N: gráf éleinek a száma

C: gráf csúcsainak a száma

$m(G)$: a gráf ciklikus rangja

Program-gráf esetén:

$$m(G) = n - c \quad \text{(ciklikus bonyolultság)}$$

Tétel: A programgráf ciklikus bonyolultságának mértéke:

$$m(T) = \Pi + 1$$

Π: predikátum csomópontok száma

Tétel: A nem strukturált program ciklikus bonyolultsága legalább 3. (legalább 2 alapforma miatt)

Lebontással a program ciklikus bonyolultsága csökken ⇒ valódi jellemző a lebontás után kapott programgráf ciklikus bonyolultsága.

A T program lényeges bonyolultságának mértékszáma:

$$M(T) = m(T) - E$$

E: ITE, WD, DU (repeat-until) típusú részgráfok száma.

Strukturált program lényeges bonyolultsági száma: 1.

minden ilyen struktúra mögött van egy predikátum-csomópont.

A program helyessége:

A „vizsgálat" módszerei:

(A programot fgv-nek tekintjük, van ért. tart-a, és ért. készlete)

∀ ért. tartománybeli elemhez az értékkészlet beli elemet (a) helyes rendeljük

Valilyen módszerrel meggyőződünk a helyességéről.

Van bemenete és kimenete

Ha a megfelelő bemenethez a megfelelő kimenetet (és annak értékét) rendeli.

- A program helyességének a vizsgálata:

A: 1) tesztelés (programigazolás)

- a cél a programhiba felismerése
- próbaadatok feldolgozása

bemenete: program

kimenet: logikai típusú (helyes v. nem?)

- azért próbaadatokat használunk, mert ∀ sok bemenetet nekt. nem produkálni.

Hogyan válasszuk meg a próbaadatokat, h. minél kevesebbet nelljen használni?

2: Programérvényesítés:

- cél annak vizsgálata, hogy a program alkalmas-e a kitűzött feladatok megoldására
- logikai helyesség valódi környezetben

3: Programhitelesítés:

- hosszú idő alatt összegyűlt futási tapasztalat alapján

A: gyakorlati tapasztalatokra épülő ellenőrzés
A: helyesség vizsgálat

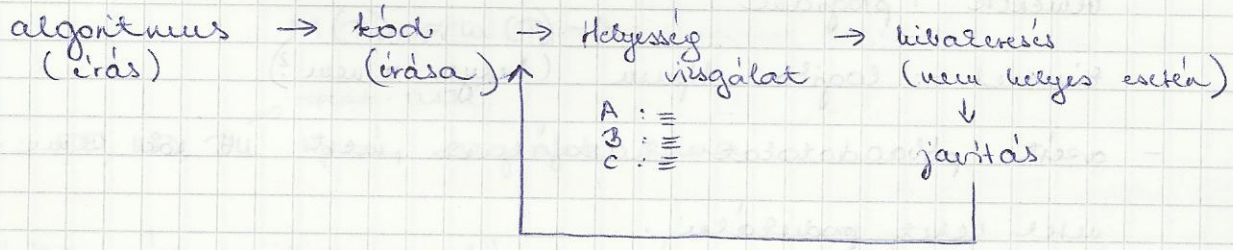
B: 4: Hibakeresés:

- cél a hiba helyének és okának feltárása
- ha A1, A2, A3 mind hamis értéket ad.

C: 5; Programhelyesség bizonyítása:

- logikai helyesség igazolása axiómák, szabályok alapján.
- A program állításként fogalmazható meg → lehet bizonyítani v. cáfolni.

Hogy készül egy program?



Miért van több vizsgálat?

pl.: nem mindig készül a program hitelesítés:

hadászat, orvoslás ... stb.

Tesztelés:

- cél : a próbaadatok feldolgozására kevesebb időt fordítsunk, de minél több programhibára fény derüljön.

- teszteset: a be- és kimenő adatok és feltételek együttes megadása

- A tesztelés alapelvei → (tesztesek választásának alapelvei)

• még föl nem fedezett hibát tárjon fel
(nincs szükség olyanra, amit felfedeztünk már)

• a megismételhetetlen tesztesetek feleslegesek

• a lehető legtöbbet kell hasznosítani egy tesztből

• külső személy bevonása → nézze át vele, magyarázzuk el

• a teszteseteket dokumentálni kell (be és ki)

• érvényes és érvénytelen adatokra is ki kell készíteni

• elemzés: + egy funkciót miért nem hajt végre, pedig kellene
+ miért végez el valamit, amit nem kell.

I. Statikus tesztelés: nem él a program, nem kell közben gép.
(a kódot vizsgáljuk)

- kódellenőrzés → algoritm logikáját követjük a programkódban
felsőkellesség, h. az algoritm helyes

- formai ellenőrzés : szintaktikai ellenőrzés

- tartalmi ellentmondás: (őrzés)

• ellentmondás - keresés

A mostani fejlesztői programok nagy segítséget nyújtanak.

pl.: „;" hiánya

vannak olyan progr.nyelvek, amelyek a formai ellenőrzést

soronként végzik, addig nem engednek tovább.

tartalmi ellenőrzés: pl.: A = 3.
A := 5;

pl.: deklarálunk 1 változót, és nem használjuk fel (C nyelv)

hibás ciklusfeltétel

rekurzióban hibás báziskritériumok.

II. Dinamikus: futtatjuk a programot, figyeljük a működését

- fekete doboz -*ekvivalencia oszt. ker.

adatvezérelt

Kimerítő bemenet teszt ☹ → minden lehetséges bemenettel kipróbáljuk, de ilyen nincs. (még az egyszerűeknél sem)

- a rendszerszemlélettől adódik
- a program, mint rendszer működik
- nem ismerjük a rendszer belső működését, csak h. milyen bemenetre milyen kimenettel reagál
- adatvezérelt tesztelési módszerrel egyezik meg.

- fehér doboz **

logikavezérelt

Kimerítő út teszt ☹ → az összes lehetséges utat bejárjuk / nem biztos, h. sikerül

- a program felírható gráfként ⇒ utak vannak

Fekete doboz: *• ekvivalencia oszt. ker.

+ minél több bemeneti feltétel kielégítése

+ bemeneti tartományok részekre osztása → ekvivalenciaosztályok (aszerint, h. mely hibákat tárnák fel)

• határeset elemzés

testeset választások:

~a lehető legtöbb bemeneti feltételt elégítse ki

pl.: pozitív páratlan prím -> ha azt vizsgáljuk.

ekvivalenciaosztályok: ugyanazon ekvivalenciaosztályból választva

elemet => másik tesztesettel lefuttatva ugyanúgy fut le.

Ha jó, akkor itt is jó... stb.

Külön ekv. osztályokba sorolhatók: érvényes és érvénytelen adatok halmaza.

határeset elemzés:

· itt is megtörténik az ekv. osztályokra való bontás

· ha pl.: a bemenő adatok R-ből kerülnek ki => az ekv. osztályok számegyenesei -> részintervallumok adódnak. Ezek határán lévőket választjuk tesztesetnek. (intervallum alsó- és felső határa)

pl.: ciklusnál kezdő- és végértékre helyesen működik-e?

rossz: ciklus i := 1 .. n
ha A[i] > A[i+1]

· ha a bemenő adatok számossága érdekes a bemeneti értékekben: A számosság is intervallummal jellemezhető

érdemes megpróbálni

(0..1) , (0.. max) (0.. max+1),
-> érvénytelen adatok

Fehér doboz: * * - utasítások egyenkénti lefedése
· lehetőleg minden utasításra min 1x sor kerül
· csak szekvenciális program esetén biztosítja minden út bejárását.

- döntéslefedés

; cél: olyan tesztesetek, amelyeknél minden feltételes elágazás igaz és hamis ágát bejárjuk.

· elegt tesz az utasítások 1x lefedés erejéig is.

- feltétellefedés:

· olyan tesztesetet választunk, h. minden logikai lefedés minden elemét felvegye egyszer az igaz és a hamis értéket is.

- döntés - v. feltétellefedés:

- streszt - teszt

· nagy mennyiségű véletlenszerű adatokkal bombázzuk a programot

Program hibakeresése:

· hibafajták lehet pl.: pascal-nál a forrás formátuma nem tükrözi a logikai szerkezetét
(pl.: ciklusmag helyesen kezdődik ... stb)

· megelőzés → követjük a formátumozást, és így elkerüljük az ebből adódó hibákat

→ felbontjuk a feladatot részfeladatokra, ezek könnyebben áttekinthetők → jobban meg lehet vizsgálni, hogy helyesek-e...
a hibakeresés is egyenlő (nő a hiba a kód nagyságával)

→ alprogramoknál felmerül, hogyan kommunikáljanak ezek.
Nem jó: a globális változók használata (csökkenti az egyes részprogramok hordozhatóságát)

Hibakeresés alapelvei:

1; Kód vizsgálata ⇒ megpróbálni megkeresni a hibát ⇒ alsszenő tintában lenni a programmal lépésről lépésre.

2; amíg nem ismerjük a hiba teljes (pontos) okát, addig ne változtassunk a hibakódon. ⇒ újabb hibaforrás lehet.

Két dolog miatt kell újbóli hibakeresést végezni
- nem biztos, h. sikerült kijavítani
- lehet, h. a hiba elfedett más hibajelenségeket

- Törekedjünk mindig a hiba okának megszüntetésére
  pl.: nem kell egy kiírásnak megjelennie ⇒ letöröljük a képernyőt ⇒ a jó megoldás: nem jelenítjük meg.

- A hiba kijavításának valószínűsége a program méretével arányos

Hibakeresési módszerek:

a; indukciós

b; dedukciós

c; visszalépéses ⇒ a program végrehajtásának menetében a hibától addig haladunk visszafelé, amíg még helyesen működik

d; teszteléssel segített hibakeresés ⇒ mely ekvivalenciaosztálybeli elemmel juttatjuk.

Hibakeresés eszközei:

- ott ahol a hibát sejtjük, elvégzünk egy kiíratást. Nem célszerű, mert a programkódot módosítjuk.

- nyomtörvelés
- lépésenkénti végrehajtás
- töréspont beiktatása