PENGUJIAN PERANGKAT LUNAK

-BASIS PATH TESTING-

Eka Widhi Yunarso (EWD)
2017-2

Digunakan dilingkungan internal prodi D3 Manajemen Informatika, Fakultas Ilmu Terapan, Universitas Telkom

Jenis	Kajian	Kompetensi Dasar	Pokok Bahasan	Sub Pokok Bahasan	Materi Bahasan
Teori	Kajian 2: Teknik Pengujian White Box	Mahasiswa mampu membuktikan teknik pengujian white box	Basis Path Testing	Basis Path Testing,	Basis Path Testing, sejarah, definisi dan Notasi
Teori			Flowgraph dan Cyclomatic Complexity	Flowgraph dan Cyclomatic Complexity	- regions - cyclomatic complexity - Flowgraph kondisional - Flowgraph pengulangan - independent path
Teori			Basis Path Worksheet dan data uji	Basis Path Worksheet dan data uji	- kegunaan basis path worksheet - cara melengkapi basis path worksheet - cara menentukan data uji

Basis Path Testing

Telkom University

TECHNIQUES FOR TESTING - DYNAMIC

Basis Path Testing

- a white-box testing technique, proposed by Tom McCabe, 1976
- to derive a logical complexity measure of a procedural design, and use this measure as a guide for defining a basis set of execution paths
- test cases derived to exercise every *statement and branch* in the program at least once during testing (statement/branch coverage)
- if every condition in a compound condition is considered, *condition* coverage can be achieved
- Steps:
  - Draw a (control) *flow graph*, using the flowchart or the code
  - Calculate the *cyclomatic complexity*, using the flow graph
  - Determine the *basis set* of linearly independent paths
  - Design *test cases* to exercise each path in the basis set

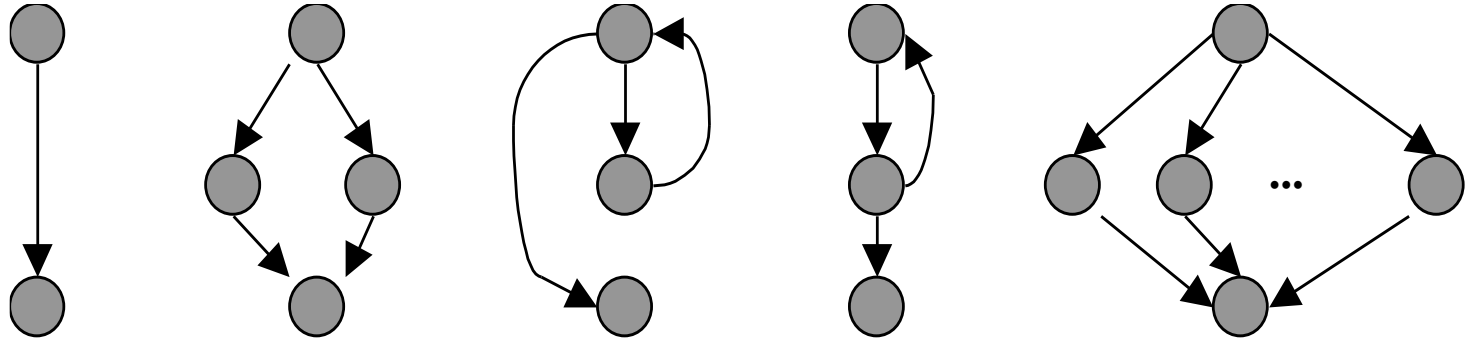
Telkom University

BASIS PATH TESTING

Flow Graph

- used to depict program control structure
- can be drawn from a flowchart (a procedural design representation)
- can be drawn from a piece of source code
- Flow Graph Notation
  - a flow graph composed of edges and nodes
  - an edge starts from a node and ends to another node

Sequence if-then-else While Repeat-until Case



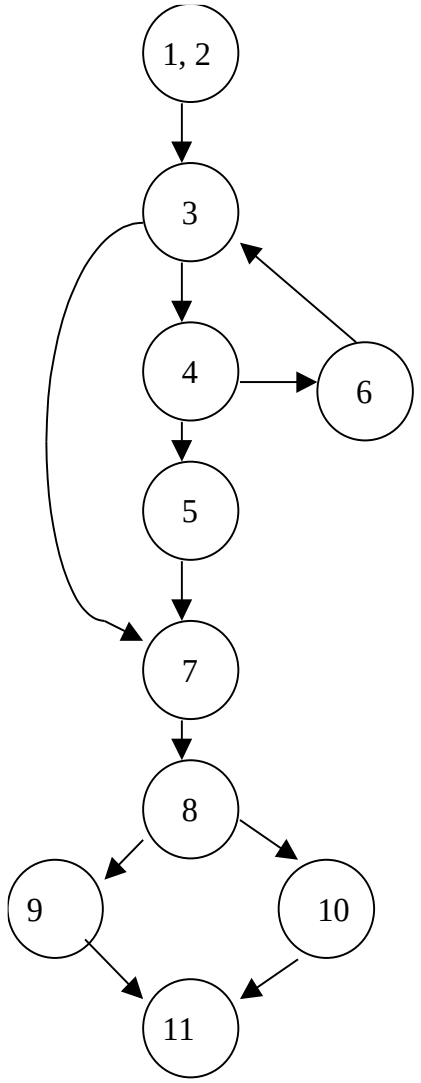
Telkom University

BASIS PATH TESTING

Flow Graph

- Draw a flow graph from source code

```
1 procedure insert(a, b, n, x)
2   begin bool found:=false;
3       for I:=1 to n do
4               if a[I]=x
5                 then found:=true; goto leave endif
6       enddo;
7       leave:
8       if found
9        then b[I]:=b[I]+1
10       else n:=n+1; a[n]:=x; b[n]:=1 endif
11 end insert
```

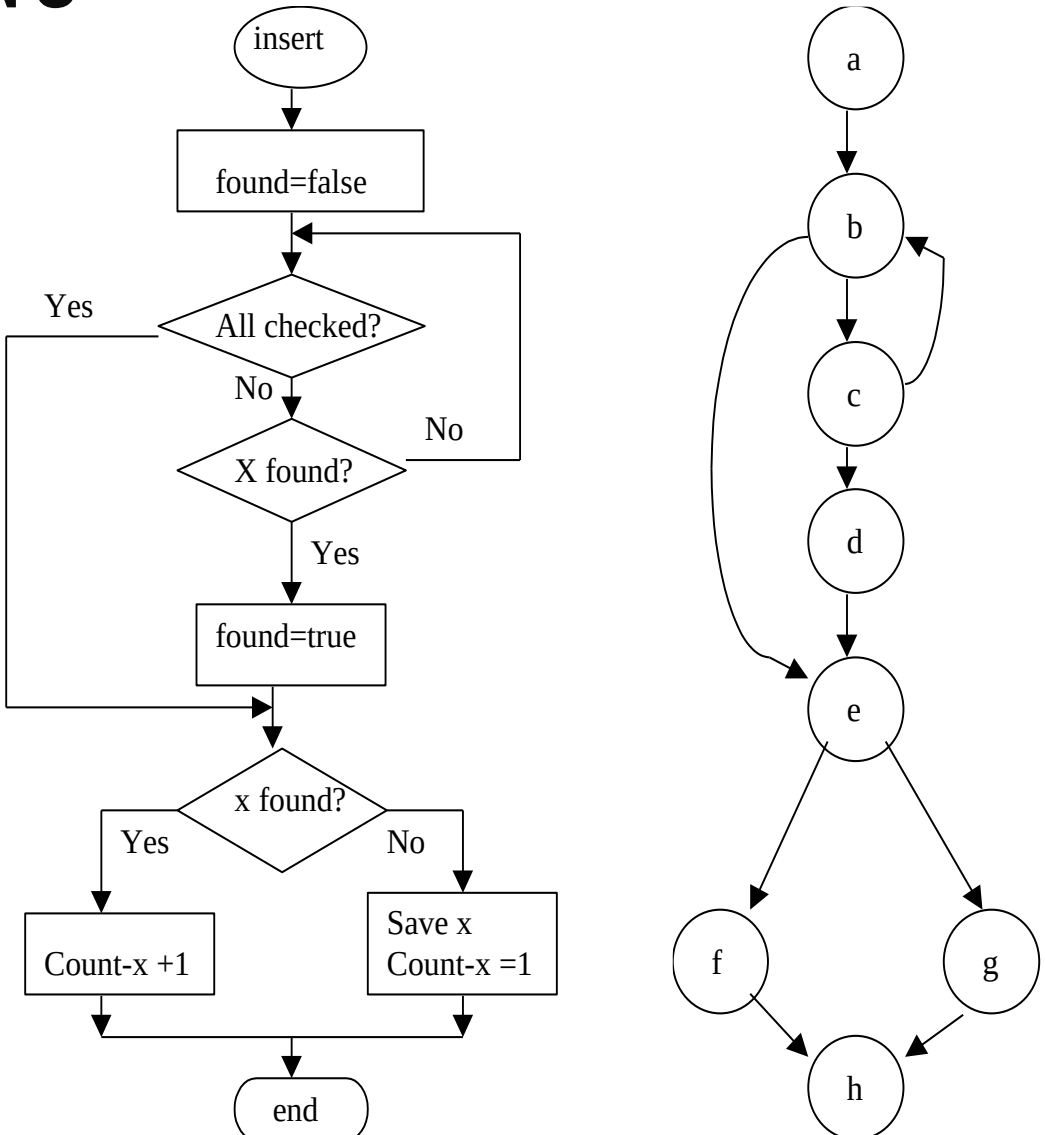


Telkom University

BASIS PATH TESTING

Flow Graph

- Draw a flow graph from a flowchart



Telkom University

BASIS PATH TESTING

Cyclomatic Complexity

- a software metric that provides a quantitative measure of the logical complexity of a program
- Basis set: is a maximal linearly independent set of paths through a graph
- An independent path: is any path through a program that introduces at least one new set of processing statements or a new condition (I.e. at least one new edge in a flow graph)
- Cyclomatic complexity defines the number of independent path in the basis set of a program
- gives an upper bound for the number of tests that must be conducted to achieve statement/branch/condition coverage
- How to calculate cyclomatic complexity:

$$cc = e - n + 2p$$

- e - number of edges; n - number of nodes; p - number of components;
- if all nodes in a graph are connected, then p = 1, thus

$$cc = e - n + 2$$

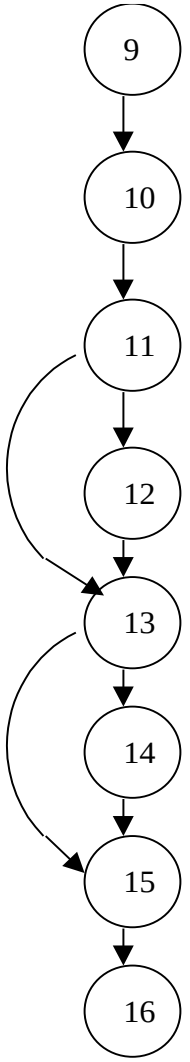
Telkom University

BASIS PATH TESTING: EXAMPLE 1

1. Draw a flow graph

```
0  /*Finding the maximum of three integers*/
1  #include <stdio.h>
2  intmaximum(int, int, int);
3  main(){
4     int a,b,c;
5     printf((“Enter three integers: “);
6     scanf(“%d%d%d”, &a,&b,&c);
7     printf(“Maximum is: %d\n”, maximum(a,b,c));
8  }
9  int maximum(int x, int y, int z){
10    int max=x;
11    if(y>max)
12       max=y;
13    if(z>max)
14       max=z;
15    return max
16 }
```

==>



Telkom University

BASIS PATH TESTING: EXAMPLE 1

2. Calculate cyclomatic complexity
   - e=7, n=6, p=1; so that cc=7-6+2=3

3. Identify a basis set of independent paths
   - p1: a-b-c-d-e-f (y>x, z>y)
   - p2: a-b-d-e-f (y<=x, z>x)
   - p3: a-b-c-d-f (y>x, z<=y)

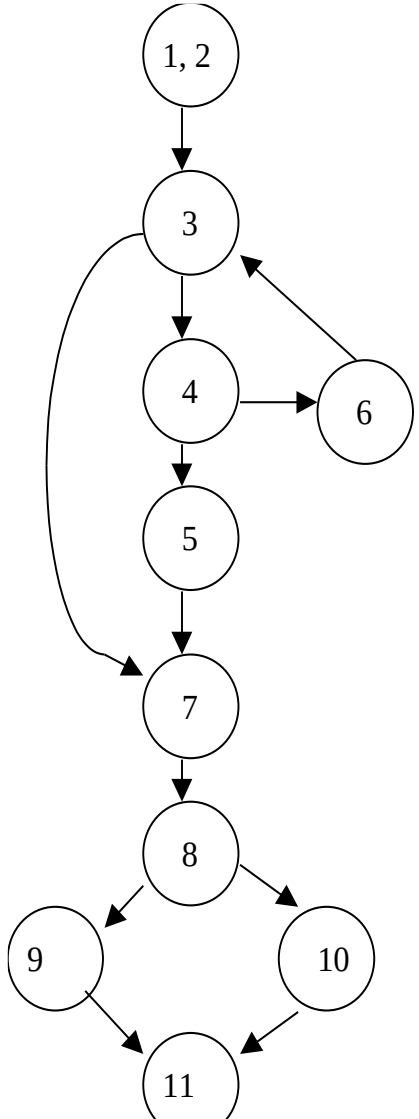
4. Design test cases

Test case	Test data (x,y,z)	Expected result	Actual result	Pass/fail
p1	2, 3, 4	4		
p2	3, 1, 6	6		
p3	5, 7, 3	7		

Telkom University

BASIS PATH TESTING: EXAMPLE 2

1. Draw a flow graph
   - see slide 6-24: source code, flow graph
2. Calculate cyclomatic complexity
   - e = 12; n = 10; p = 1
   - cc = 12 - 10 + 2 x 1 =4
3. Determine a basis set of independent paths
   - expect to specify 4 independent paths
   - p1: *1-2-3-7-8-9-11*
   - p2: 1-2-3-*4-5-7*-8-9-11
   - p3: 1-2-3-4-5-*7*-8-*10*-11
   - p4: 1-2-3-4-6-3-*7*-8-10-11 (1 or more times)
   - HOWEVER: by reading source code, we found
     - 3-*7* => 10; *5* => 9
     - p1 and p3 must be modified



Telkom University

BASIS PATH TESTING: EXAMPLE 2

3. Determine a basis set of independent paths

- if p3 modified, it would be the same as p2. Thus p3 should be deleted.
- But the new paths introduced by p3 (8-10-11) must be covered by other paths! We found p4 covers them.
- Modify p1, delete p3, we can have three independent paths
  - p1: 1-2-3-7-8-10-11
  - p2: 1-2-3-4-5-7-8-9-11
  - p3: 1-2-3-4-6-3-7-8-10-11
- if you study the program carefully, you will find the following is better
  - p1: 1-2-3-7-8-10-11 (insert x when a[] is empty)
  - p2: 1-2-3-4-5-7-8-9-11(insert x when a[1]=x)
  - p3: 1-2-3-4-6-3-4-5-7-8-9-11 (insert x when a[i]=x,i>1, n>=i)
  - p4: 1-2-3-4-6-3-7-8-10-11 (insert x when a[] is not empty and x is not in a[]; p4 does not introduce any new edge but it exercises a new combination of the program logic!)

Telkom University

BASIS PATH TESTING: EXAMPLE 2

4. Design test cases

- Path 1 test case: 1-2-3-7-8-10-11 (insert x when a[] is empty)
  - input data: n=0; x=8; a[1]=0; b[1]=0;
  - expected results: a[1]=8; b[1]=1; n=1;
- Path 2 test case: 1-2-3-4-5-7-8-9-11(insert x when a[1]=x)
  - input data: n=3; x=9; a[1]=9; a[2]=2; a[3]=3;b[1]=2;b[2]=5;b[3]=8;
  - expected results: b[1]=3
- Path 3 test case: 1-2-3-4-6-3-4-5-7-8-9-11 (insert x when a[i]=x,i>1, n>=i)
  - input data: n=3;x=3;a[1]=9;a[2]=2;a[3]=3;b[1]=3;b[2]=2;b[3]=8;
  - expected results: b[3]=9
- Path 4 test case: 1-2-3-4-6-3-7-8-10-11 (insert x when a[] is not empty and x is not in a[])
  - input data: n=3;x=6;a[1]=9;a[2]=2;a[3]=3;b[1]=3;b[2]=2;b[3]=8;
  - expected results: a[4]=6; b[4]=1; n=4;

Telkom University

LATIHAN

```
i = 1;
total.input = total.valid = 0;
sum = 0;
DO WHILE value[i] <> -999 AND total.input < 100
      increment total.input by 1;
            IF value[i] >= minimum and value[i] <= maximum
                  THEN increment total.valid by 1;
                        sum = sum + value[i]
                  ELSE skip
            ENDIF
            increment I by 1;
ENDDO

IF total.valid > 0
      THEN average = sum / total.valid;
      ELSE average = -999;
ENDIF

END average
```

Telkom University

LATIHAN

1. Buatlah *Flowgrap* dari potongan badan program di atas!
2. Tentukan:
   a) Berapa jumlah *predicate node*
   b) Berapa nilai V(G)
3. Tuliskan berapa *independent path* dari *flowgrap* di atas!
4. Desainlah *basis path test cases*-nya (lihat contoh di materi)!

Kumpulkan sebagai tugas individu, kirim melalui email dengan subject basis_path maksimal hari ini, Senin 12 Maret 2018 pukul 08.00 WIB

REFERENCES

- http://softwaretestingfundamentals.com
- Pressmann, R.S (2010). Software Engineering A Practitioner's approach. New York: McGraw-Hill.
- Agus Pratondo, d. (2009). Jaminan Mutu Sistem Informasi. Bandung: Politeknik Telkom.
- Eka Widhi Yunarso (2013). Student Workbook Jaminan Mutu Sistem Informasi. Yogyakarta: DeePublished

Telkom University