ENKODING KODE PRUFER
DALAM INISIALISASI VARIABEL & PEMROSESAN PROGRAM
//1. Inisialisasi & Input
      printf("Masukkan jumlah vertek :
");scanf("%d",&vertek);
      int code[vertek-2]; //menyimpan code
      int M[vertek][vertek]; //menyimpan matriks bobot
      int drjt[vertek]; //menyimpan derajat
      //Syntax Dibawah ini adalah inisialisasi code kosong ,
derajat semua vertek 0 , dan matriks bobot kosong
      for(i=0;i<vertek;i++){
            code[i]=0;
            drjt[i]=0;
            for(j=0;j<vertek;j++){
                  M[i][j]=0;
            }
      }
      for(i=0;i<vertek-1;i++){
            printf("\nMasukkan edge = ");scanf("%d
%d",&a,&b);
            M[(a-1)][(b-1)]=M[(b-1)][(a-1)]=1;
      }
//2. Syntax dibawah ini adalah langkah menghitung derajat
      for(i=0;i<vertek;i++){
            for(j=0;j<vertek;j++){
                  drjt[i]+=M[i][j];
            }
            printf("\nDerajat vertek %d = %d ",i+1,drjt[i]);
      }
//Syntax dibawah ini adalah langkah dalam mencari kode
prufer
//3. kode prufer suatu tree berjumlah sebanyak jumlah vertek
- 2
      for(k=0;k<vertek-2;k++){
//4. mencari vertek dengan angka terkecil dan derajat 1
//5. perulangan ini akan mendapatkan nilai vertek dengan
angka terkecil
//6. karena di mulai dari vertek awal dan jika sudah ketemu
kode untuk edge pada vertek tersebut
//7. maka perulangan akan dipaksa berhenti, dengan begitu
akan didapat vertek terkecil
            for(i=0;i<vertek;i++){
                  if(drjt[i]==1){
//8. Mencari vertek yang berhubungan dengan vertek i
                        for(j=0;j<vertek;j++){
                              if(M[i][j]==1){
                                    code[k]=j+1;
                //kode yang disimpan adalah vertek j
                                    drjt[i]--;
                                    drjt[j]--;
                                    break;
                              }
                        }
                        M[i][code[k]]=0;
```

```
                        M[code[k]][i]=0;
                        break;
                  }
            }
      }
//9. Syntax Dibawah ini adalah langkah dalam mencetak hasil
akhir
      printf("\n\nKode Prufer = ");
      for(i=0;i<vertek-2;i++){
            printf("%d ",code[i]);
      }
      printf("\n\n");
}
//<END, PROGRAM MST - DEKONSTRUKSI PRUFER SUDAH SELESAI>

void pruf_dec(){
      int jml_k,i,j,k,jml_v,temp;
    printf("\nMenu 2 ( MST - Prufer )\n\n");
      printf("-- REKONSTRUKSI PRUFER --\n\n");
//SYNTAX DIBAWAH INI ADALAH LANGKAH MST DEKODING KODE PRUFER
DALAM INISIALISASI VARIABEL & PEMROSESAN PROGRAM
//1. Inisialisasi Variabel & Input
      printf("Masukkan jumlah kode = ");scanf("%d",&jml_k);
      int kode[jml_k];
      printf("\nMasukkan kode = ");
      for(i=0;i<jml_k;i++){
            scanf("%d",&kode[i]);
      }
      jml_v=jml_k+2; //Syntax disamping menunjukkan jumlah
vertek sama dengan jumlah kode + 2
      int vertek[jml_v];
      int m[jml_v][jml_v];
      for(i=0;i<jml_v;i++){
vertek[i]=i+1;
//2. Mengisi array vertek dengan angka sesuai urutan dari 1
- jumlah vertek
            for(j=0;j<jml_v;j++){
                  m[i][j]=0;
            }
      }
      printf("\nTree = ");
      for(i=0;i<jml_k;i++){
            for(j=0;j<jml_v;j++){
if(kode[i]!= vertek[j] && vertek[j]!=0 && kode[i]!=0 ){
                        for(k=0;k<jml_k;k++){
                              temp=0;
                              if(kode[k]!=vertek[j]){
                                    temp=1;
                              }
                              else{
                                    break;
                              }
                        }
                        if(temp==1){
printf("\n[%d] - [%d]",kode[i],vertek[j]);
m[kode[i]-1][vertek[j]-1]=m[vertek[j]-1][kode[i]-1] = 1;
                              kode[i]=0;
                              vertek[j]=0;
```

```
                        }
                  }
            }
      }
      for(i=0;i<jml_v;i++){
            if(vertek[i]!=0){
                  for(j=0;j<jml_v;j++){
                        if(vertek[i]!=vertek[j] &&
vertek[j]!=0 ){
printf("\n[%d] - [%d]",vertek[i],vertek[j]);
m[vertek[i]-1][vertek[j]-1]=m[vertek[j]-1][vertek[i]-1] = 1;
                              vertek[i]=0;
                              vertek[j]=0;
                        }
                  }
            }
      }
      printf("\n\nDengan matriks ketetanggaan  = \n\n  ");
      for(i=1;i<=jml_v;i++){
            printf("%d ",i);
      }
      for(i=0;i<jml_v;i++){
            printf("\n%d ",i+1);
            for(j=0;j<jml_v;j++){
                  printf("%d ",m[i][j]);
            }
      }
      printf("\n\n");
}
//<END, PROGRAM MST - REKONSTRUKSI PRUFER SUDAH SELESAI>

//SYNTAX DIBAWAH INI ADALAH FUNGSI PEMANGGILAN KEDUA FUNGSI
PRUFER DIATAS
void prufer(){
      int sel;
    printf("\nMenu 5 ( Minimum Spanning Tree - MST )\n\n");
      printf("   -- KODE PRUFER --  \n\n");
      printf("1. Dekonstruksi Prufer\n");
      printf("2. Rekonstruksi Prufer\n\n");
      printf("Pilihan = ");scanf("%d",&sel);
      switch(sel){
            case 1:
                  system("cls");
                  pruf_en();
            break;
            case 2:
                  system("cls");
                  pruf_dec();
            break;
      }
}

void kruskal(){
    printf("\nMenu 6 ( Minimum Spanning Tree - MST )\n\n");
      printf("-- ALGORITMA KRUSKAL --\n\n");
//SYNTAX DIBAWAH INI ADALAH LANGKAH MST - ALGORITMA KRUSKAL
DALAM INISIALISASI VARIABEL & PEMROSESAN PROGRAM
//1. Input akan disimpan dalam bentuk struct yang menyimpan
vertek terhubung (edge) dan bobot dari edge tersebut
```

```
	struct data{
		int bobot[1];
		int edge[2];
	};

	int total=0;
	int i,j,vertek,edge,temp,cek,t,k=0;
//2. Inisialisasi Variabel & Input
	printf("Masukkan jumlah vertek :
");scanf("%d",&vertek);
	printf("Masukkan jumlah edge : ");scanf("%d",&edge);
	int graph[vertek][vertek];
	struct data hub[edge];
	int used[edge];

//3. Inisialisasi nilai awal bobot semua edge infinity
	for(i=0;i<vertek;i++){
		for(j=0;j<vertek;j++){
			graph[i][j]=999;
		}
	}
//4. Mengosongkan array used
	for(i=0;i<edge;i++){
		used[i]=0;
	}
//5. Input bobot
	printf("\n");
	for(i=0;i<edge;i++){
printf("Masukkan edge :
");scanf("%d%d",&hub[i].edge[1],&hub[i].edge[2]);
printf("Bobot : ");scanf("%d",&hub[i].bobot[1]);
graph[hub[i].edge[1]-1][hub[i].edge[2]-1]=hub[i].bobot[1];
graph[hub[i].edge[2]-1][hub[i].edge[1]-1]=hub[i].bobot[1];
		printf("\n");
	}
//6. Cetak matriks - matriks ketetanggaan
	printf("Matriks ketetanggaan : \n\n");
	for(i=0;i<vertek;i++){
		for(j=0;j<vertek;j++){
			if(graph[i][j]!=999){
				printf("%d ",graph[i][j]);
			}
			else{
				printf("~ ");
			}
		}
		printf("\n");
	}
//7. Sorting berdasarkan bobot dari terkecil hingga terbesar
	for (i=0;i<edge-1;i++){
	for (j=0;j<edge-i-1;j++){
		if (hub[j+1].bobot[1] < hub[j].bobot[1]){
		    temp = hub[j].edge[1];
		    hub[j].edge[1]=hub[j+1].edge[1];
		    hub[j+1].edge[1]=temp;
		    temp = hub[j].edge[2];
		    hub[j].edge[2]=hub[j+1].edge[2];
		    hub[j+1].edge[2]=temp;
		    temp = hub[j].bobot[1];
```

```
                hub[j].bobot[1] = hub[j+1].bobot[1];
                hub[j+1].bobot[1] = temp;
            }
        }
      }

//8. Syntax Dibawah menunjukkan hasil sorting
      printf("\n*============================*\n");
      printf("        hasil sorting        \n\n");
      for(i=0;i<edge;i++){
            printf("  edge %d - %d <=> bobot = %d
\n",hub[i].edge[1],hub[i].edge[2],hub[i].bobot[1]);
      }
        printf("*============================*\n");

//9. Syntax Dibawah menunjukkan minimum spanning tree
      printf("\n*============================*\n");
      printf("    Minimum Spanning Tree    \n\n");
      int from,to;
    //from menyimpan vertek awal
      //to menyimpan vertek tujuan
      for(i=0;i<edge;i++){
            from=hub[i].edge[1];
            to=hub[i].edge[2];
//10. Syntax Dibawah akan mengecek apakah edge tersebut
membentuk cycle atau tidak
      //dengan cara mencari union graph tersebut
      //jika from dan to merupakan union(gabungan) graph
yang sama
      //maka mereka akan membentuk cycle
      //dan jika akhir dari union from dan to sama maka
mereka membentuk cycle
      //dengan kata lain jika from sama dengan to
      //maka vertek awal bertemu kembali dengan dirinya
sendiri dan akan membentuk cycle
            while(used[from]){
                  from=used[from];
            }
            while(used[to]){
                  to=used[to];
            }
            if(from!=to){
        printf("    edge %d - %d bobot = %d
\n",hub[i].edge[1],hub[i].edge[2],hub[i].bobot[1]);
                  total+=hub[i].bobot[1];
//11. Syntax Dibawah adalah langkah menambahkan total bobot
                  used[from]=to;
        //syntax diatas akan menyimpan vertek terakhir pada
union(gabungan)
        //yang dibuat oleh vertek from
            }

      }
        printf("*============================*\n");
//12. Mencetak bobot total
      printf("\nbobot total : %d\n\n",total);
}
//<END, PROGRAM MST - ALGORITMA KRUSKAL SUDAH SELESAI>
```

```
void prim(){
    printf("\nMenu 7 ( Minimum Spanning Tree - MST )\n\n");
      printf("-- ALGORITMA PRIM --\n\n");
      int n,i,j,awal;
//SYNTAX DIBAWAH INI ADALAH LANGKAH MST - ALGORITMA PRIM
DALAM INISIALISASI VARIABEL & PEMROSESAN PROGRAM
//1. Inisialisasi Variabel & Input
      printf("Masukkan jumlah vertek = ");scanf("%d",&n);
      int M[n][n],visited[n];
      printf("\nMasukkan matriks bobot = \n\n  ");
      for(i=0;i<n;i++){
            printf("%c ",i+65);
      }
      printf("\n");
      for(i=0;i<n;i++){
            printf("%c ",i+65);
            for(j=0;j<n;j++){
                  scanf("%d",&M[i][j]);
            }
      }
    int jumlah=0,min,end,start,bobot=0;
//2. Keterangan - Keterangan
    //jumlah = jumlah pengecekan yg dilakukan
    //min = jarak minimum
    //end = menyimpan vertek tujuan
    //start = menyimpan vertek asal
    //bobot = total mst
    //inisialisasi semua vertek belum terkunjungi
    for(i=0;i<n;i++){
            visited[i]=0;
      }
//3. Input vertek awal, sehingga nantinya vertek awal
menjadi sudah terkunjungi
    printf("\nMasukkan vertex awal yang dipilih : ");
scanf("%d", &awal);
    visited[awal-1]=1;
//4. Pengecekan dilakukan sampai jumlah edge yang didapat
sebanyak jumlah vertek -1
    //karena vertek awal tidak dicek lagi
    while(jumlah<n-1){
      min=999;
      for(i=0;i<n;i++){
//5. Kondisi - Kondisi
    //jika vertek i sudah dikunjungi
    //dan bobot vertek i terhadap j lebih kecil dari min ,
dan juga tidak 0
    //maka bobot min akan dubah menjadi bobot vertek i
terhadap j
    //start akan menyimpan vertek i sebagai vertek awal
    //dan end akan menyimpan vertek j sebagai vertek tujuan
            if(visited[i]==1){
                  for(j=0;j<n;j++){
                        if(M[i][j]<min&&M[i][j]!=0){
                              min=M[i][j];
                              start=i;
                              end=j;
                              }
                        }
                  }
```

```
            }
      //6. Syntax dibawah ini menunjukkan bobot vertek start dan
      end akan diubah menjadi 0
            M[start][end]=0;
            M[end][start]=0;
      //7. jika vertak start dan vertek end belum terkunjungi maka
      mereka tidak membuat cycle
      //8. jika keduanya sudah terkunjungi maka edge selanjutnya
      akan membentuk cycle
            if(visited[start]==0||visited[end]==0){
      //9. Mencetak edge, menambahkan bobot total
          //dan merubah status vertek end menjadi sudah
      terkunjungi
          //dan jumlah edge bertambah
                  printf("[%c] - [%c]:
      %d\n",start+65,end+65,min);
                  bobot=bobot+min;
                  visited[end]=1;
                  jumlah++;
            }
       }
      //10. Mencetak bobot akhir
       printf("\nTotal bobot adalah = %d \n\n",bobot);
      }
      //<END, PROGRAM MST - ALGORITMA PRIM SUDAH SELESAI>

      //SYNTAX DIBAWAH INI ADALAH FUNGSI PEMANGGILAN KESELURUHAN
      FUNGSI ALHORITMA DIATAS
      int main(){
       do{
            do{
                  system("cls");
                  menu();
            }while(pilih < 1 || pilih > 8);
            system("cls");
            switch(pilih){
                  case 1 :
      pewarnaan();system("PAUSE");break;
                  case 2 : djikstra();system("PAUSE");break;
                  case 3 : floyd();system("PAUSE");break;
                  case 4 : bellman();system("PAUSE");break;
                  case 5 : prufer();system("PAUSE");break;
                  case 6 : kruskal();system("PAUSE");break;
                  case 7 : prim();system("PAUSE");break;
            }
       }while(pilih!=8);
      }
//END, PROGRAM AKHIR MATEMATIKA DISKRIT II SUDAH SELESAI
```