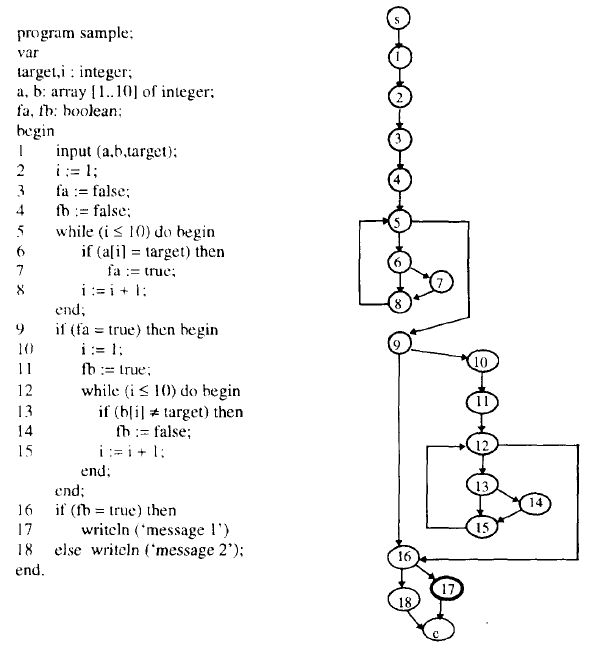
Testare Software 1,3,5

* 1. Testarea software este un proces de foarte lunga durata care atrage asupra sa costuri ridicate. Bugetul pentru aceasta procedura poate atinge 50% din costul total de dezvoltare al aplicatiei. Diverse studii au aratat ca automatizarea testarii poate scadea drastic atat costul cat si timpul de dezvoltare.
  2. Testarea software presupune ca anumite elemente din program sa fie evaluate. Din pacate, testarea initiala nu ajunge in toate punctele executiei programului din cauza diverselor ramificatii care pot exista in cod, asa ca programatorii trebuie sa gaseasca manual un set de date pentru a testa complet programul. Astfel, programatorii trebuie sa inteleaga foarte bine codul ceea ce nu se intampla mereu din cauza reutilizarii frecvente a bucatilor de program scrise de alti oameni.
  3. Exista 3 moduri in care putem genera date pentru test:
* Generare aleatoare
* Path-oriented (se selecteaza in mod automat sau manual o cale de executie pana la o anumita linie si se genereaza date pentru a ajunge pe toate nodurile din calea selectata)
* Goal-oriented (se selecteaza o linie de cod care sa fie rulata indiferent de calea prin care se ajunge la ea)
  1. Pentru unele programe, executia unei anumite linii de cod implica executia altor linii din program. In cazul metodelor path si goal-oriented, graful de control al fluxului de date nu este suficient pentru generarea datelor de test. De aceea chaining approach se bazeaza pe metoda goal-oriented, insa analizeaza dependenta de date pentru a identifica liniile de cod care afecteaza executia liniei selectate ca goal. Astfel, programul este obligat sa execute liniile gasite, iar sansele ca linia selectata sa fie de asemenea executata cresc.
  2. Atunci cand dorim sa generam date de test prin metoda goal-oriented cerinta este ceva de genul: Avand nodul g ca obiectiv al testarii, scopul este sa se gaseasca un set de date de intrare pentru ca nodul g sa fie executat.
  3. Exista 2 metode propuse pentru a gasi un input astfel incat sa se execute calea selectata: executia simbolica si generarea de date pentru test bazata pe executie
  4. Executia simbolica este o metoda de analizare a programului care genereaza o expresie algebrica pe valorile de intrare simbolice. De exemplu se genereaza constrangeri pentru respectarea firului de executie ales: egalitati si inegalitati intre variabilele de intrare. Desi pare o metoda buna pentru testare, aceasta metoda intampina dificultati cand exista structuri de date dinamice, structuri de date multidimensionale cum ar fi vectori sau matrice.
  5. Generarea bazata pe executie presupune executarea programului folosind minimizarea functiilor. In aceasta varianta obiectivul de a gasi datele de intrare este atins prin rezolvarea unei secvente de obiective intermediare, folosind aceasta minimizare. Deoarece metoda se bazeaza pe executarea propriu-zisa a programului, datele problematice in metoda anterioara sunt acum stiute si nu mai reprezinta un impediment.
  6. Totusi exista limitari in ambele metode path-oriented din cauza procesului de selectie a firului de executie. Dezavantajul de a nu sti daca firul selectat este fezabil, limiteaza utilitatea acestei metode de testare pentru anumite programe, intrucat se folosesc resurse computationale semnificative pentru fire de executie nefezabile. Metoda path-oriented este cea mai potrivita pentru programe cu un numar relativ mic de fire de executie. In caz ca avem programe cu structuri de control complexe si un numar infinit de cai pentru a ajunge intr-un anumit nod, aceasta metoda devine ineficienta, deoarece se foloseste numai graful de control atunci cand se selecteaza calea.
  7. Din cauza neajunsurilor prezentate anterior, metoda chaining-approach se bazeaza pe goal-oriented-approach si astfel partea in care este selectata calea va fi eliminata. Aceasta metoda incepe prin executarea programului folosind date de intrare arbitrare. Dupa ce programul e executat, fluxul de executie al programului este monitorizat. In timpul executiei, procedura de cautare decide daca executia ar trebui sa continue prin ramificatia curenta sau daca alta ramificatie ar trebui urmata. Aceasta schimbare ar putea fi facuta daca ramificatia aleasa nu duce la nodul selectat. Ideea generala a acestei abordari este de a concentra testarea numai pe firele de executie care influenteaza executia nodului obiectiv si de a ignora testarea cailor care nu influenteaza in niciun fel ajungerea la acest nod. De exemplu in imaginea de mai jos executia caii (5, 6) nu influenteaza cu nimic executia nodului 11, programul ajungand in nodul 9, daca executia buclei 5-8 se termina. Calea (9, 16) este esentiala pentru executia nodului 11, deoarece daca aceasta se executa, nodul 11 nu va fi atins. Daca programul de testare constata executia caii (9, 16), atunci procedura de cautare suspenda executia si incearca sa gaseasca un nou input pentru a intra pe calea (9, 10). Procedura de cautare clasifica toate caile pe baza grafului de executie. Aceasta foloseste clasificarea pentru a decide pe parcursul executiei daca suspenda sau continua executia programului.



* 1. O ramura (p, q) este o ramura critica pentru evenimentele ei si ei+1 daca:

1. Nu exista o cale de la p la ni+1 sau ni care sa nu schimbe datele din Si prin parcurgerea ramurii (p, q) si
2. Exista o cale alternativa ramurii (p,q) de la p la ni+1 .

* Daca o astfel de ramura critica este executata, intreaga executie a programului este suspendata deoarece executia nu poate ajunge la ni+1 sau pentru ca acea cale ar schimba datele.
* Algoritmul de cautare incearca sa gaseasca date de intrare care va schimba firul de executie al programului astfel incat sa nu se mai execute (p, q).
* Daca se gaseste o alternativa, executia v-a continua pe aceasta.
* Daca nu se gaseste, nodul p este declarat nod problema si se for genera noi secvente de evenimente.
* De exemplu 2 evenimente (11, {fb} ) si (16, Ø ):
  + Ramura (13, 14) este critica, deoarece daca este executata valoarea lui fb este modificata in nodul 14
  + In acest caz un nou input trebuie sa fie gasit pentru ca ramura (13, 15) sa fie aleasa
  1. O ramura (p, q) este o ramura semi-critica pentru evenimentele ei si ei+1 daca:

1. (p, q) nu este critica
2. ni+1 este dependent de nodul p
3. nu exista o cale aciclica de la p la ni+1  fara ca datele din Si  sa fie modificate.

* Daca o astfel de ramura critica este executata, intreaga executie a programului este suspendata deoarece executia nu poate ajunge la ni+1 sau pentru ca acea cale ar schimba datele.
* Algoritmul de cautare incearca sa gaseasca date de intrare care va schimba firul de executie al programului astfel incat sa nu se mai execute (p, q).
* Daca se gaseste o alternativa, executia v-a continua pe aceasta.
* Daca nu se gaseste, executia va fi permisa in continuare pe ramura (p, q), sperand ca la urmatoarea executie a nodului p, ramura alternativa va fi urmata.
* De exemplu 2 evenimente (11, {fb} ) si (16, Ø ):
  + Ramura (12, 13) este critica deoarece pentru a ajunge in nodul 16 programul trebuie sa execute (probabil repetat) bucla (12, 15) si sa ajunga din nou in nodul 12, fara a modifica variabila „fb”
  + Daca ramura semi-critica este executata exista pericolul de a executa o ramura critica, acesta fiind motivul pentru care mereu este de preferat o ramura alternativa in detrimentul uneia semi-critice
  1. Celelalte ramuri sunt clasificate ca ramuri neesentiale. Daca programul executa o astfel de ramura, executia este lasata sa continue.
  2. Exemplu: avem urmatoarea secventa de evenimente

E = < (s, Ø ) , (7, {fa} ), (9, Ø ), (11, {fb} ), (16, Ø ), (17, Ø ) > rezulta urmatoarea clasificare

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. Clasificare | Perechi de evenimente | Ramuri Critice | Ramuri Semi-Critice |
| **1** | (s, Ø ) , (7, {fa} ) | (5, 9) | (6, 8) |
| **2** | (7, {fa} ), (9, Ø ) | - | (5, 6) |
| **3** | (9, Ø ), (11, {fb} ) | (9, 16) | - |
| **4** | (11, {fb} ), (16, Ø ) | (13, 14) | (12, 13) |
| **5** | (16, Ø ), (17, Ø ) | (16, 18) | - |

Aceste clasificari sunt folosite pentru a controla executia programului. La inceput se foloseste clasificarea 1, pana cand programul ajunge in nodul 7, apoi se trece la clasificarea 2 pana cand ajunge in nodul 9, apoi foloseste clasificarea 3 si tot asa. De fiecare data cand un eveniment se intampla, se trece la clasificarea urmatoare.