OOP Concept
in Machine Learning

REVIEW TAHAPAN MEMBUAT SOFTWARE

Bagian Software

Membuat sebuah software adalah hal yang sudah biasa dilakukan oleh para programmer. Dalam pembuatannya, sebuah software yang sederhana pasti memiliki tiga bagian yaitu:

- Input – memasukan data.
- Proses – mengolah input.
- Output – keluaran dari program, kebutuhan yang diharapkan.

Tahap membuat software

Setelah mengetahui bagian software, sekarang kita masuk ke tahap-tahap yang harus dilakukan dalam membuat sebuah software. Berikut ini adalah tahap-tahapnya:

- **Analisis kebutuhan** ***(requirement analysis)***
  Tahap pertama adalah analisa kebutuhan. Pada tahap ini dilakukan proses pendefinisian masalah. Tujuannya untuk mengetahui permasalahan apa saja yang mungkin terjadi dalam sistem, serta mengetahui program seperti apa yang pengguna inginkan.

  Selain itu, pada tahap ini dilakukan juga untuk menganalisis kebutuhan dari pengguna. Pengumpulan data dapat dilakukan dengan berbagai cara, yakni cara observasi, melakukan wawancara, atau mengumpulkan sampel.

- **Perencanaan** ***(planning)***
  Selanjutnya adalah tahap perencanaan. Perencanaan dilakukan untuk mengefisienkan waktu pembuatan software. Bentuk dari perencanaan dapat berupa penyusunan jadwal kerja, pembagian tugas, atau juga dapat berupa algoritma dari program yang akan dibuat.

- **Pembuatan Desain**
  Seorang UI dan UX designer berperan sangat penting dalam tahap ini. Keduanya berkolaborasi untuk membuat dan merancang desain dari software berdasarkan hasil dari analisis kebutuhan pengguna. Desain tersebut dapat berbentuk flowchart atau prototype yang nantinya diserahkan kepada programmer untuk dibuat menjadi sebuah program atau software.
- **Implementasi**
  Selanjutnya adalah tahap implementasi. Tahap ini mencakup penulisan kode program yang dilakukan oleh developer. Kode tersebut ditulis berdasarkan desain yang telah dibuat sebelumnya.
- **Dokumentasi**
  Pada tahap dokumentasi ini dilakukan setelah tahap implementasi selesai. Dokumentasi berfungsi sebagai panduan untuk proses mengembangkan software dan dapat digunakan sebagai alat untuk menjelaskan software kepada client atau tim lain. Dalam pembuatan software ada dua jenis dokumentasi, yaitu:

  - Dokumentasi Produk
    Dokumentasi yang digunakan untuk menjelaskan software yang sedang dikembangkan.

  - Dokumentasi Proses
    Dokumentasi yang digunakan untuk membuat proses pengembangan software menjadi lebih transparan serta lebih mudah untuk dikelola.

- **Testing**
  Berikutnya adalah tahap testing. Testing adalah tahap di mana software yang sudah dibuat akan diuji coba dan dievaluasi. Pengujian dapat dilakukan untuk mengukur kualitas software dari segi ketepatan, kelengkapan, kegunaan, kinerja, dan segi fungsional maupun non-fungsionalnya. Ada beberapa pengujian yang dilakukan, yaitu:

  - Unit testing
    Unit testing ini adalah pengujian yang dilakukan untuk setiap unit dan modul yang ada dalam software.

  - Integration testing
    Pengujian yang dilakukan untuk menguji integrasi antara unit-unit atau komponen software yang sudah dikombinasikan. Tujuannya untuk memastikan agar semua komponen dapat berinteraksi dan berjalan dengan baik.
  - Validation testing
    Validation testing dilakukan untuk menguji input ke dalam software. Pengujian pada tahap ini dilakukan untuk memastikan software agar dapat menyelesaikan input dengan baik.

    - System testing
      System testing bertujuan untuk memastikan keseluruhan sistem berfungsi dengan baik, serta memenuhi persyaratan dari pengguna. Testing ini dilakukan pada akhir pembuatan software.
- **Deployment**
  Deployment dilakukan setelah seluruh pengujian dilakukan dan software sudah layak untuk diluncurkan dan digunakan.
- **Maintenance & Update**
  Pembuatan software tidak berhenti saat software telah selesai dibuat dan diluncurkan. Namun, kita perlu melakukan perawatan dan pembaruan software agar software tetap bisa digunakan dengan baik oleh pengguna.

OOPs Concepts in Python

- Class
- Objects
- Polymorphism
- Encapsulation
- Inheritance
- Data Abstraction

Creating Classes and objects with methods

```
class Dog:

    # class attribute
    attr1 = "mammal"

    # Instance attribute
    def __init__(self, name):
        self.name = name

    def speak(self):
        print("My name is {}".format(self.name))

# Driver code
# Object instantiation
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")

# Accessing class methods
Rodger.speak()
Tommy.speak()
```

Inheritance in Python

```
# parent class
class Person(object):

    # __init__ is known as the constructor
    def __init__(self, name, idnumber):
        self.name = name
        self.idnumber = idnumber

    def display(self):
        print(self.name)
        print(self.idnumber)

    def details(self):
        print("My name is {}".format(self.name))
        print("IdNumber: {}".format(self.idnumber))

# child class
class Employee(Person):
    def __init__(self, name, idnumber, salary, post):
        self.salary = salary
        self.post = post

        # invoking the __init__ of the parent class
        Person.__init__(self, name, idnumber)

    def details(self):
        print("My name is {}".format(self.name))
        print("IdNumber: {}".format(self.idnumber))
        print("Post: {}".format(self.post))


# creation of an object variable or an instance
a = Employee('Rahul', 886012, 200000, "Intern")

# calling a function of the class Person using
# its instance
a.display()
a.details()
```

Enkapsulasi adalah sebuah peroses pemaketan / penyatu data bersama metode – metodenya, dimana hal ini bermanfaat untuk menyembunyikan rincian – rincian implementasi dari pemakai. Maksud dari enkapsulasi ini adalah untuk menjaga suatu proses program agar tidak dapat diakses secara sembarangan atau di intervensi oleh program lain. Konsep enkapsulasi sangat penting dilakukan untuk menjaga kebutuhan program agar dapat diakses sewaktu-waktu, sekaligus menjaga program tersebut.

```
class Students:
   def __init__(self, name, rank, points):
      self.name = name
      self.rank = rank
      self.points = points

   # custom function
   def demofunc(self):
      print("I am "+self.name)
      print("I got Rank ",+self.rank)

# create 4 objects
st1 = Students("Steve", 1, 100)
st2 = Students("Chris", 2, 90)
st3 = Students("Mark", 3, 76)
st4 = Students("Kate", 4, 60)

# call the functions using the objects created above
st1.demofunc()
st2.demofunc()
st3.demofunc()
st4.demofunc()
```

Output

```
I am Steve
I got Rank  1
I am Chris
I got Rank  2
I am Mark
I got Rank  3
I am Kate
I got Rank  4
```

Linear Regression Implementation From Scratch using Python

Linear Regression is a supervised learning algorithm which is both a statistical and a machine learning algorithm. It is used to predict the real-valued output y based on the given input value x. It depicts the relationship between the dependent variable y and the independent variables x_i (or features). The hypothetical function used for prediction is represented by h(x).

```
h( x ) = w * x + b


here, b is the bias.
x represents the feature vector
w represents the weight vector.
```

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#variables to store mean and standard deviation for each feature
mu = []
std = []

def load_data(filename):
        df = pd.read_csv(filename, sep=",", index_col=False)
        df.columns = ["housesize", "rooms", "price"]
        data = np.array(df, dtype=float)
        plot_data(data[:,:2], data[:, -1])
        normalize(data)
        return data[:,:2], data[:, -1]

def plot_data(x, y):
        plt.xlabel('house size')
        plt.ylabel('price')
        plt.plot(x[:,0], y, 'bo')
        plt.show()

```

$$Z = (x - \mu) / \sigma$$

μ : mean

σ : standard deviation

```
23  def normalize(data):
24          for i in range(0,data.shape[1]-1):
25                  data[:,i] = ((data[:,i] - np.mean(data[:,i]))/np.std(data[:, i]))
26                  mu.append(np.mean(data[:,i]))
27                  std.append(np.std(data[:, i]))
28
29
30  def h(x,theta):
31          return np.matmul(x, theta)
32
33  def cost_function(x, y, theta):
34          return ((h(x, theta)-y).T@(h(x, theta)-y))/(2*y.shape[0])
35
36  def gradient_descent(x, y, theta, learning_rate=0.1, num_epochs=10):
37          m = x.shape[0]
38          J_all = []
39
40          for _ in range(num_epochs):
41                  h_x = h(x, theta)
42                  cost_ = (1/m)*(x.T@(h_x - y))
43                  theta = theta - (learning_rate)*cost_
44                  J_all.append(cost_function(x, y, theta))
45
46          return theta, J_all
47
```

Cost Function

$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Gradient Descent

The main update step for gradient descent is:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (\text{for } j = 0 \text{ and } j = 1)$$

```
47
48  def plot_cost(J_all, num_epochs):
49          plt.xlabel('Epochs')
50          plt.ylabel('Cost')
51          plt.plot(num_epochs, J_all, 'm', linewidth = "5")
52          plt.show()
53
54  def test(theta, x):
55          x[0] = (x[0] - mu[0])/std[0]
56          x[1] = (x[1] - mu[1])/std[1]
57
58          y = theta[0] + theta[1]*x[0] + theta[2]*x[1]
59          print("Price of house: ", y)
60
61  x,y = load_data("house_price_data.txt")
62  y = np.reshape(y, (46,1))
63  x = np.hstack((np.ones((x.shape[0],1)), x))
64  theta = np.zeros((x.shape[1], 1))
65  learning_rate = 0.1
66  num_epochs = 50
67  theta, J_all = gradient_descent(x, y, theta, learning_rate, num_epochs)
68  J = cost_function(x, y, theta)
69  print("Cost: ", J)
70  print("Parameters: ", theta)
71
```

```
71
72  #for testing and plotting cost
73  n_epochs = []
74  jplot = []
75  count = 0
76  for i in J_all:
77          jplot.append(i[0][0])
78          n_epochs.append(count)
79          count += 1
80  jplot = np.array(jplot)
81  n_epochs = np.array(n_epochs)
82  plot_cost(jplot, n_epochs)
83
84  test(theta, [1600, 3])
```

White Box Testing

- Structural Testing or Logic-driven Testing or Glass Box Testing
- Yang dibutuhkan > **Source code**
- Menguji lebih “dekat” tentang detail prosedur perangkat lunak.
- Yang diselidiki: *logical path* (jalur logika) perangkat lunak

Mengapa “**Source code**”?

- Dengan source code, dapat dilakukan pengujian tentang:
  - Structural Testing process
  - Program Logic-driven Testing
  - Design-based Testing
  - Examines the internal structure of program

Jalur Logika

- Conditions
  - If .. Then ..
  - If .. Then .. Else ..
  - If .. Then .. Else if .. Then ..
  - Case .. Of ..
- Loop
  - While .. Do ..
  - Repeat .. Until ..
  - For .. To .. Do ..

Keuntungan White Box

Menghasilkan program yang benar dan sempurna 100%,
karena:

- Mengerjakan seluruh keputusan logika
- Mengerjakan seluruh loop (sesuai batas)
- Menjamin seluruh jalur independen dalam modul dikerjakan minimal 1x
- Mengerjakan seluruh data internal yang menjamin validitas

Syarat:

- Mendefinisikan semua logical path
- Membangun kasus untuk pengujian
- Mengevaluasi hasilnya
- Menguji secara **menyeluruh**

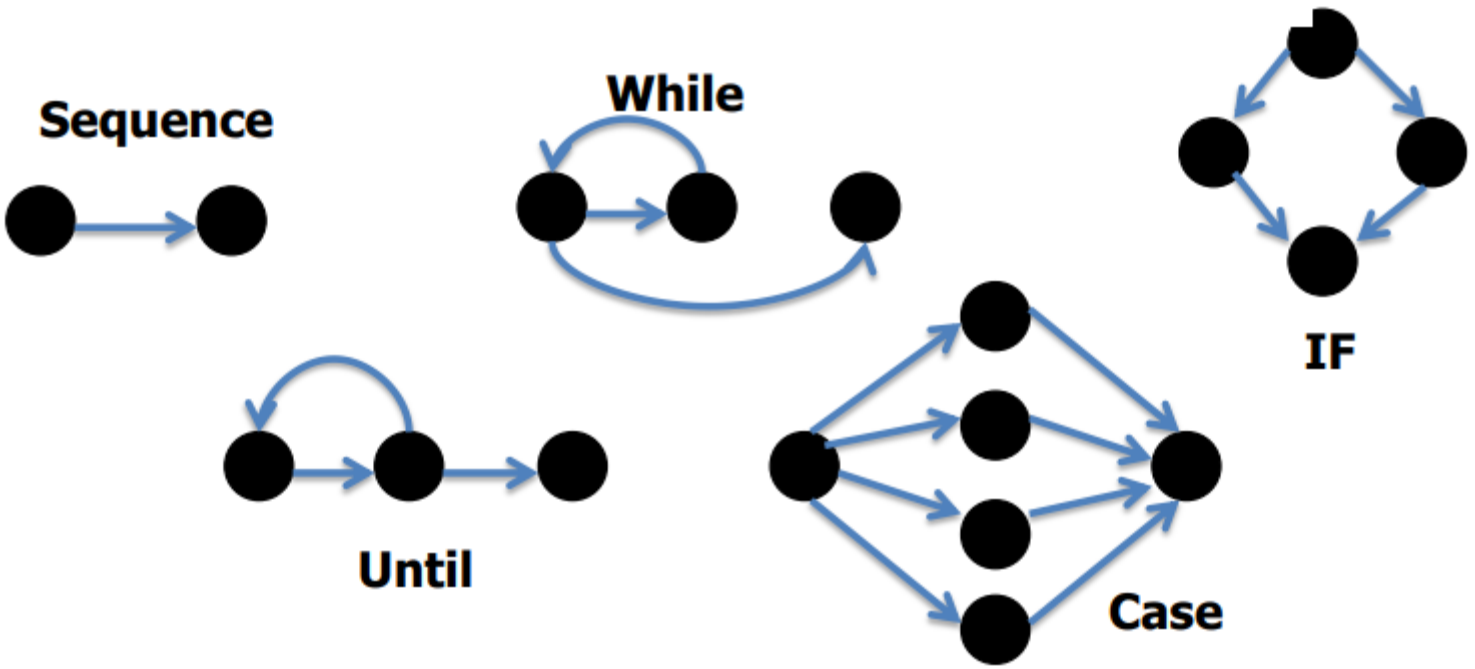
Metode White Box | **Basis path testing**

- Diusulkan oleh Tom McCabe pada tahun 1976
- Digunakan untuk mendapatkan ukuran kompleksitas logika
- Ukuran ini dijadikan sebagai panduan untuk menentukan jalur-jalur utama untuk dieksekusi.

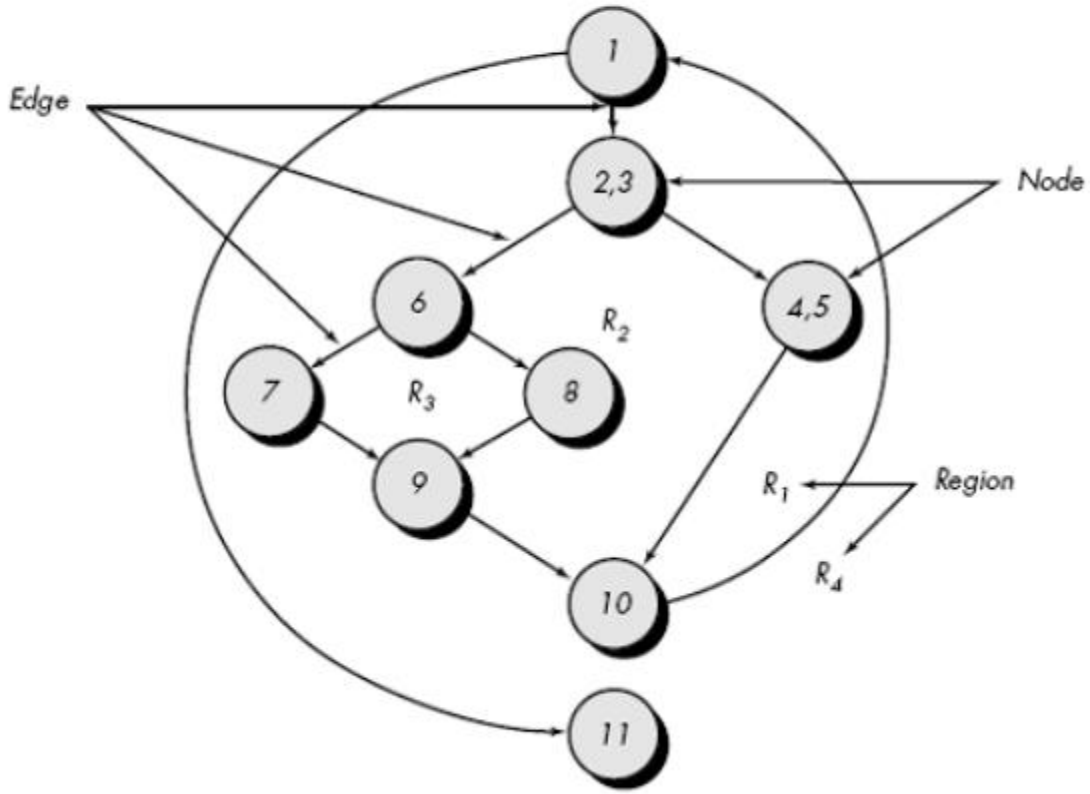
Notasi

Lingkaran (flow graph node)
Menyatakan satu atau beberapa statement prosedural

Panah (edges / links)
Menyatakan aliran kendali (sama dengan panah flowchart)



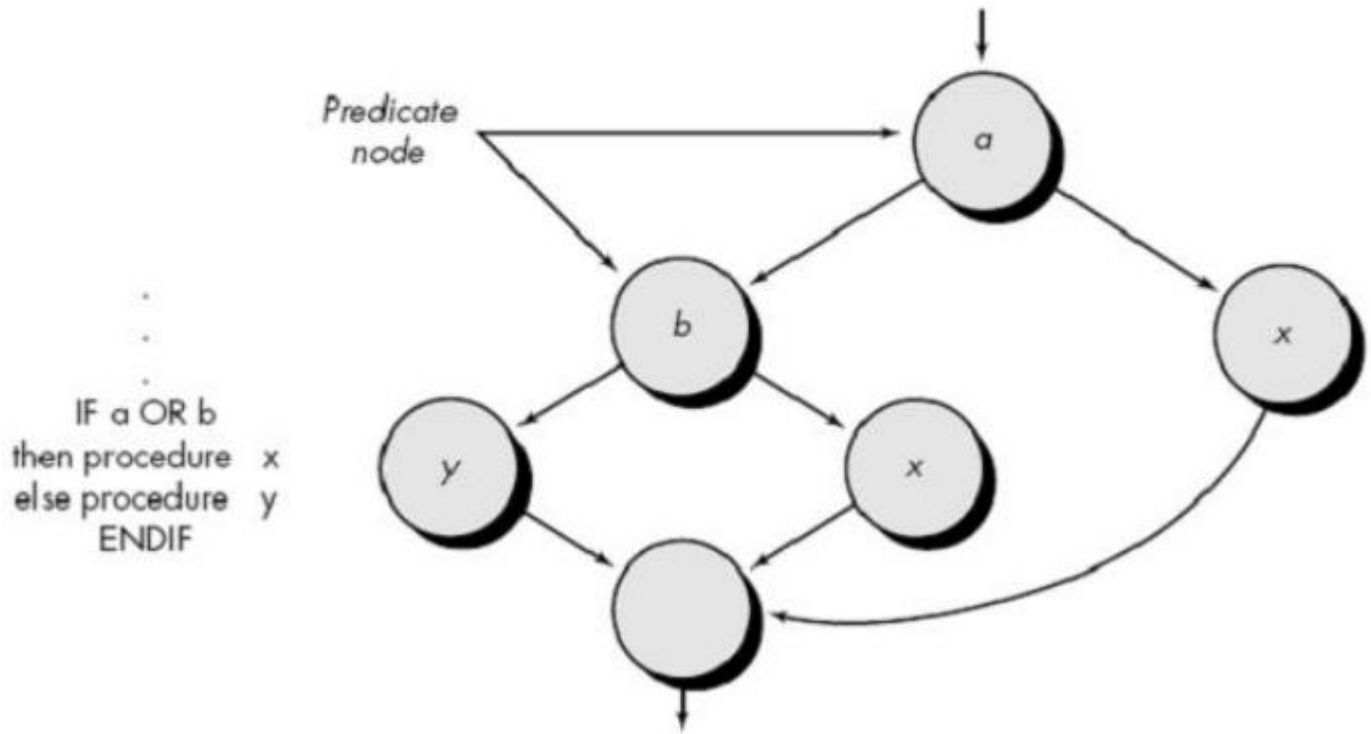
Regions



- Adalah suatu area yang dibatasi oleh edges dan nodes
- Pada saat menghitung regions, area di luar graph ikut ditambahkan

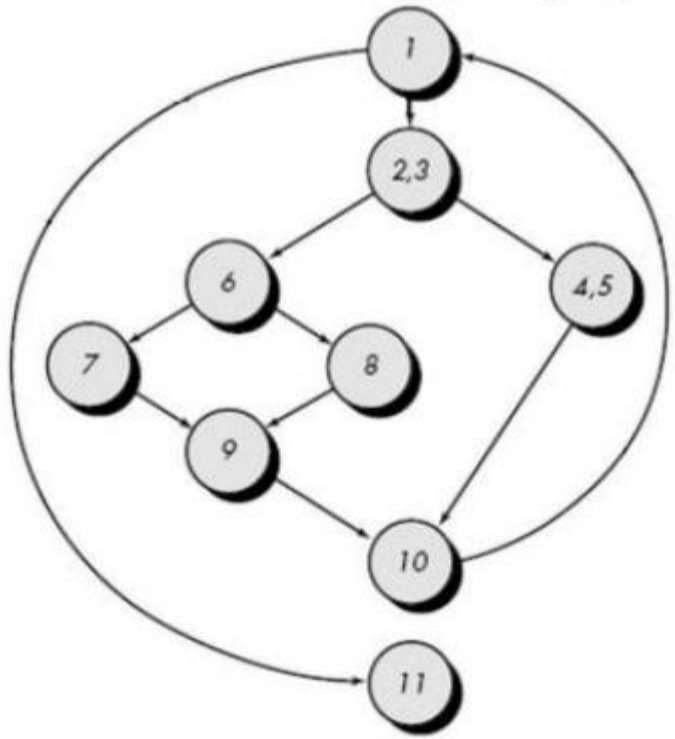
Compound Conditions

- Kondisi majemuk terjadi apabila satu atau lebih dari operator Boolean (AND, OR, NOR, NAND)muncul pada pernyataan kondisi.
- Setiap *node* yang terdapat kondisi disebut ***predicate node***



Independent Path

- Adalah jalur yang melintasi minimal satu kumpulan pernyataan baru atau sebuah kondisi baru pada program.



Path 1 : 1 – 11
Path 2 : 1 – 2 – 3 – 4 – 5 – 10 – 1 – 11
Path 3 : 1 – 2 – 3 – 6 – 8 – 9 – 10 – 1 – 11
Path 4 : 1 – 2 – 3 – 6 – 7 – 9 – 10 – 1 – 11

1-2-3-4-5-10-1-2-3-6-8-9-10-1-11 **bukan** Independent path, karena merupakan kombinasi dari *paths* yang sudah didefinisikan.

Cyclomatic Complexity

- Untuk menentukan banyaknya *independent path* yang merupakan *basis path*
- Rumus:

V(G) = E – N + 2

V(G) = P + 1

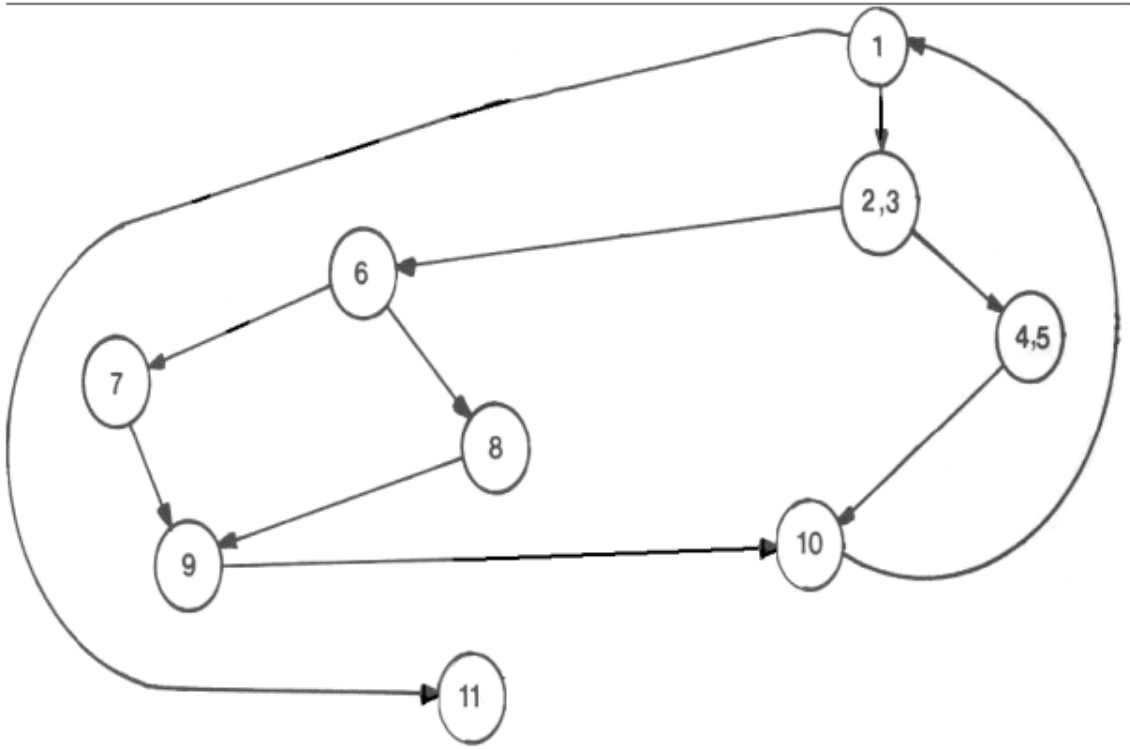
dimana:

E : Edge

N : Node

P : Predicate node

Contoh:



Tentukan independen path?

Jawab :

Dari gambar:

- Path 1 = 1 - 11
- Path 2 = 1 - 2 - 3 - 4 - 5 - 10 - 1 - 11
- Path 3 = 1 - 2 - 3 - 6 - 8 – 9 - 10 - 1 - 11
- Path 4 = 1 - 2 - 3 - 6 - 7 - 9 - 10 - 1 - 11
- Path 1,2,3,4 yang telah didefinisikan diatas merupakan basis set untuk diagram alir.

cyclomatic complexity

- Flowgraph mempunyai 4 region
- V(G) = 11 edge - 9 node + 2 = 4
- V(G) = 3 predicate node + 1 = 4
- Jadi cyclomatic complexity untuk flowgraph adalah 4