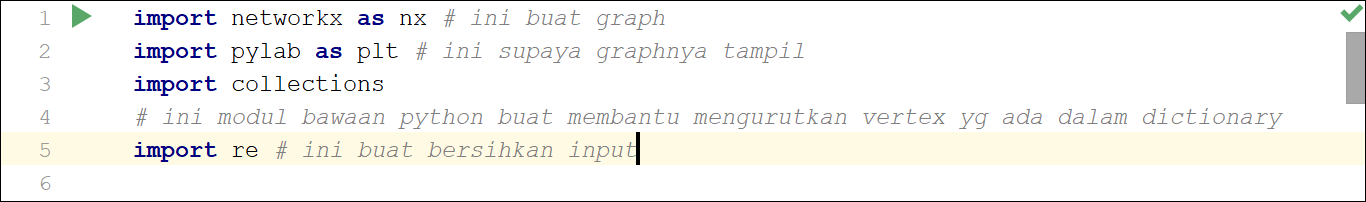
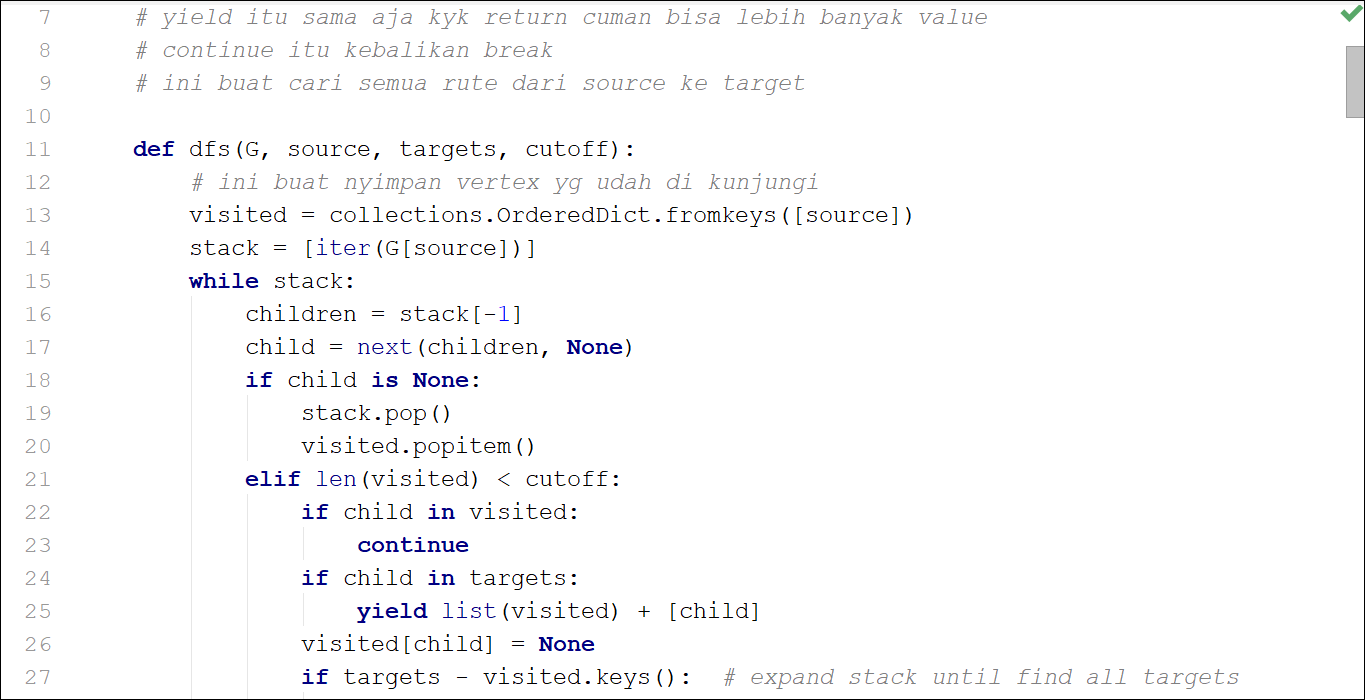
SourceCode **TeoriGraf\_DFS.py**

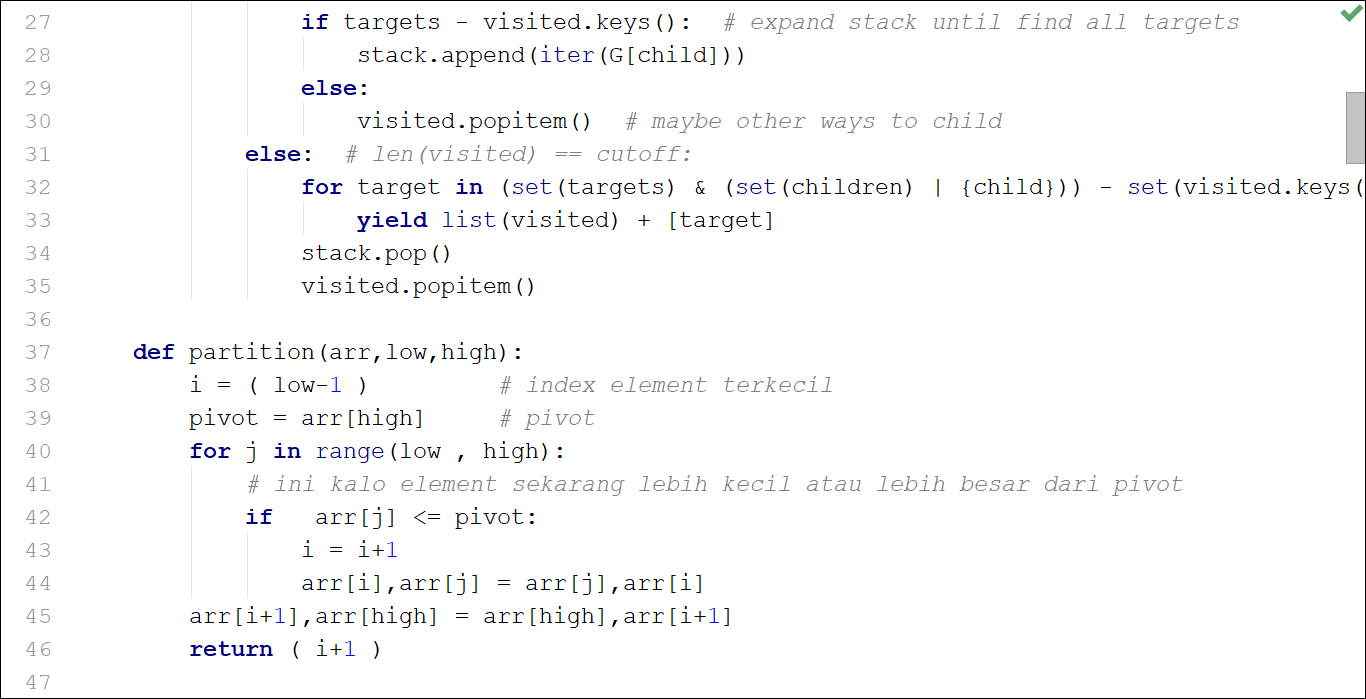
**import** networkx **as** nx *# ini buat graph***import** pylab **as** plt *# ini supaya graphnya tampil***import** collections  
*# ini modul bawaan python buat membantu mengurutkan vertex yg ada dalam dictionary***import** re *# ini buat bersihkan input  
  
# yield itu sama aja kyk return cuman bisa lebih banyak value  
# continue itu kebalikan break  
# ini buat cari semua rute dari source ke target***def** dfs(G, source, targets, cutoff):  
 *# ini buat nyimpan vertex yg udah di kunjungi* visited = collections.OrderedDict.fromkeys([source])  
 stack = [iter(G[source])]  
 **while** stack:  
 children = stack[-1]  
 child = next(children, **None**)  
 **if** child **is None**:  
 stack.pop()  
 visited.popitem()  
 **elif** len(visited) < cutoff:  
 **if** child **in** visited:  
 **continue  
 if** child **in** targets:  
 **yield** list(visited) + [child]  
 visited[child] = **None  
 if** targets - visited.keys(): *# expand stack until find all targets* stack.append(iter(G[child]))  
 **else**:  
 visited.popitem() *# maybe other ways to child* **else**: *# len(visited) == cutoff:* **for** target **in** (set(targets) & (set(children) | {child})) - set(visited.keys()):  
 **yield** list(visited) + [target]  
 stack.pop()  
 visited.popitem()  
  
**def** partition(arr,low,high):  
 i = ( low-1 ) *# index element terkecil* pivot = arr[high] *# pivot* **for** j **in** range(low , high):  
 *# ini kalo element sekarang lebih kecil atau lebih besar dari pivot* **if** arr[j] <= pivot:  
 i = i+1  
 arr[i],arr[j] = arr[j],arr[i]  
 arr[i+1],arr[high] = arr[high],arr[i+1]  
 **return** ( i+1 )  
  
*# Function to do Quick sort***def** quickSort(arr,low,high):  
 **if** low < high:  
 pi = partition(arr,low,high)  
 quickSort(arr, low, pi-1)  
 quickSort(arr, pi+1, high)  
  
**def** Matrix():  
 *# ini buat simpan nama simpul default* defaultV = [**'a'**,**'b'**,**'c'**,**'d'**,**'e'**,**'f'**,**'g'**,**'z'**]  
 *# ini buat matrix default* defaultmatrix = [  
 [0,0,0,1,0,0,1,1],  
 [0,0,1,1,1,0,0,1],  
 [0,1,0,0,0,1,1,1],  
 [1,1,0,0,1,0,0,0],  
 [0,1,0,1,0,1,0,0],  
 [0,0,1,0,1,0,1,0],  
 [1,0,1,0,0,1,0,0],  
 [1,1,1,0,0,0,0,0]  
 ]  
 e = **None  
 while** e != **"sudah"**:  
 e = input(**"Pilih 1 atau 2\n1. buat matrix sendiri atau?\t 2. pake yg sudah ada?\n >>> "**)  
 **if** e == **"1"**:  
 print(**"Anda akan membuat matrix sendiri \nAturan Input!!! \nContoh Vertex: a,b,c,d,e \nContoh Matrix : 1,0,0,0,1\nCatatan! Masukkan matrix perbaris!\n"**)  
 vertex = input(**"Masukkan vertex : "**)  
 V = [x.strip() **for** x **in** vertex.split(**','**)]  
 print(**"Vertex anda : "**,V)  
  
 matrix = []  
 **while** e != **"sudah"**:  
 e = input(**"Masukkan matrix : "**)  
 c = re.split(**","**,e)  
 c = [x.strip() **for** x **in** e.split(**','**)]  
 *# c.append(int(c[2]))* **if** e != **"sudah" and** e != **"undo" and** e != **"show"**:  
 matrix.append(c)  
 **elif** e == **"undo" and** matrix != **""**:  
 matrix.pop()  
 **elif** e == **"show"**:  
 **for** m **in** matrix:  
 print(m)  
 **else**:  
 print(**"Matrix = ["**)  
 **for** m **in** matrix:  
 print(m)  
 print(**"]"**)  
 **elif** e == **"2"**:  
 matrix = defaultmatrix  
 V = defaultV  
 **break  
 else**:  
 print(**"Input Wrong"**)  
  
  
 E = {} *# menyimpan data vertex dan edges* **for** i **in** range(len(matrix)):  
 edge = []  
 **for** j **in** range(len(matrix)):  
 **if** matrix[i][j] != 0:  
 edge.append(V[j])  
 E[str(V[i])] = (edge)  
  
 edges = [] *#membuat list untuk kepenting menggambar graff* **for** i **in** range(len(V)):  
 **for** j **in** range(len(matrix)):  
 **if** matrix[i][j] != 0 **and** matrix[i][j] != **"0"**:  
 edge = (V[i],V[j])  
 edges.append(edge)  
  
 print(**"Setting "**,edges,**"\nEdges Connected ..."**)  
 G = nx.Graph() *# Empty Graph* **for** vertex **in** V: *# Menambahkan semua vertexnya* G.add\_node(vertex)  
 **for** v1,v2 **in** edges: *# Menambahkan edgesnya* **if**(v1 **and** v2 **is not None**):  
 G.add\_edge(v1,v2)  
  
 **while True**:  
 *# Ini masalah GUI graph* pos = nx.spring\_layout(G) *# menentukan posisi node secara random* nx.draw\_networkx\_nodes(G,pos) *# membuat node graph* nx.draw\_networkx\_labels(G,pos) *# membuat label graph* nx.draw\_networkx\_edges(G,pos) *# membuat edge graph* print(**"Displaying Graph ..."**)  
 plt.show() *# menampilkan graph* print(**"Close Graph ..."**)  
 *# ini masalah Input* print(**"Welcome to Pencari Rute"**)  
 print(**"Penting!!! Masukkan titik awal untuk mencari rute atau ketik \"sudah\" kalo udahan!\n"**)  
 *# perulangan buat mastikan inpu gk salah* **while True**:  
 a = input(**"titik awal: "**)  
 **if** a == **"sudah"**:  
 Menu()  
 **elif** a **not in** V:  
 print(**"Vertex yang anda maksud gk ada!!"**)  
 **else**:  
 **while True**:  
 b = input(**"titik akhir: "**)  
 **if** b **not in** V:  
 print(**"Vertex yang anda maksud gk ada!!"**)  
 **else**:  
 **break  
 break** print(**"\nJadi ini adalah semua rute yang bisa anda laluli dari titik {} ke titik {}"**.format(a,b))  
 rute = (list(dfs(G,a,b,len(V))))  
  
 jalur = [] *# nyimpan jumlah panjang rute  
 # perulangan buat ngambil panjang rute* **for** i **in** range(len(rute)):  
 jalur.append(len(rute[i]))  
 print(rute[i])  
  
 *# sorting rute terpendek ke terpanjang* quickSort(jalur,0,len(jalur)-1)  
  
 print(**"\nDiantara semua jalur diatas, jalur terpendeknya adalah : "**)  
 *# tampilkan semua rute* **for** j **in** range(len(rute)):  
 **if**(len(rute[j])== jalur[0]):  
 print (rute[j],end=**"\n"**)  
 print(**"Looping ..."**)  
  
**def** Edges():  
 *# ini buat simpan nama simpul default* defaultV = [**'a'**,**'b'**,**'c'**,**'d'**,**'e'**,**'f'**,**'g'**,**'z'**]  
 *# ini buat edges default* defaultE = [(**'a'**,**'b'**),(**'a'**,**'f'**),(**'b'**,**'c'**),(**'b'**,**'d'**),  
 (**'b'**,**'e'**),(**'c'**,**'e'**),(**'c'**,**'z'**),(**'d'**,**'e'**),(**'d'**,**'f'**),  
 (**'e'**,**'g'**),(**'f'**,**'g'**),(**'g'**,**'z'**)] *# Edges & Weight-nya* e = **None  
 while** e != **"sudah"**:  
 e = input(**"Pilih 1 atau 2\n1. buat edges sendiri atau?\t 2. pake yg sudah ada?\n >>> "**)  
 **if** e == **"1"**:  
 vertex = input(**"Masukkan vertex : "**)  
 V = [x.strip() **for** x **in** vertex.split(**','**)]  
 e =**None** edges = []  
 **while** e != **"sudah"**:  
 e = input(**"Masukkan edges : "**)  
 c = re.split(**","**,e)  
 c = [x.strip() **for** x **in** e.split(**','**)]  
 *# c.append(int(c[2]))* **if** e != **"sudah" and** e != **"undo" and** e != **"show"**:  
 edges.append(c)  
 **elif** e == **"undo" and** edges != **""**:  
 edges.pop()  
 **elif** e == **"show"**:  
 print(edges)  
 **else**:  
 print(edges,**"\nEdges Connected!!"**)  
 **elif** e == **"2"**:  
 edges = defaultE  
 V = defaultV  
 print(**"Edges = ["**)  
 **for** m **in** edges:  
 print(m)  
 print(**"]"**)  
 **break  
 else**:  
 print(**"Input Wrong"**)  
  
 G = nx.Graph() *# Empty Graph* **for** vertex **in** V: *# Menambahkan semua vertexnya* G.add\_node(vertex)  
 **for** v1,v2 **in** edges: *# Menambahkan edgesnya* **if**(v1 **and** v2 **is not None**):  
 G.add\_edge(v1,v2)  
  
 **while True**:  
 *# Ini masalah GUI graph* pos = nx.spring\_layout(G) *# menentukan posisi node secara random* nx.draw\_networkx\_nodes(G,pos) *# membuat node graph* nx.draw\_networkx\_labels(G,pos) *# membuat label graph* nx.draw\_networkx\_edges(G,pos) *# membuat edge graph* print(**"Displaying Graph ..."**)  
 plt.show() *# menampilkan graph* print(**"Close Graph ..."**)  
 *# ini masalah Input* print(**"Welcome to Pencari Rute"**)  
 print(**"Penting!!! Masukkan titik awal untuk mencari rute atau ketik \"sudah\" kalo udahan!\n"**)  
 *# perulangan buat mastikan input gk salah* **while True**:  
 a = input(**"titik awal: "**)  
 **if** a == **"sudah"**:  
 Menu()  
 **elif** a **not in** V:  
 print(**"Vertex yang anda maksud gk ada!!"**)  
 **else**:  
 **while True**:  
 b = input(**"titik akhir: "**)  
 **if** b **not in** V:  
 print(**"Vertex yang anda maksud gk ada!!"**)  
 **else**:  
 **break  
 break** print(**"\nJadi ini adalah semua rute yang bisa anda laluli dari titik {} ke titik {}"**.format(a,b))  
 rute = (list(dfs(G,a,b,len(V))))  
  
 jalur = [] *# nyimpan jumlah panjang rute  
 # perulangan buat ngambil panjang rute* **for** i **in** range(len(rute)):  
 jalur.append(len(rute[i]))  
 print(rute[i])  
  
 *# sorting rute terpendek ke terpanjang* quickSort(jalur,0,len(jalur)-1)  
  
 print(**"\nDiantara semua jalur diatas, jalur terpendeknya adalah : "**)  
 *# tampilkan semua rute* **for** j **in** range(len(rute)):  
 **if**(len(rute[j])== jalur[0]):  
 print (rute[j],end=**"\n"**)  
 print()  
  
**def** Menu():  
 print(**"Welcome to Main Menu"**)  
 e = **None  
 while** e != **"sudah"**:  
 e = input(**"Silahkan pilih metode pembuatan graph! \n 1. Pake Matrix \t 2. Pake Edges dan Vertex langsung\n>>> "**)  
 **if** e == **"1"**:  
 Matrix()  
 **elif** e == **"2"**:  
 Edges()  
 **elif** e != **"sudah" and** e != **"1" and** e !=**"2"**:  
 print(**"Input Wrong"**)  
 **else**:  
 print(**"\nbye...bye..\n"**)  
 exit()  
  
  
Menu()

Penjelasan dan SS

Line 1 - 6 berfungsi untuk import module

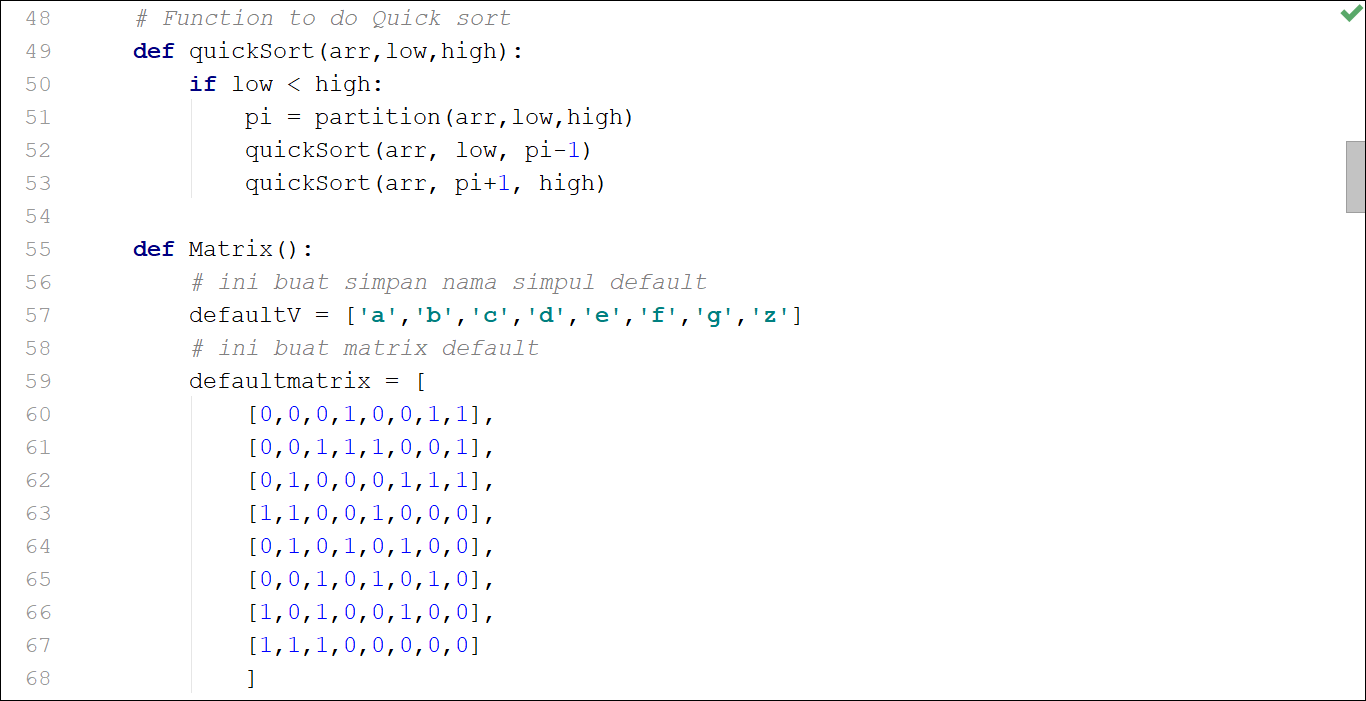
Line 7-36 def dfs(G, source, target, cutoff):





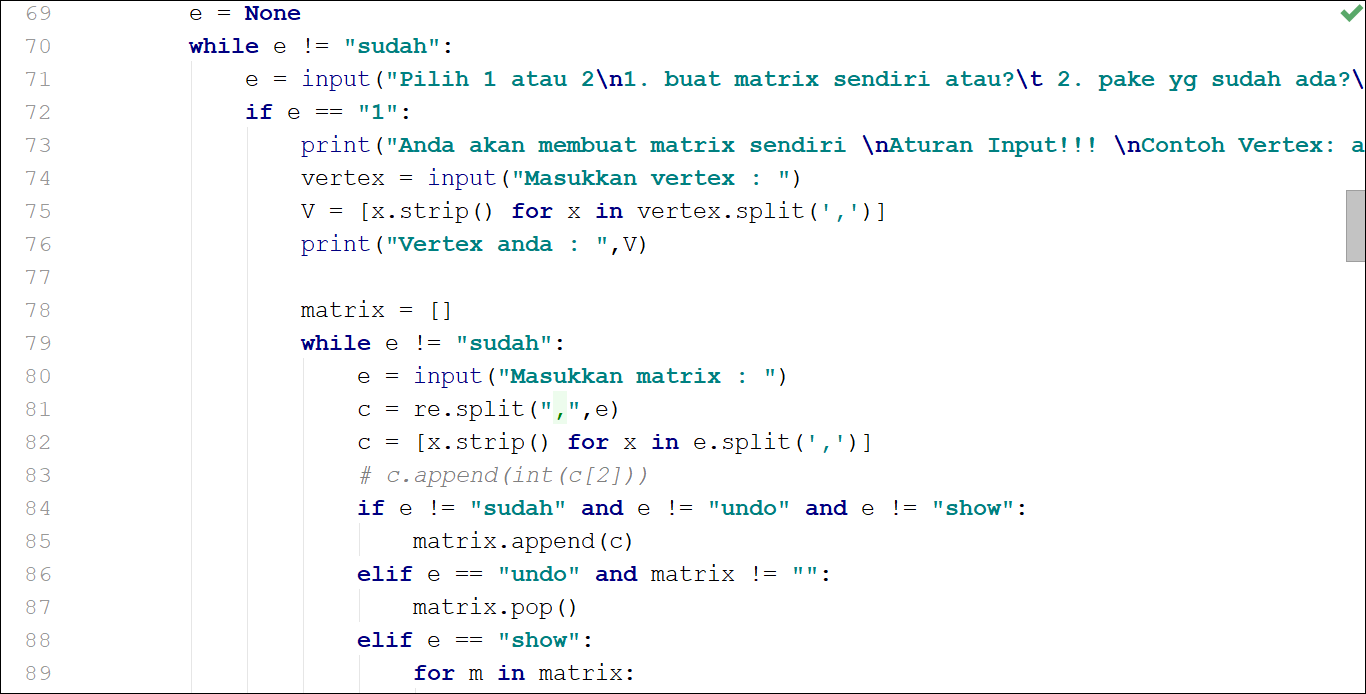
**Penjelasan :**

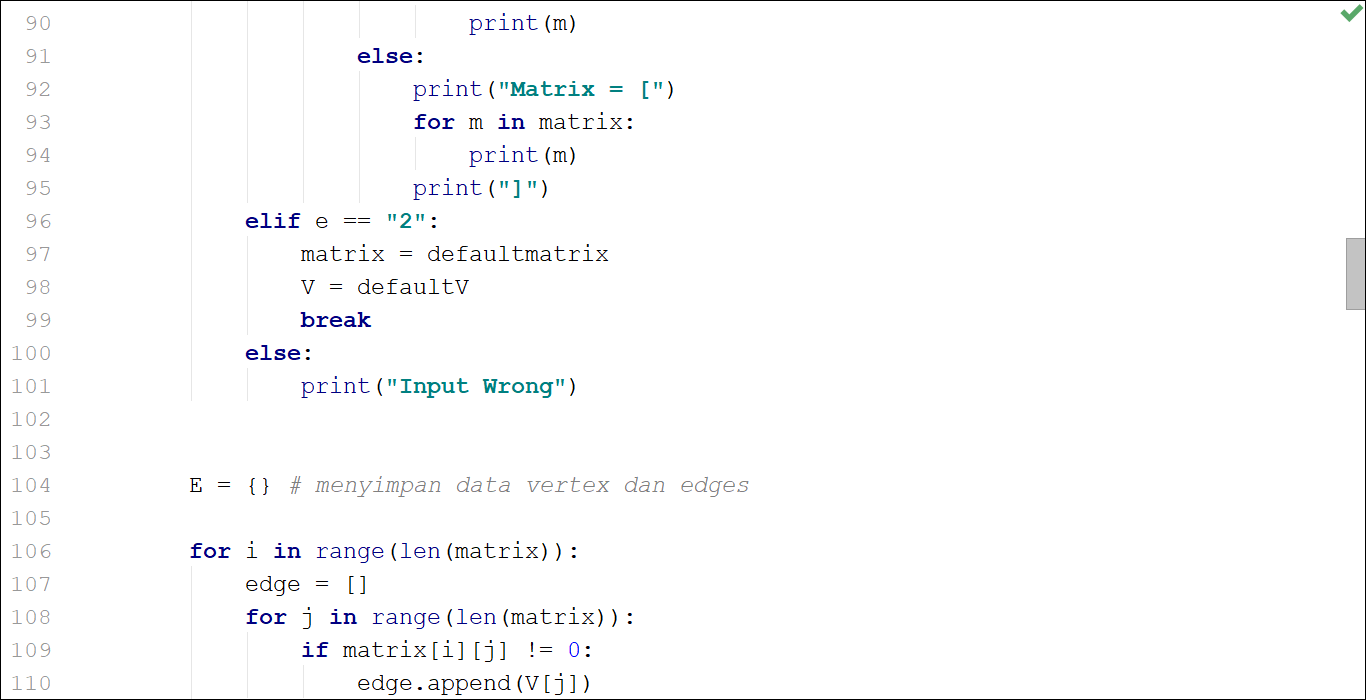
1. **def dfs(G, source, target, cutoff)**
2. **def dfs(G, source, target, cutoff)** berfungsi untuk mencari rute dalam graph dengan metode Depth First Search ( DFS )
3. **def dfs(G, source, target, cutoff)** menerima 4 parameter yaitu :
4. **G** sebagai graph
5. **Source** sebagai titik awal
6. **Target** sebagai titik akhir
7. **Cutoff** sebagai jumlah maksimal langkah pada rute dari titik awal ke titik akhir
8. Proses pertama **def dfs(G, source, target, cutoff)** adalah menyimpan vertex – vertex yang terhubung dengan **source** dalam variabel **visited.**
9. Kedua **def dfs(G, source, target, cutoff)** akan melakukan perulangan dimana selama **stack** masih ada dia akan mencari children dari stack tersebut yang mana adalah **vertex** dari titik awal hingga titik akhir sampai dapat dan apabila ketemu maka akan menyimpan rute yang telah di laluinya, kemudian melakukan pencarian lagi untuk kemungkinan rute lainnya.
10. **def partition(arr, low, high)**
    1. **def partititon(arr, low, high)** memiliki 3 parameter yang masing – masing berfungsi untuk ambil data dari list, data terkeci, data tertinggi.
    2. **def partititon(arr, low, high)** berfungsi untuk melakukan partisi – partisi data untuk keperluan sorting pada **def quickSort(arr, low, high)**

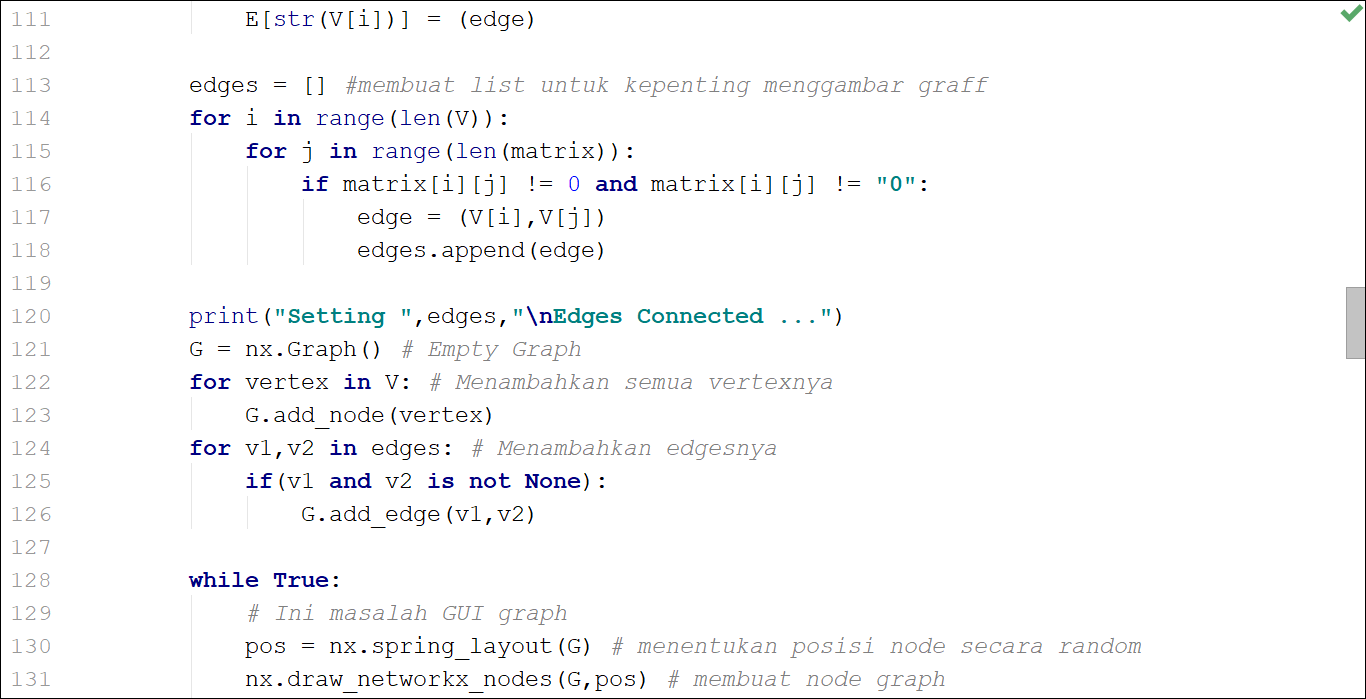


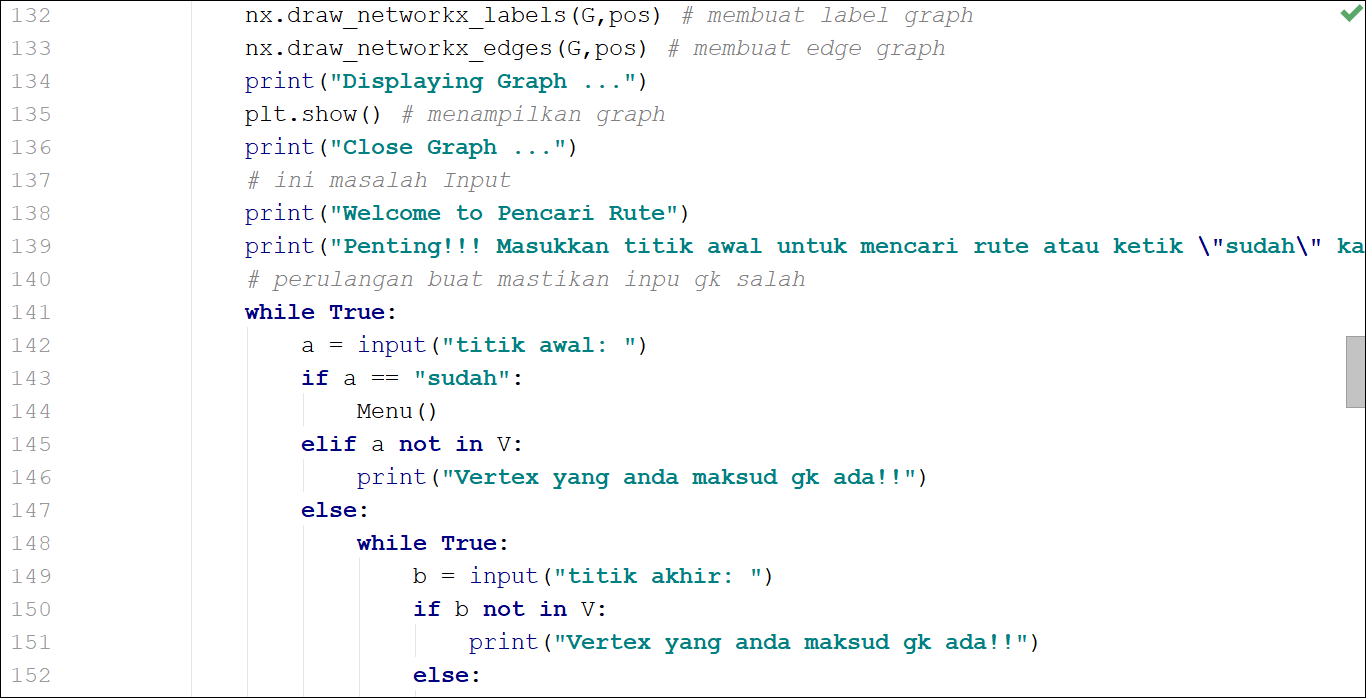
**Penjelasan :**

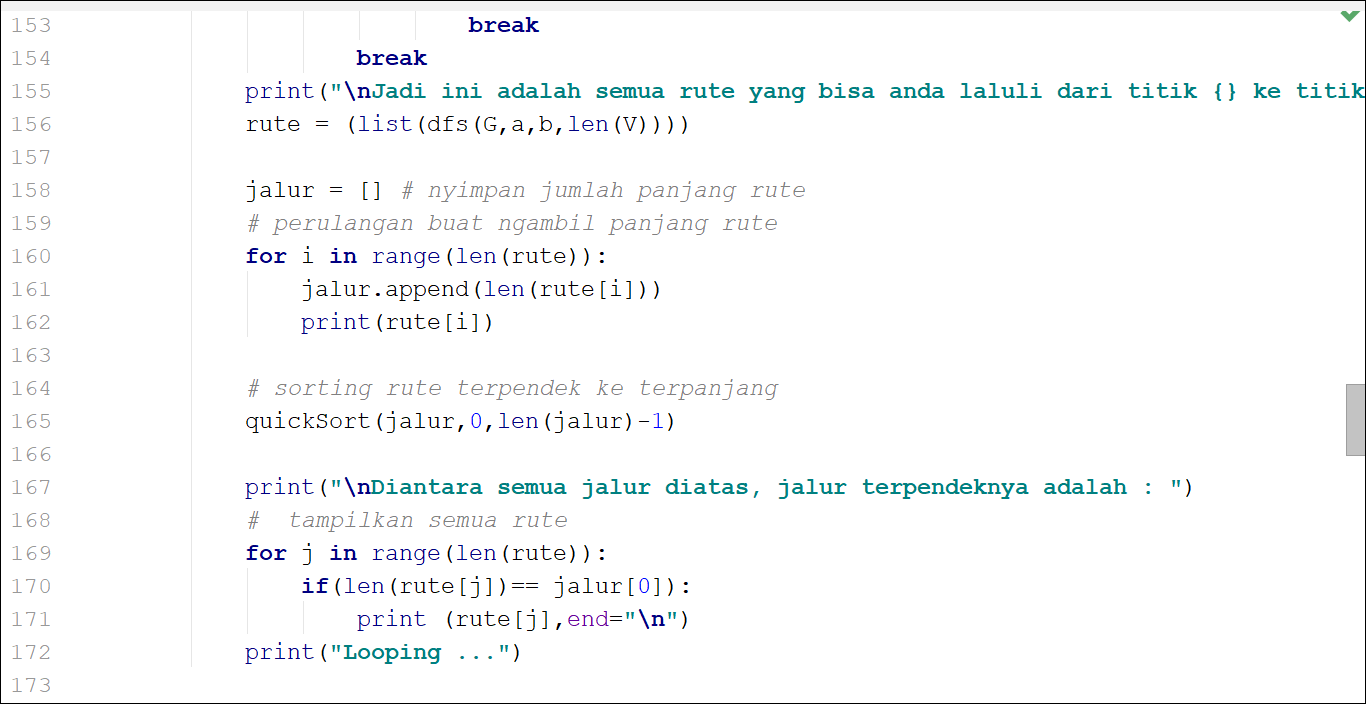
1. **def quickSort(arr, low, high)**
2. **def quickSort(ar, low, high)** berfungsi untuk mengurutkan semua rute yang telah didapat dengan metode Depth First Search ( DFS ) dari yang terpendek hingga yang terpanjang
3. **def quickSort(arr, low, high)** memiliki 3 parameter yang masing – masing berfungsi untuk ambil data dari list, data terkeci, data tertinggi.
4. **def Matrix()**
5. **def Matrix()** berfungsi untuk menjalankan fungsi – fungsi ketika user memilih untuk membuat graph dengan matrix
6. **def Matrix()** memiliki beberapa variable yang berfungsi sebagai berikut :
   1. **defaultV** berfungsi untuk menyimpan vertex default
   2. **defaultMatrix** berfungsi untuk menyimpan matrix default
   3. **e** berfungsi untuk input data
   4. **perulangan While** pada **def Matrix()** rata – rata digunakan untuk mastikan input user itu benar yang di bantu dengan kondisional **if else** di dalamnya
   5. **perulangan For** pada **def Matrix()** rata – rata digunakan masalah conver data, print data dan pengolahan data pada **Matrix dan Graph.**

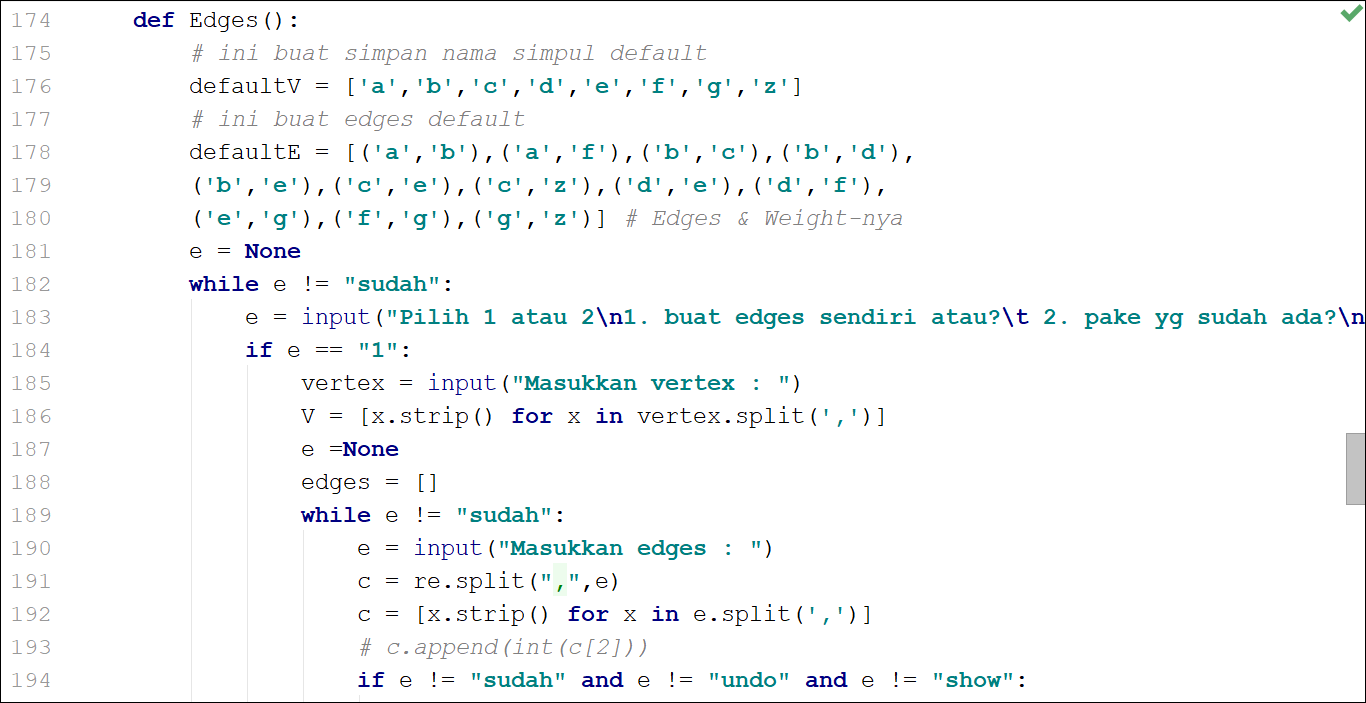






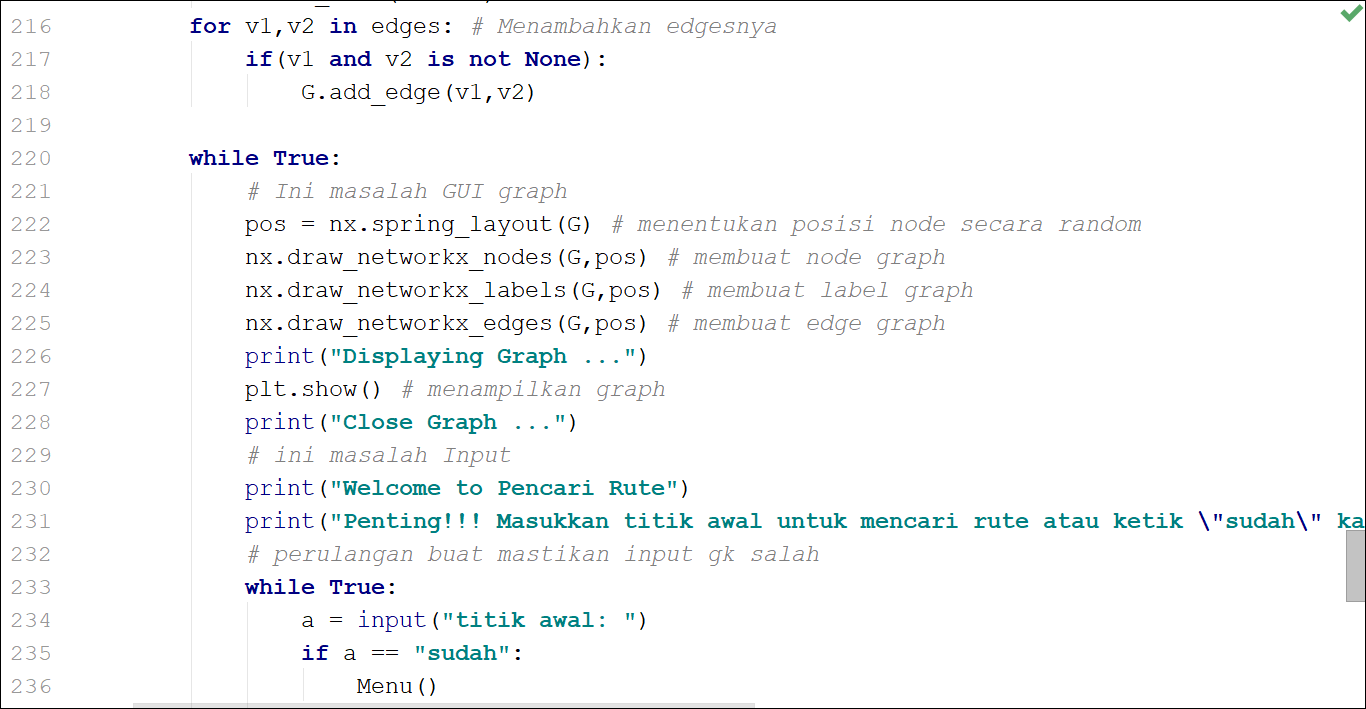


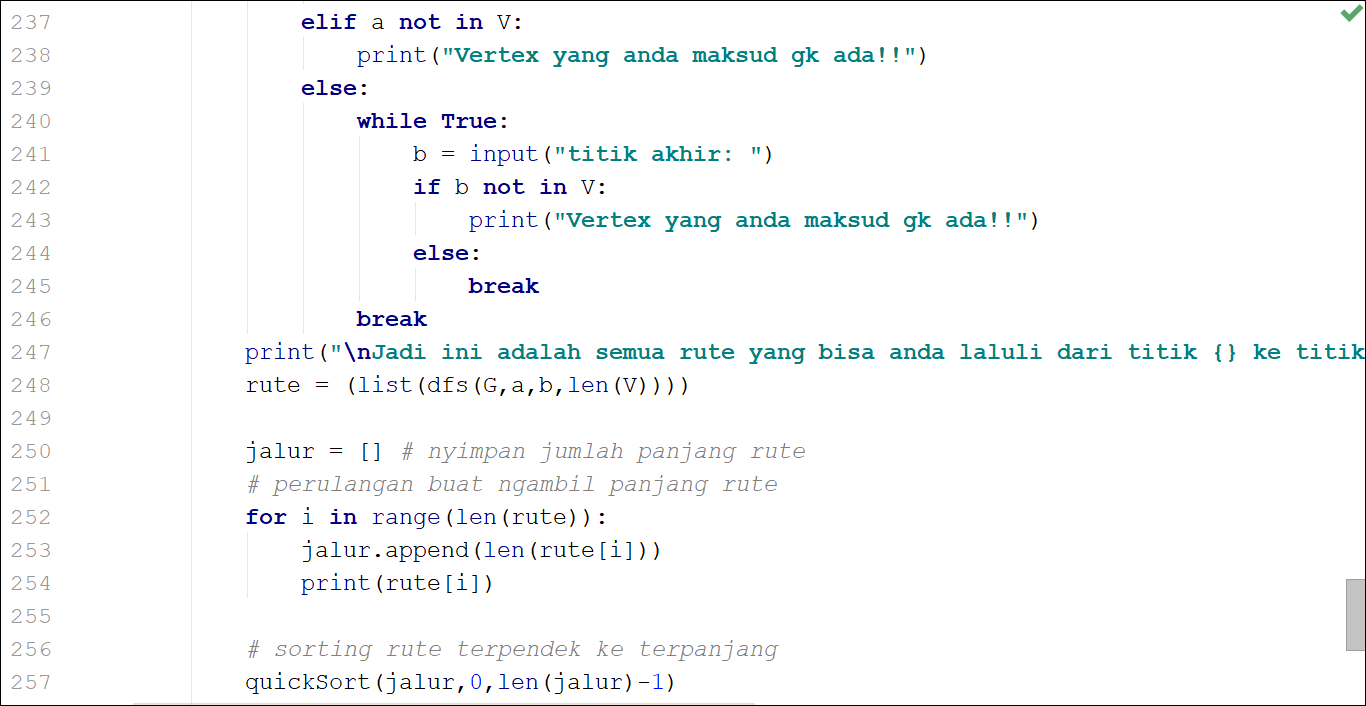


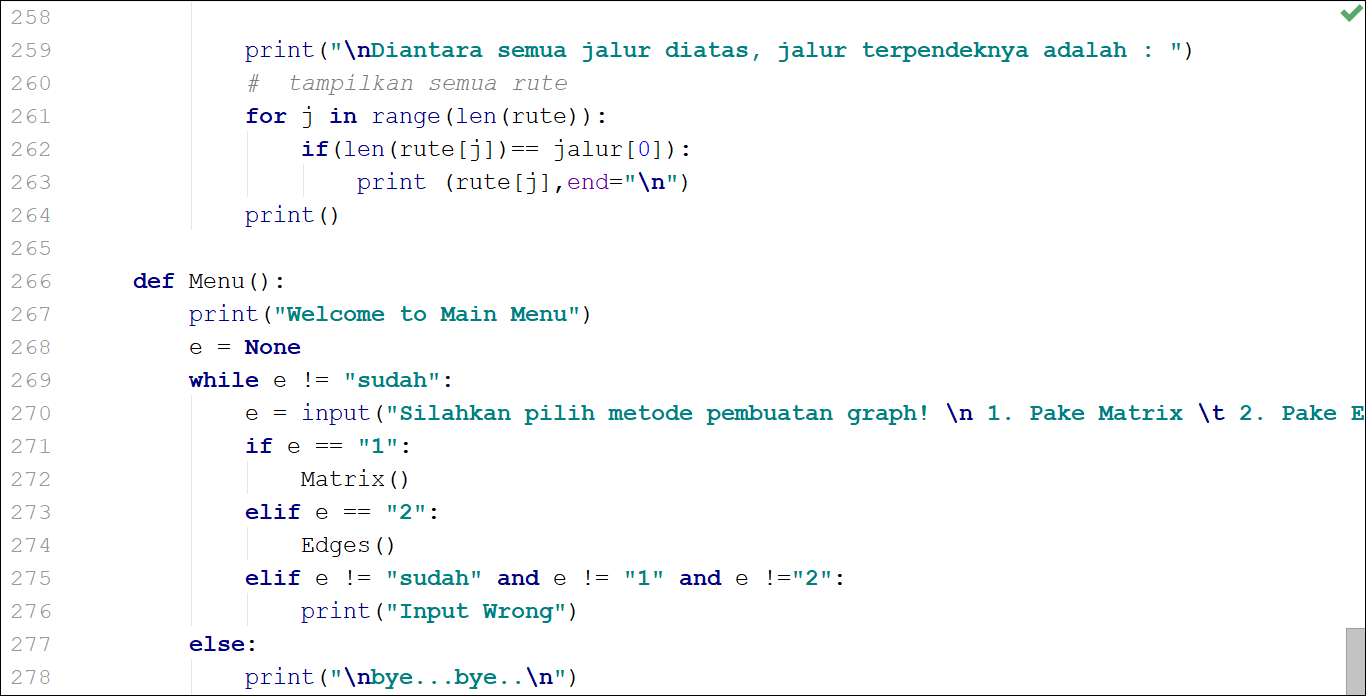


1. **def Edges()**
2. **def Edges()** berfungsi untuk menjalankan fungsi – fungsi ketika user memilih untuk membuat graph dengan matrix
3. **def Edges()** memiliki beberapa variable yang berfungsi sebagai berikut :
   1. **defaultV** berfungsi untuk menyimpan vertex default
   2. **defaultE** berfungsi untuk menyimpan edges default
   3. **e** berfungsi untuk input data
   4. **perulangan While** pada **def Edges()** rata – rata digunakan untuk mastikan input user itu benar yang di bantu dengan kondisional **if else** di dalamnya
   5. **perulangan For** pada **def Eges()** rata – rata digunakan masalah conver data, print data dan pengolahan data pada **Edges dan Graph.**









1. **def Menu()**
2. **def Menu()** berfungsi untuk menampilkan menu dimana user akan memilih untuk menggunak metode apa dalam pembuatan graph
3. **def Menu()** jika user memilih Matrix maka user akan di arahkan ke **def Matrix()** dan sebalikknya ke **def Edges()**
4. **Menu()** berfungsi untuk menjalan program dengan mengeksekusi **def Menu()**

**SS program ketika berjalan**

