آزمون نرم افزار – بخش۲–۳

پوشش گراف برای سورس کد

صدیقه خوشنویس
دانشگاه آزاد اسلامی – واحد شهرقدس

یادآوری

- رایج ترین کاربرد گراف، برای آزمایش سورس کد برنامه است.

- گراف: معمولاً گراف جریان کنترل (CFG = Control Flow Graph)
- پوشش گره: اجرای همه عبارت های برنامه
- پوشش یال: اجرای همه انشعاب های برنامه
- حلقه ها: اجرای همه حلقه ها مانند حلقه های while، for و ...
- پوشش جریان داده (DU): بهبود CFG
  - تعاریف (def): عبارتهایی که مقداری را به یک متغیر منتسب می کنند
  - کاربردها(use): عبارتهایی که از متغیرها استفاده می کنند

گراف جریان کنترل (CFG)

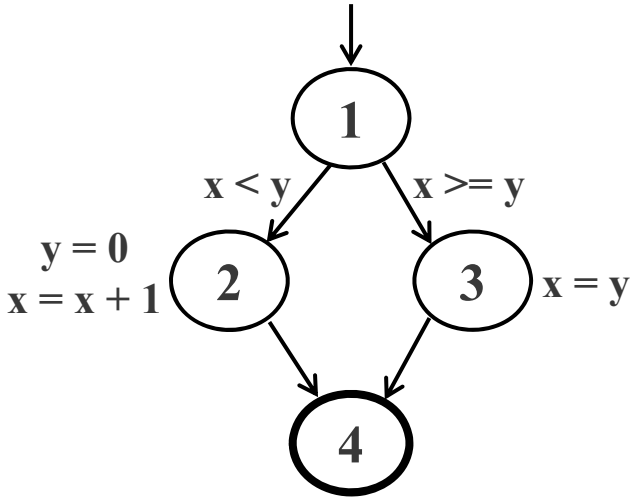
- یک گراف جریان کنترل (CFG) همه اجراهای یک قطعه کد را با توصیف ساختارهای کنترلی آن مدل سازی می کند (نمایش مسیرهای اجرای برنامه)

- گره ها: عبارت یا دنباله ای از عبارتها (یا بلوک پایه)
- یال ها: انتقال کنترل (=ترتیب اجرا)
- بلوکهای پایه: دنباله ای از عبارت ها به طوری که اگر عبارت اول اجرا شود همه عبارتهای بعدی اجرا شوند (مجموعه ای از عبارت ها که درون آنها انشعاب وجود نداشته باشد)

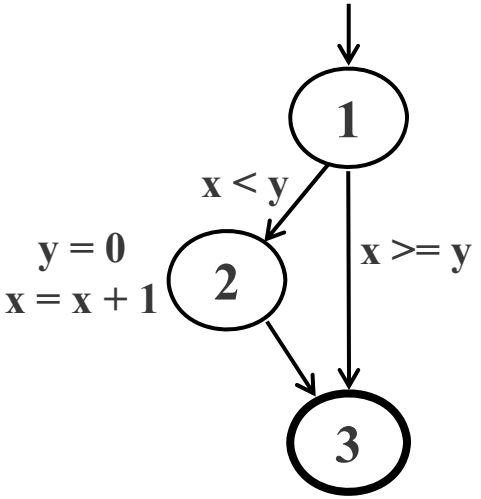
- CFGها گاهی با اطلاعات جانبی حاشیه نویسی می شوند
  - گزاره منطقی شرط ها
  - def ها و useها
- قواعد تبدیل عبارت ها به گراف چیست؟

CFG : دستور if

```
if (x < y)
{
   y = 0;
   x = x + 1;
}
else
{
   x = y;
}
```

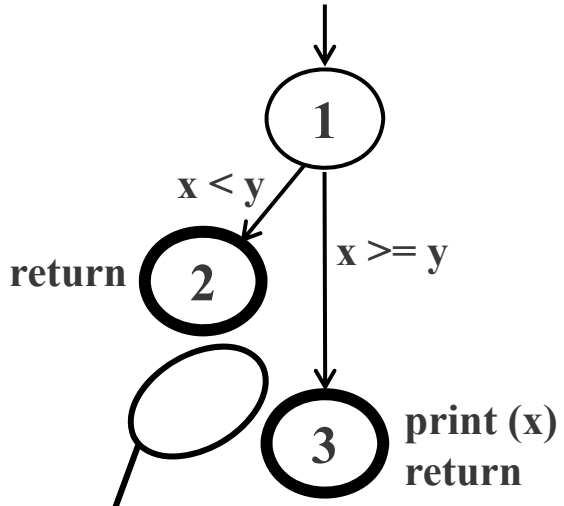



```
if (x < y)
{
   y = 0;
   x = x + 1;
}
```



CFG : دستور if همراه return

```
if (x < y)
{
   return;
}
print (x);
return;
```

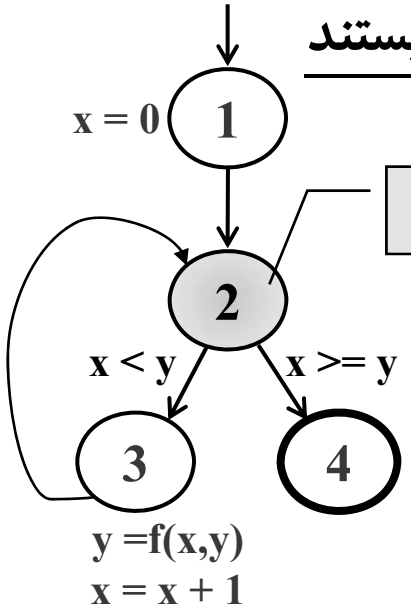


یالی از گره ۲ به ۳ وجود ندارد
گره return باید مجزا و گره نهایی باشد

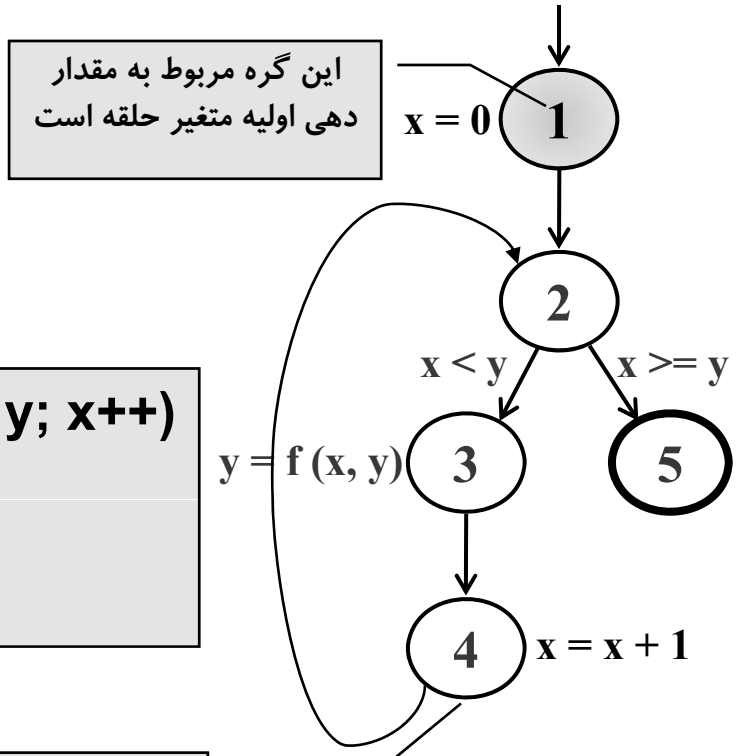
حلقه ها در CFG

- برای نمایش حلقه ها در گراف نیاز به گره های مجزایی داریم
- که نشان دهنده عبارت ها یا بلوک های پایه نیستند

```
x = 0;
while (x < y)
{
   y = f (x, y);
   x = x + 1;
}
```

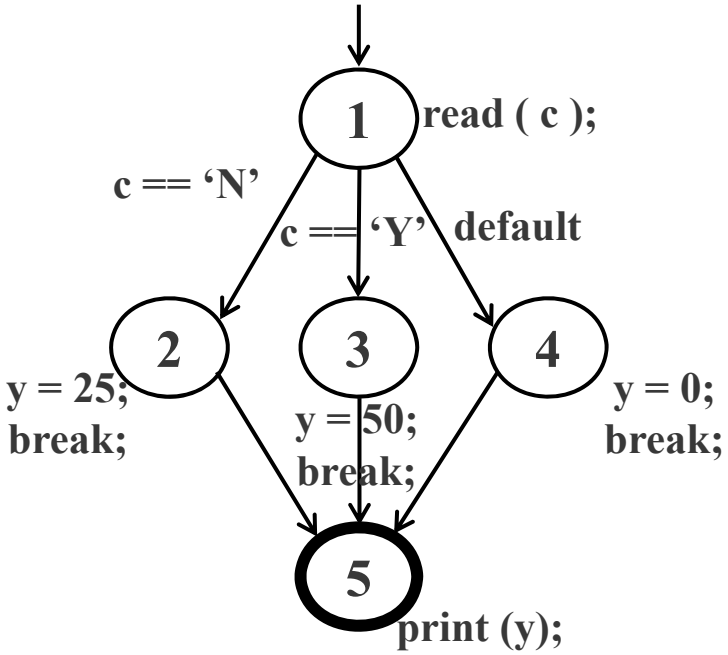



```
for (x = 0; x < y; x++)
{
   y = f (x, y);
}
```



ساختار switch-case در CFG

```
read ( c) ;
switch ( c )
{
  case ‘N’:
    y = 25;
    break;
  case ‘Y’:
    y = 50;
    break;
  default:
    y = 0;
    break;
}
print (y);
```

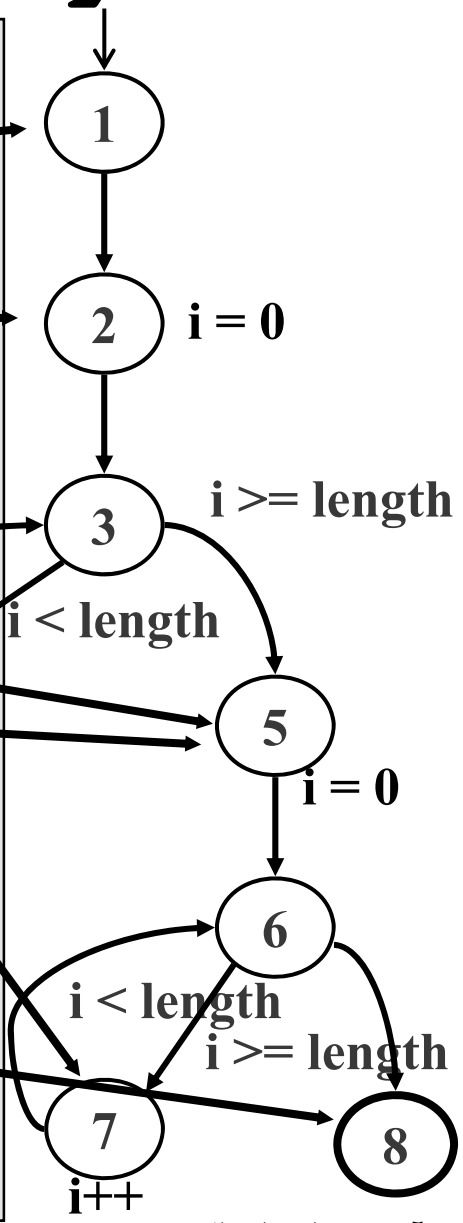


مثال – برنامه Stats

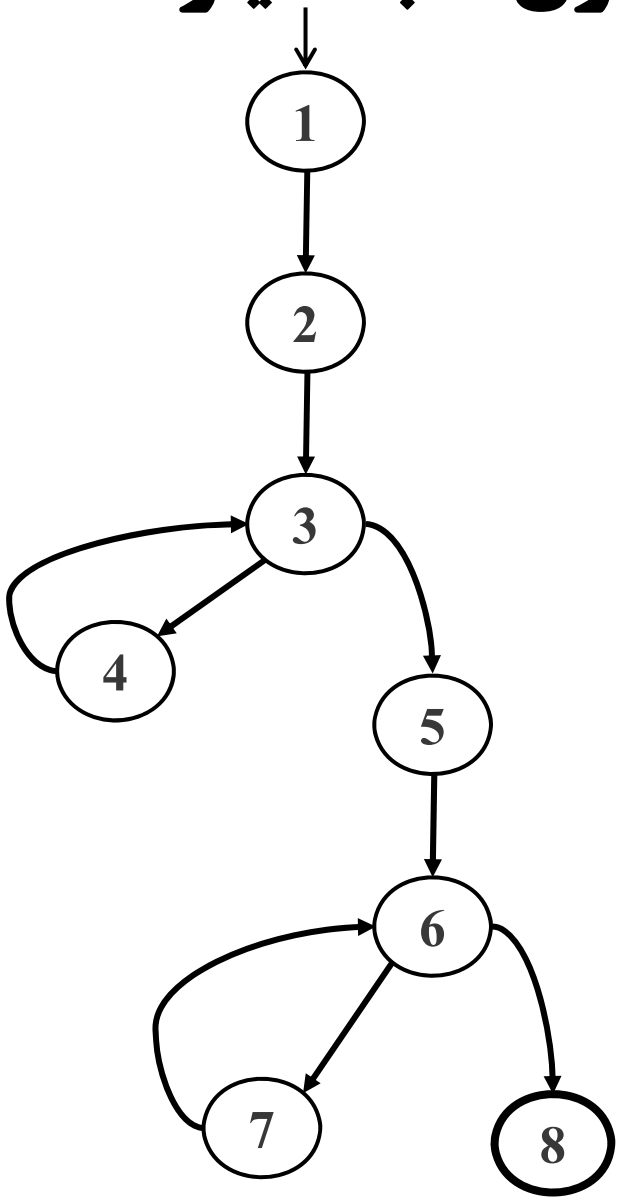
```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, mean, sum, varsum;

    sum = 0;
    for (int i = 0; i < length; i++)
    {
        sum += numbers [ i ];
    }
    med   = numbers [ length / 2 ];
    mean = sum / (double) length;

    varsum = 0;
    for (int i = 0; i < length; i++)
    {
        varsum = varsum  + ((numbers [ I ] - mean) * (numbers [ I ] - mean));
    }
    var = varsum / ( length - 1.0 );
    sd  = Math.sqrt ( var );

    System.out.println ("length:                " + length);
    System.out.println ("mean:                  " + mean);
    System.out.println ("median:                " + med);
    System.out.println ("variance:              " + var);
    System.out.println ("standard deviation: " + sd);
}
```

گراف CFG برای مثال (برنامه Stats)

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, mean, sum, varsum;

    sum = 0;
    for (int i = 0; i < length; i++)
    {
        sum += numbers [ i ];
    }
    med   = numbers [ length / 2 ];
    mean = sum / (double) length;

    varsum = 0;
    for (int i = 0; i < length; i++)
    {
        varsum = varsum  + ((numbers [ i ] - mean) * (numbers [ i ] - mean));
    }
    var = varsum / ( length - 1.0 );
    sd  = Math.sqrt ( var );

    System.out.println ("length:                " + length);
    System.out.println ("mean:                  " + mean);
    System.out.println ("median:                " + med);
    System.out.println ("variance:              " + var);
    System.out.println ("standard deviation: " + sd);
}
```

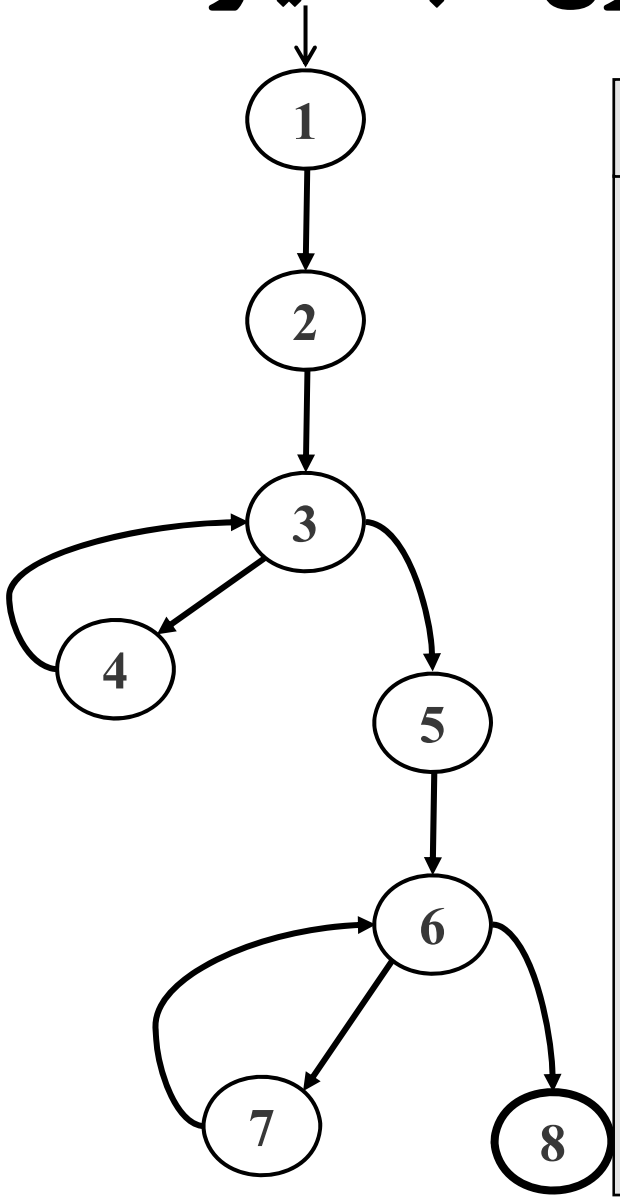


نیازمندیهای آزمون و مسیرهای آزمون – با معیار EC



پوشش یال (EC)	
TR	مسیر آزمون(TP)
A. [1, 2]	[1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
B. [2, 3]	
C. [3, 4]	
D. [3, 5]	
E. [4, 3]	
F. [5, 6]	
G. [6, 7]	
H. [6, 8]	
I. [7, 6]	

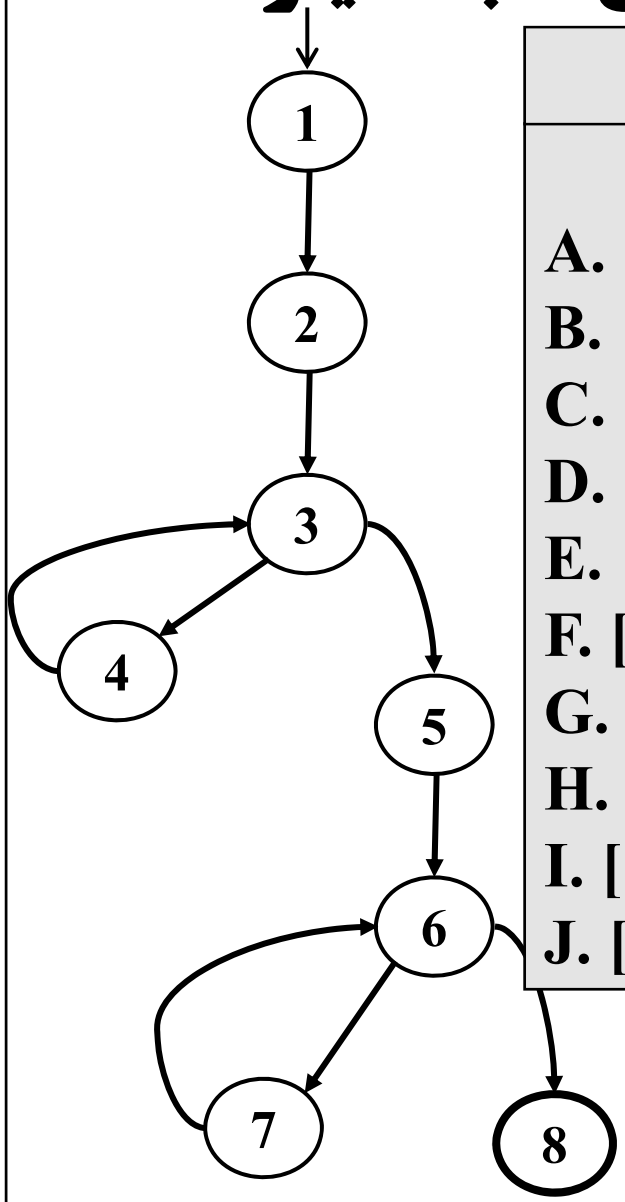
نیازمندیهای آزمون و مسیرهای آزمون – با معیار EPC



پوشش زوج یال (EPC)	
TR	مسیر آزمون (TP)
A. [1, 2, 3] B. [2, 3, 4] C. [2, 3, 5] D. [3, 4, 3] E. [3, 5, 6] F. [4, 3, 5] G. [5, 6, 7] H. [5, 6, 8] I. [6, 7, 6] J. [7, 6, 8] K. [4, 3, 4] L. [7, 6, 7]	i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8] ii. [1, 2, 3, 5, 6, 8] iii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]

TP	TR های پیمایش شده
i	A, B, D, E, F, G, I, J
ii	A, C, E, H
iii	A, B, D, E, F, G, I, J, K, L

نیازمندیهای آزمون و مسیرهای آزمون – با معیار PPC



پوشش مسیر اصلی (PPC)

TR
A. [3, 4, 3]
B. [4, 3, 4]
C. [7, 6, 7]
D. [7, 6, 8]
E. [6, 7, 6]
F. [1, 2, 3, 4]
G. [4, 3, 5, 6, 7]
H. [4, 3, 5, 6, 8]
I. [1, 2, 3, 5, 6, 7]
J. [1, 2, 3, 5, 6, 8]

مسیرهای آزمون (TP)

i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
ii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]
iii. [1, 2, 3, 4, 3, 5, 6, 8]
iv. [1, 2, 3, 5, 6, 7, 6, 8]
v. [1, 2, 3, 5, 6, 8]

TP	TRهای پیمایش شده
i	A, D, E, F, G
ii	A, B, C, D, E, F, G,
iii	A, F, H
iv	D, E, F, I
v	J

پوشش جریان داده (DU) برای سورس کد

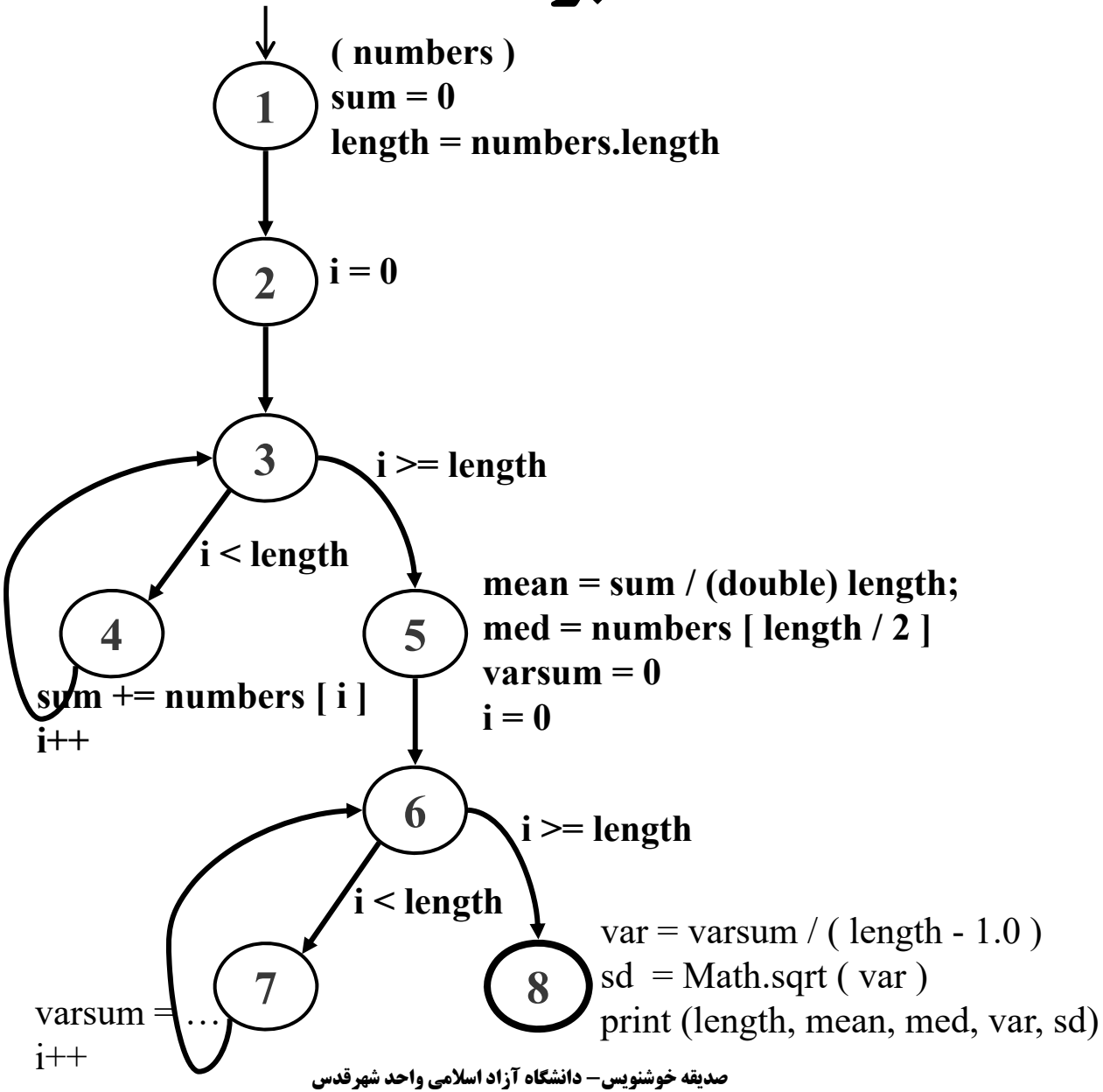
- **def**: محلی از برنامه که در آن مقدار متغیری در حافظه ذخیره (نوشته) می شود
  - x در سمت چپ یک انتساب واقع شده باشد (x=44;)
  - x پارامتر واقعی (آرگومان) در فراخوانی باشد و متد مقدار آن را تغییر دهد
  - x پارامتر رسمی یک متد باشد (وقتی اجرای متد آغاز می شود به طور ضمنی def می شود)
  - x ورودی یک برنامه باشد
- **use**: محلی از برنامه که در آن مقدار متغیری مورد دسترسی (خواندن) قرار می گیرد .
  - x در سمت راست یک انتساب واقع شده باشد
  - x در یک شرط ظاهر شده باشد
  - x پارامتر واقعی (آرگومان) متد باشد
  - x خروجی برنامه باشد
  - x خروجی متد در دستور return باشد

- اگر در یک دستور، هم def و هم use وجود داشته باشد، آنگاه یک زوج DU خواهد بود اگر def بعد از use باشد و آن گره در یک حلقه باشد

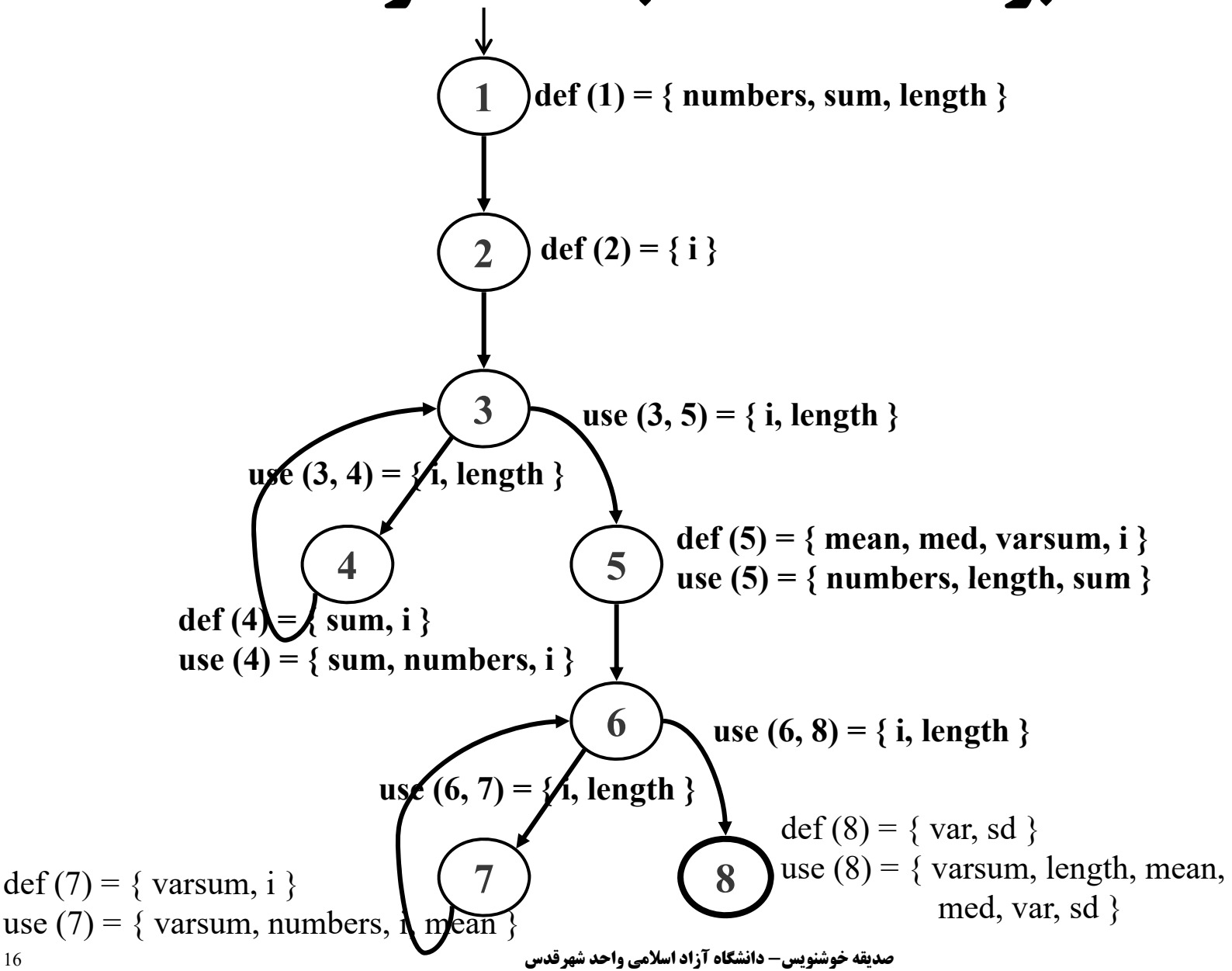
مثال از پوشش DU – برنامه Stats

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, mean, sum, varsum;

    sum = 0;
    for (int i = 0; i < length; i++)
    {
        sum += numbers [ i ];
    }
    mean = sum / (double) length;
    med   = numbers [ length / 2 ];

    varsum = 0;
    for (int i = 0; i < length; i++)
    {
        varsum = varsum  + ((numbers [ i ] - mean) * (numbers [ i ] - mean));
    }
    var = varsum / ( length - 1.0 );
    sd  = Math.sqrt ( var );

    System.out.println ("length:                " + length);
    System.out.println ("mean:                  " + mean);
    System.out.println ("median:                " + med);
    System.out.println ("variance:              " + var);
    System.out.println ("standard deviation: " + sd);
}
```

CFG برنامه Stats



CFG برنامه Stats – با def ها و use ها



جدول def ها و use ها برای Stats

گره	Def	Use
1	{ numbers, sum, length }	
2	{ i }	
3		
4	{ sum, i }	{ numbers, i, sum }
5	{ mean, med, varsum, i }	{ numbers, length, sum }
6		
7	{ varsum, i }	{ varsum, numbers, i, mean }
8	{ var, sd }	{ varsum, length, var, mean, med, var, sd }

یال	Use
(1, 2)	
(2, 3)	
(3, 4)	{ i, length }
(4, 3)	
(3, 5)	{ i, length }
(5, 6)	
(6, 7)	{ i, length }
(7, 6)	
(6, 8)	{ i, length }

زوج های DU برای Stats

متغیر	زوجهای DU
numbers	(1, 4) (1, 5) (1, 7)
length	(1, 5) (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))
med	(5, 8)
var	(8, 8)
sd	(8, 8)
mean	(5, 7) (5, 8)
sum	(1, 4) (1, 5) (4, 4) (4, 5)
varsum	(5, 7) (5, 8) (7, 7) (7, 8)
i	(2, 4) (2, (3,4)) (2, (3,5)) ~~(2, 7) (2, (6,7)) (2, (6,8))~~
	(4, 4) (4, (3,4)) (4, (3,5)) ~~(4, 7) (4, (6,7)) (4, (6,8))~~
	(5, 7) (5, (6,7)) (5, (6,8))
	(7, 7) (7, (6,7)) (7, (6,8))

def ها قبل از useها آمده اند، پس زوج DU حساب نمی شود

def ها بعد از use و در حلقه آمده اند. پس زوجهای DU معتبر هستند

مسیر، def-clear نیست حیطه i متفاوت است

هیچ مسیری در گراف از گره های ۵ و ۷ به گره های ۴ و ۳ وجود ندارد

مسیرهای DU برای Stats

متغیر	زوج های DU	مسیرهای DU
numbers	(1, 4) (1, 5) (1, 7)	[1, 2, 3, 4] [1, 2, 3, 5] [1, 2, 3, 5, 6, 7]
length	(1, 5) (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))	[1, 2, 3, 5] [1, 2, 3, 5, 6, 8] [1, 2, 3, 4] [1, 2, 3, 5] [1, 2, 3, 5, 6, 7] [1, 2, 3, 5, 6, 8]
med	(5, 8)	[5, 6, 8]
var	(8, 8)	*هیچ مسیری نیاز نیست*
sd	(8, 8)	*هیچ مسیری نیاز نیست*
sum	(1, 4) (1, 5) (4, 4) (4, 5)	[1, 2, 3, 4] [1, 2, 3, 5] [4, 3, 4] [4, 3, 5]

متغیر	زوج های DU	مسیرهای DU
mean	(5, 7) (5, 8)	[5, 6, 7] [5, 6, 8]
varsum	(5, 7) (5, 8) (7, 7) (7, 8)	[5, 6, 7] [5, 6, 8] [7, 6, 7] [7, 6, 8]
i	(2, 4) (2, (3,4)) (2, (3,5)) (4, 4) (4, (3,4)) (4, (3,5)) (5, 7) (5, (6,7)) (5, (6,8)) (7, 7) (7, (6,7)) (7, (6,8))	[2, 3, 4] [2, 3, 4] [2, 3, 5] [4, 3, 4] [4, 3, 4] [4, 3, 5] [5, 6, 7] [5, 6, 7] [5, 6, 8] [7, 6, 7] [7, 6, 7] [7, 6, 8]

مسیرهای DU برای Stats – بدون تکراری ها

۳۸ مسیر DU برای Stats وجود دارد ولی تنها ۱۲ تای آنها یکتا هستند (بقیه تکراری اند)

[1, 2, 3, 4]	[4, 3, 4]
[1, 2, 3, 5]	[4, 3, 5]
[1, 2, 3, 5, 6, 7]	[5, 6, 7]
[1, 2, 3, 5, 6, 8]	[5, 6, 8]
[2, 3, 4]	[7, 6, 7]
[2, 3, 5]	[7, 6, 8]

☆ ۵ تا از این مسیرها، نیازمند عدم عبور از حلقه هستند

✦ ۵ تا از این مسیرها نیازمند یک بار عبور از حلقه هستند

۲ تا از این مسیرها نیازمند دو بار عبور از حلقه هستند

موارد آزمون و مسیرهای آزمون

مورد آزمون: numbers = (44) ; length = 1

مسیر آزمون: [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]

مسیرهای DU که پوشش داده می شوند:

[1, 2, 3, 4] [2, 3, 4] [4, 3, 5] [5, 6, 7] [7, 6, 8]

اینها همان ۵ مسیری هستند که نیازمند یک بار عبور از حلقه اند ✧

مورد آزمون: numbers = (2, 10, 15) ; length = 3

مسیر آزمون: [1, 2, 3, 4, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 7, 6, 8]

مسیرهای DU که پوشش داده می شوند:

[4, 3, 4] [7, 6, 7]

اینها همان ۲مسیری هستند که نیازمند دو بار عبور از حلقه اند ✡

سایر مسیرهای DU (☆) نیازمند آن هستند که آرایه هایی با طول صفر، از حلقه عبور نکنند؛ اما متد با خطای تقسیم بر صفر روی دستور زیر دچار خرابی می شود:

```
mean = sum / (double) length;
```

یک نقص پیدا شد

مثال: TestPat

```
public int pat (char[] subject, char[] pattern)
{
// Post: if pattern is not a substring of subject,
      return -1
//       else return (zero-based) index where the
      pattern (first)
//       starts in subject
   final int NOTFOUND = -1;
   int  iSub = 0, rtnIndex = NOTFOUND;
   boolean isPat  = false;
   int subjectLen = subject.length;
   int patternLen = pattern.length;
   while (isPat == false && iSub + patternLen - 1 <
       subjectLen)
    {
      if (subject [iSub] == pattern [0])
      {
        rtnIndex = iSub; // Starting at zero
        isPat = true;
        for (int iPat = 1; iPat < patternLen; iPat ++)
        {
          if (subject[iSub + iPat] != pattern[iPat])
          {
            rtnIndex = NOTFOUND;
            isPat = false;
            break;  // out of for loop
          }
        }
      }
      iSub ++;
    }
    return (rtnIndex);
  }
  }
```

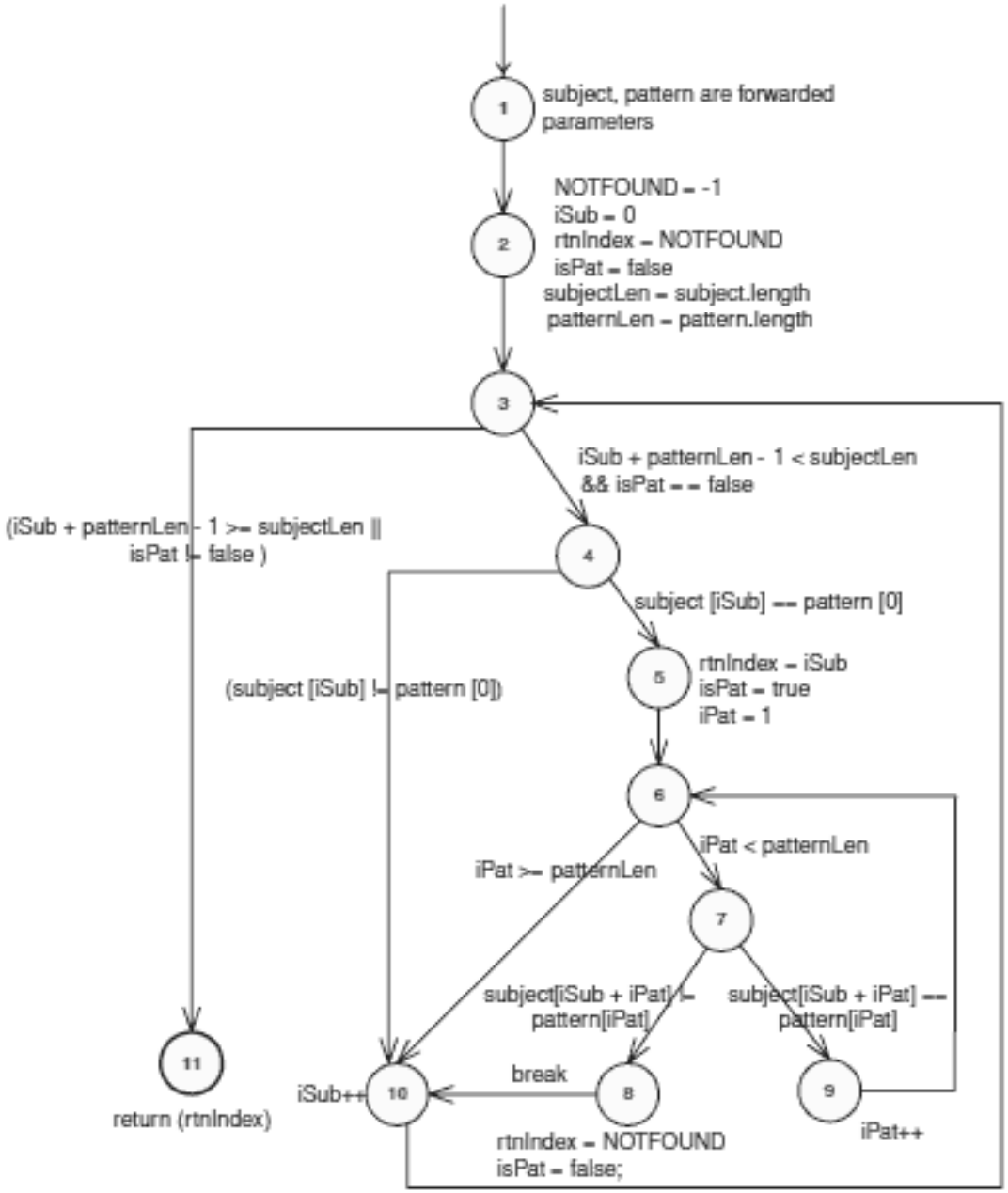


Figure 2.12. A graph showing an example of du-paths.

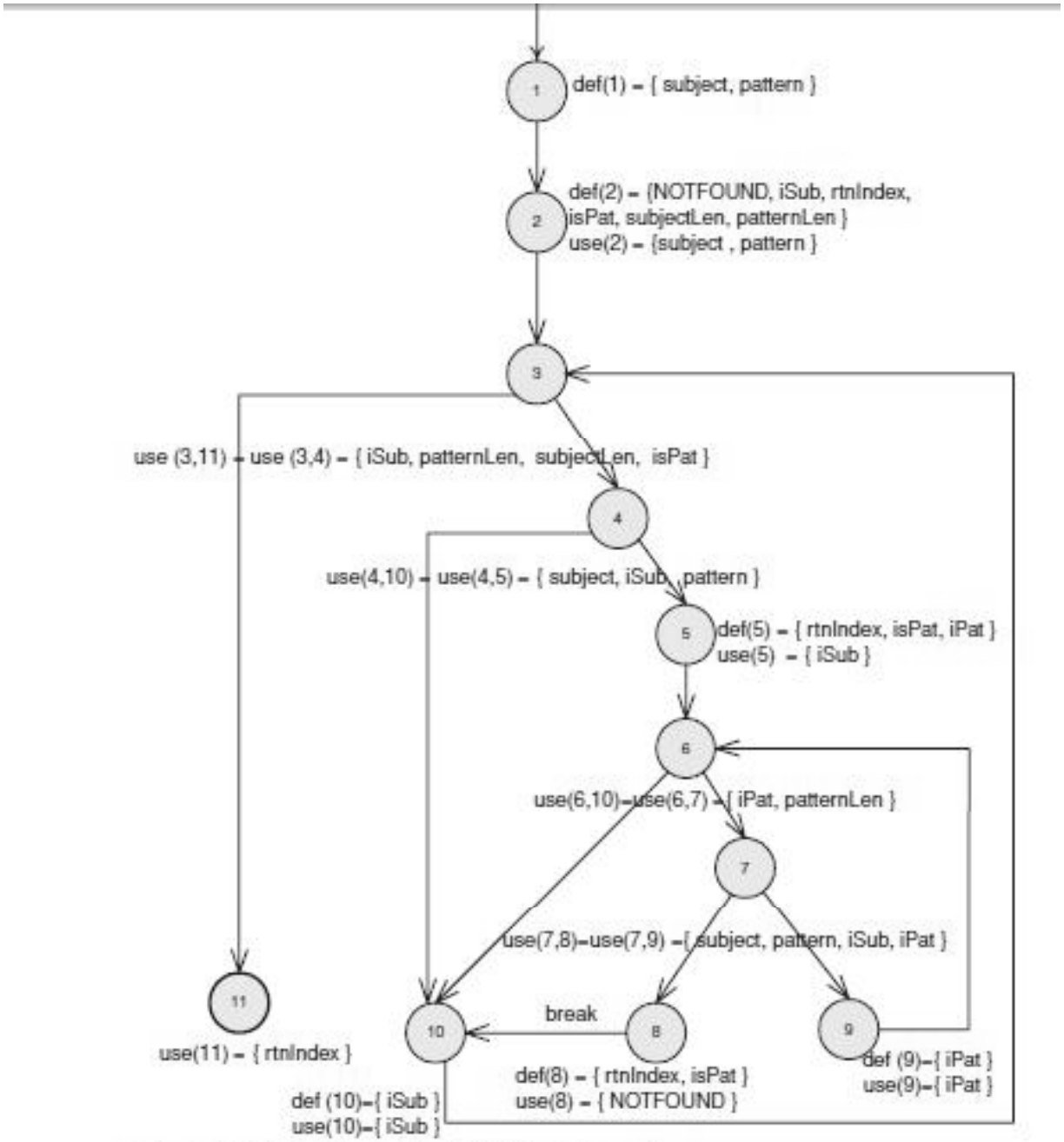


Figure 2.13. Graph showing explicit def and use sets.

Table 2.1. Defs and uses at each node in the CFG for TestPat

node	def	use
1	{subject, pattern}	
2	{NOTFOUND, isPat, iSub, rtnIndex, subjectLen, patternLen}	{subject, pattern}
3		
4		
5	{rtnIndex, isPat, iPat}	{iSub}
6		
7		
8	{rtnIndex, isPat}	{NOTFOUND}
9	{iPat}	{iPat}
10	{iSub}	{iSub}
11		{rtnIndex}

Table 2.2. Defs and uses at each edge in the CFG for TestPat.

edge	use
(1, 2)	
(2, 3)	
(3, 4)	{iSub, patternLen, subjectLen, isPat}
(3, 11)	{iSub, patternLen, subjectLen, isPat}
(4, 5)	{subject, iSub, pattern}
(4, 10)	{subject, iSub, pattern}
(5, 6)	
(6, 7)	{iPat, patternLen}
(6, 10)	{iPat, patternLen}
(7, 8)	{subject, iSub, iPat, pattern}
(7, 9)	{subject, iSub, iPat, pattern}
(8, 10)	
(9, 6)	
(10, 3)	

Table 2.3. Du-path sets for each variable in TestPat

variable	du-path set	du-paths	prefix?
NOTFOUND	du (2, NOTFOUND)	[2,3,4,5,6,7,8]	
rtnIndex	du (2, rtnIndex)	[2,3,11]	
	du (5, rtnIndex)	[5,6,10,3,11]	
	du (8, rtnIndex)	[8,10,3,11]	
iSub	du (2, iSub)	[2,3,4]	Yes
		[2,3,4,5]	Yes
		[2,3,4,5,6,7,8]	Yes
		[2,3,4,5,6,7,9]	
		[2,3,4,5,6,10]	
		[2,3,4,5,6,7,8,10]	
		[2,3,4,10]	
		[2,3,11]	
	du (10, iSub)	[10,3,4]	Yes
		[10,3,4,5]	Yes
		[10,3,4,5,6,7,8]	Yes
		[10,3,4,5,6,7,9]	
		[10,3,4,5,6,10]	
		[10,3,4,5,6,7,8,10]	
		[10,3,4,10]	
		[10,3,11]	
iPat	du (5, iPat)	[5,6,7]	Yes
		[5,6,10]	
		[5,6,7,8]	
		[5,6,7,9]	
	du (9, iPat)	[9,6,7]	Yes
		[9,6,10]	
		[9,6,7,8]	
		[9,6,7,9]	
isPat	du (2, isPat)	[2,3,4]	
		[2,3,11]	
	du (5, isPat)	[5,6,10,3,4]	
		[5,6,10,3,11]	
	du (8, isPat)	[8,10,3,4]	
		[8,10,3,11]	
subject	du (1, subject)	[1,2]	Yes
		[1,2,3,4,5]	Yes
		[1,2,3,4,10]	
		[1,2,3,4,5,6,7,8]	
		[1,2,3,4,5,6,7,9]	
pattern	du (1, pattern)	[1,2]	Yes
		[1,2,3,4,5]	Yes
		[1,2,3,4,10]	
		[1,2,3,4,5,6,7,8]	
		[1,2,3,4,5,6,7,9]	
subjectLen	du (2, subjectLen)	[2,3,4]	
		[2,3,11]	
patternLen	du (2, patternLen)	[2,3,4]	Yes
		[2,3,11]	
		[2,3,4,5,6,7]	
		[2,3,4,5,6,10]	

Table 2.4. Test paths to satisfy all du-paths coverage on TestPat

test case (subject,pattern,output)	test path(t)
(a, bc, −1)	[1,2,3,11]
(ab, a, 0)	[1,2,3,4,5,6,10,3,11]
(ab, ab, 0)	[1,2,3,4,5,6,7,9,6,10,3,11]
(ab, ac, −1)	[1,2,3,4,5,6,7,8,10,3,11]
(ab, b, 1)	[1,2,3,4,10,3,4,5,6,10,3,11]
(ab, c, −1)	[1,2,3,4,10,3,4,10,3,11]
(abc, abc, 0)	[1,2,3,4,5,6,7,9,6,7,9,6,10,3,11]
(abc, abd, −1)	[1,2,3,4,5,6,7,9,6,7,8,10,3,11]
(abc, ac −1)	[1,2,3,4,5,6,7,8,10,3,4,10,3,11]
(abc, ba, −1)	[1,2,3,4,10,3,4,5,6,7,8,10,3,11]
(abc, bc, 1)	[1,2,3,4,10,3,4,5,6,7,9,6,10,3,11]

Table 2.5. Test paths and du-paths covered on TestPat.

test case (subject,pattern, output)	test path(t)	du-path toured
(ab, ac, −1)	[1,2,3,4,5,6,7,8,10,3,11]	[2,3,4,5,6,7,8](NOTFOUND)
(a, bc, −1)	[1,2,3,11]	[2,3,11](rtnIndex)
(ab, a, 0)	[1,2,3,4,5,6,10,3,11]	[5,6,10,3,11](rtnIndex)
(ab, ac, −1)	[1,2,3,4,5,6,7,8,10,3,11]	[8,10,3,11](rtnIndex)
(ab, ab, 0)	[1,2,3,4,5,6,7,9,6,10,3,11]	[2,3,4,5,6,7,9] (iSub)
(ab, a, 0)	[1,2,3,4,5,6,10,3,11]	[2,3,4,5,6,10](iSub)
(ab, ac, −1)	[1,2,3,4,5,6,7,8,10,3,11]	[2,3,4,5,6,7,8,10](iSub)
(ab, c, −1)	[1,2,3,4,10,3,4,10,3,11]	[2,3,4,10](iSub)
(a, bc, −1)	[1,2,3,11]	[2,3,11] (iSub)
(abc, bc, 1)	[1,2,3,4,10,3,4,5,6,7,9,6,10,3,11]	[10,3,4,5,6,7,9](iSub)
(ab, b, 1)	[1,2,3,4,10,3,4,5,6,10,3,11]	[10,3,4,5,6,10](iSub)
(abc, ba, −1)	[1,2,3,4,10,3,4,5,6,7,8,10,3,11]	[10,3,4,5,6,7,8,10](iSub)
(ab, c, −1)	[1,2,3,4,10,3,4,10,3,11]	[10,3,4,10](iSub)

پایان جلسه سوم