Testiranje softvera

ETF BEOGRAD, 2019/2020.

VEŽBE #6, ASISTENT: DR DRAŽEN DRAŠKOVIĆ

Testiranje strategijama bele kutije

WHITE BOX TESTING

Tehnike zasnovano na programskim putanjama

- Test primeri se projektuju tako da se zadovolji neki od kriterijuma pokrivenosti putanja u programu.
- Vrste testiranja:
  - Testiranje potpunim pokrivanjem putanja
  - Pokrivanje linearno nezavisnih putanja
  - Testiranje petlji
  - Granično testiranje unutrašnje petlje
  - Pokrivanje LCSAJ sekvenci
- Definicija programske putanje zasniva se na grafu toka kontrole programa (eng. *Control Flow Graph*, skraćeno *CFG*).

Zadatak 1 - Testiranje ugneždene WHILE petlje

- U sledećem programu za sortiranje testirati sve petlje Bajzerovom heurističkom tehnikom:

```
S1      i := 2
C1      while (i is less than or equal to n) do
S2          j := i - 1
C2          while ((j is greater than or equal to 1) and
                    (A[j] is greater than A[j+1])) do
S3              temp := A[j]
S4              A[j] := A[j+1]
S5              A[j+1] := temp
S6              j := j-1
            end while
S7          i := i + 1
        end while
```

Zadatak 1 - Testiranje ugneždene WHILE petlje - Rešenje (1)

- Možemo da nacrtamo dijagram toka. Postoje 2 **while-do** petlje.
- Maksimalni broj iteracija unutrašnje petlje je za jedan manji od trenutne vrednosti promenljive *i*. Promenljiva *j* dekrementira se na kraju jednog ciklusa i petlja se završava kada vrednost promenljive *j* postane jednaka 0.
- Minimalni broj iteracija unutrašnje petlje je 0, a pošto se petlja može završiti trenutno (i to se dešava ukoliko element A[i-1] ima vrednost koja je manja ili jednaka vrednosti elementa A[i]).
- Kada započne izvršavanje unutrašnje petlje, za datu vrednost *i*, element na poziciji A[i] niza je element koji je na toj poziciji bio na početku izvršavanja, dok su elementi na pozicijama A[1], A[2],…, A[i-1] preuređeni, tako da se u nizu pojavljuju u rastućem redosledu.
- Određivanje broja izvršavanja za spoljašnju petlju znatno je jednostavnije: ova petlja će uvek biti izvršena tačno *n-1* puta, ukoliko je *n* veličina ulaznog niza.

Zadatak 1 - Testiranje ugneždene WHILE petlje - Rešenje (2)

- Test #1 koji će obezbediti da se unutrašnja petlja izvrši dva puta je sledeći:
  - Ulaz: n = 3; A[1] = 3, A[2] = 2, A[3] = 1
  - Očekivani izlaz: A[1] = 1, A[2] = 2, A[3] = 3
- Test #2 koji će unutrašnju petlju izvršiti 50. puta, 98. puta, 99. puta i 100. puta. Broj iteracija spoljašnje petlje čuvamo na najmanjoj mogućoj vrednosti, koristićemo testove gde su dužine nizova 51, 99, 100 i 101 respektivno, tako da je najmanji element na kraju ulaznog niza:
  - Ulaz: n = 51, A[i], gde je *i* između 1 i 50, i vrednost A[51] = 0
  - Očekivani izlaz: A[i] = [i-1] za *i* između 1 i 51

Zadatak 1 - Testiranje ugneždene WHILE petlje - Rešenje (3)

- Test #3

  Ulaz: n = 99

  - A[i] = 1, ako je *i* između 1 i 98, i ako je *i* paran broj;
  - A[i] = 2, ako je *i* između 1 i 98, i ako je *i* neparan broj;
  - A[99] = 0

  Očekivani izlaz:

  - A[1] = 0,
  - A[i] = 1, ako je *i* između 2 i 50,
  - A[i] = 2 , ako je *i* između 51 i 99.

- Test #4

  Ulaz: n = 100, A[i] = 101 - i , za i između 1 i 100

  Očekivani izlaz: A[i] = i za i između 1 i 100

- Test #5

  Ulaz: n = 101,

  - A[i] = 2, ako je i između 1 i 50,
  - A[i] = 1, ako je i između 51 i 100,
  - A[101] = 0

  Očekivani izlaz:

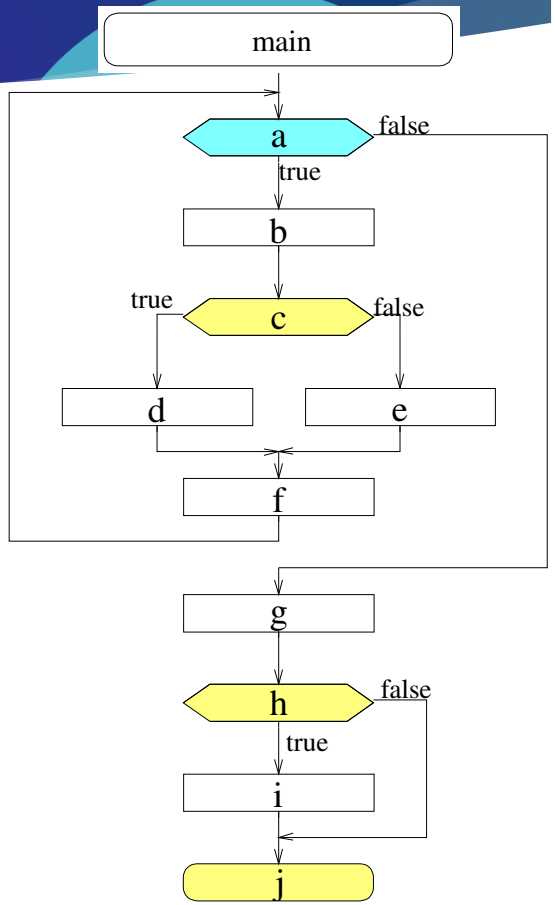
  - A[1] = 0,
  - A[i] = 1, ako je i između 2 i 51,
  - A[i] = 2, ako je i između 52 i 101.

Zadatak 2 - Uspeh studenata

- Metodom pokrivanja svih **linearno nezavisnih putanja** odrediti skup testova za sledeći C++ program koji određuje da li nastavniku treba povećati platu na osnovu pokazanog uspeha njegovih studenata na kvalifikacionom ispitu ☺.

```
#include <iostream.h>
  int main () {
    int uspesni = 0, neuspesni = 0, studentiBrojac = 1, rez;
1   while (studentiBrojac <= 10) {
2     cout<<"Unesite uspeh studenta (1=polozio,2=pao):";
3     cin >> rez;
4     if (rez == 1)
5       uspesni++;
6     else
        neuspesni++;
7     studentiBrojac++;
    }
8   cout << "Polozili " << uspesni << endl;
9   cout << "Pali " << neuspesni << endl;
10  if (uspesni > 8)
11    cout << "Podici platu " << endl;
12  return 0;
  }
```

Zadatak 2 - Uspeh studenata - Rešenje (1)



- Ciklomatska kompleknost: V(G) = e – n + 2
  zamenimo da je: e = 13 i n = 11 (zajedno sa main), dobijamo: **V(G) = 4**
- Putanje:
  - main – a – b – c – d – f – a – g – h – j
  - main – a – g – h – j
  - main – a – b – c – e – f – a – g – h – j
  - main – a – b – c – d – f – a – g – h – i – j
- Putanja 2 ne može se pokriti test primerom.
- Kod putanja 3 i 4 podvučeni deo se ponavlja još 9 puta zbog *while* uslova.
- Kod putanje 1, posle a – b – c – d – f dodajemo 9 puta a – b – c – e – f.

Zadatak 2 - Uspeh studenata - Rešenje (2)

- Testovi
- Za putanju 2: ulaz je 2, 2, 2, 2, 2, 2, 2, 2, 2, 2.
  Očekivani izlaz je: Položili 0 Pali 10
- Za putanju 3: ulaz je 1, 2, 2, 2, 2, 2, 2, 2, 2, 2.
  Očekivani izlaz je: Položili 1 Pali 9
- Za putanju 4: ulaz je 1, 1, 1, 1, 1, 1, 1, 1, 1, 1.
  Očekivani izlaz je: Položili 10 Pali 0 Podići platu

```
#include <iostream.h>
  int main () {
    int uspesni = 0, neuspesni = 0, studentiBrojac = 1, rez;
1   while (studentiBrojac <= 10) {
2     cout<<"Unesite uspeh studenta (1=polozio,2=pao):";
3     cin >> rez;
4     if (rez == 1)
5       uspesni++;
6     else
        neuspesni++;
7     studentiBrojac++;
    }
8   cout << "Polozili " << uspesni << endl;
9   cout << "Pali " << neuspesni << endl;
10  if (uspesni > 8)
11    cout << "Podici platu " << endl;
12  return 0;
  }
```

Zadatak 3 - Skokovi sa GoTo

- Odrediti sve LCSAJ sekvence u narednom programu.
- LCSAJ (eng. *Linear Code Sequence And Jump*) definiše se kao linearna sekvenca koda koja se izvršava ili od početka programa ili od mesta gde tok kontrole može skočiti, pa sve do kraja, ili do narednog skoka.
- Naziva i JJ-path, jer obuhvata putanje od skoka do skoka (eng. *jump-to-jump path*).

```
1.    A = 1
2.    IF X1 GOTO 40
3.    B = 1
4.    IF X2 GOTO 40
5.    A = 2
6.    GOTO 50
7.    40 A = A + B
8.    IF X3 GOTO 50
9.    B = B + 1
10.   50 WRITE(A, B)
```

Zadatak 3 - Skokovi sa GoTo - Rešenje (1)

- LCSAJ sekvence:

Sekvenca	Uslov	Ima skoka
2 -> 3	(X1 false)	
2 -> 7	(X1 true)	SKOK!
4 -> 5	(X2 false)	
4 -> 7	(X2 true)	SKOK!
6 -> 10	(true)	SKOK!
8 -> 9	(X3 = false)	
8 -> 10	(X3 = true)	SKOK!

- Postoje tri LCSAJ sekvence koji počinju od linije 1 (gledamo gde sve počev od 1 može da se izvrši skok):
  - (1, 2, 7), (1, 2, 3, 4, 7) i (1, 2, 3, 4, 5, 6, 10)
- LCSAJ sekvence sa početkom na liniji 7:
  - (7, 8, 10) i (7, 8, 9, 10, kraj programa)
- Poslednja sekvenca LCSAJ počinje na liniji 10:
  - (10, kraj programa)

Zadatak 3 - Skokovi sa GoTo - Rešenje (2)

- LCSAJ sekvence (kao uređene trojke):

Br.sekvence	POČETAK_SEKV	KRAJ_SEKV	SKOK
S1.	1	2	7
S2.	1	4	7
S3.	1	6	10
S4.	7	8	10
S5.	7	10	-1
S6.	10	10	-1

Zadatak 4 - FOR petlja i LCSAJ

- Neka je dat sledeći deo programskog koda u programskom jeziku Java.

  a) Odrediti sve LCSAJ za datu metodu.

  b) Odrediti minimalan skup testova koji pokrivaju sve LCSAJ.

```
   public void uredi() throws Greska{
1.      int n = niz.length;
2.      for (int i=1; i<n; i++) {
3.          double p = niz[i];
4.          int j = i - 1;
5.          while (j >= 0 && p < niz[j]) {
6.                niz[j+1] = niz[j--];
7.                prikazi();
                  //funkcija za prikazivanje niza
8.          }//while
9.      niz[j+1] = p;
10.     prikazi();
11.     }//for
12.   }
```

Zadatak 4 - FOR petlja i LCSAJ - Rešenje

- a) Postoje sledeće sekvence LCSAJ:

Br.sekvence	POČETAK_SEKV	KRAJ_SEKV	SKOK
S1.	1	2	12
S2.	1	5	9
S3.	1	8	5
S4.	5	8	5
S5.	5	5	9
S6.	9	11	2
S7.	2	8	5
S8.	2	5	9
S9.	2	2	12
S10.	12	12	-1

- Test primeri:
- TP1: niz = { 5 } pokriva sekvence S1 i S10
- TP2: niz = { 1, 2, 3 } pokriva sekvence S2, S6, S8, S9, S10
- TP3: niz = { 6, 5, 1, 2, 3 } pokriva sekvence S3, S5, S6, S7, S4, S9, S10

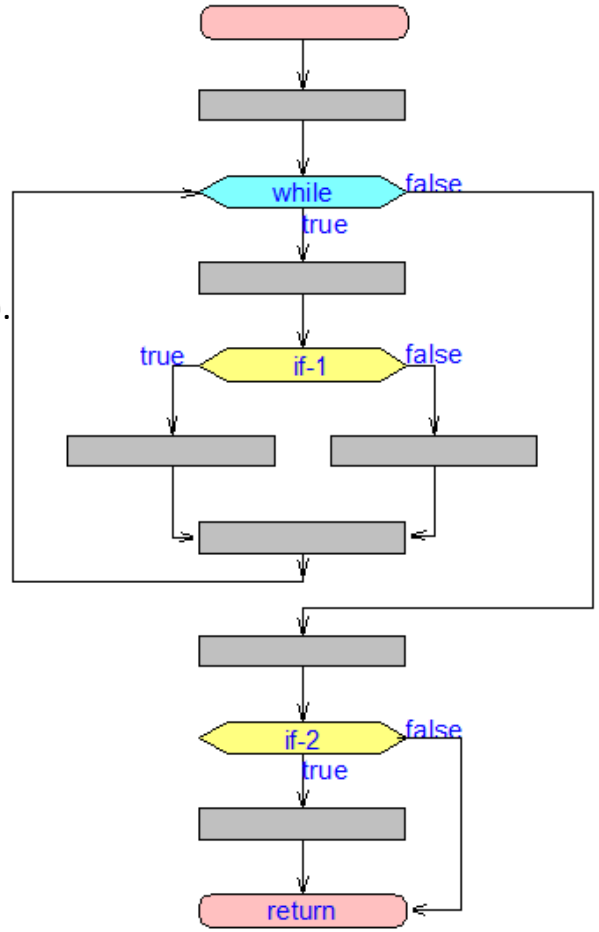
Zadatak 5 - Uspeh studenata

- **Metodom graničnog testiranja unutrašnje putanje** (eng. *boundary interior path testing*) odrediti test primere za dati program napisan na programskom jeziku C++.

```
#include <iostream.h>

int main () {
  int uspesni = 0, neuspesni = 0;
  int studentBrojac, rezultat;

  cout<<"Broj studenata:";
  cin>>studentBrojac;

  while (studentBrojac > 0)  {
    cout<<"Unesite rezultat (1=polozio,2=pao):";
    cin >> rezultat;
    if (rezultat == 1)
        uspesni++;
    else
        neuspesni++;

    studentBrojac--;
  }
  cout << "Broj koji su polozili: " << uspesni << endl;
  cout << "Broj koji su pali: " << neuspesni << endl;

  if (uspesni > 8) cout<<"Dobra prolaznost!";
  return 0;
}
```

Zadatak 5 - Uspeh studenata - Rešenje (1)

- 1. Za slučaj izvršavanja tela petlje 0 puta, postoje 2 putanje u grafu:
  - *False* grana WHILE, *False* grana IF-2. Odgovarajući test primer je: 0 studenata.
  - *False* grana WHILE, *True* grana IF-2.
    Ovde nije moguće odrediti test primer (ako ima 0 studenata, nemoguće je postići uspesni > 8).
- 2. Za slučaj izvršavanja tela petlje 1 put, postoje 4 putanje u grafu:
  - *True* grana WHILE, *False* grana IF-1, *False* grana WHILE, *False* grana IF-2.
    Odgovarajući test primer je: 1 student koji je pao.
  - *True* grana WHILE, *True* grana IF-1, *False* grana WHILE, *False* grana IF-2.
    Odgovarajući test primer je: 1 student koji je položio.
  - *True* grana WHILE, *False* grana IF-1, *False* grana WHILE, True grana IF-2.
    Ovde nije moguće odrediti test primer (ako ima 1 student, nemoguće je postići uspesni > 8).
  - *True* grana WHILE, *True* grana IF-1, *False* grana WHILE, *True* grana IF-2.
    Ovde nije moguće odrediti test primer (ako ima 1 student, nemoguće je postići uspesni > 8).



Zadatak 5 - Uspeh studenata - Rešenje (2)

- 3. Za slučaj ponavljanja tela petlje 1 ili više puta, postoje 4 putanje u grafu:
  - *True* grana WHILE, *False* grana IF-1, *True* grana WHILE,...
    False grana WHILE, *False* grana IF-2.
    Odgovarajući test primer je: 2 studenta koji su pali.
  - *True* grana WHILE, *True* grana IF-1, *True* grana WHILE,...
    False grana WHILE, *False* grana IF-2.
    Odgovarajući test primer je: 2 studenta koji su položili.
  - *True* grana WHILE, *False* grana IF-1, *True* grana WHILE,...
    False grana WHILE, *True* grana IF-2.
    Odgovarajući test primer je: 10 studenata, prvi pao, ostali položili.
  - *True* grana WHILE, *True* grana IF-1, *True* grana WHILE,...
    False grana WHILE, *True* grana IF-2.
    Odgovarajući test primer je: 10 studenata, svi položili.
- Konačan skup ima 7 test primera.