

The background of the slide is a black and white aerial photograph of Budapest, Hungary. It shows the city's dense architecture, including the prominent dome of St. Stephen's Basilica in the foreground. The Danube River is visible on the left, and the city extends to the hills in the distance.

Programozás

7. előadás

A mai előadás foglalatja képekben, szólásokkal



➤ „Tévedni emberi (dolog).”

- Seneca



➤ „Más szemében meglátja a szálkát,
a magáéban a gerendát sem veszi észre.”

- Mt. 7,3/Lk 6,41



A mai előadás foglalatja, mondásokkal



➤ „Tévedni emberi (dolog).”



A mai előadás foglalatja, mondásokkal



- „Más szemében meglátja a szálkát, a magáéban a gerendát sem veszi észre.”



Tartalom

➤ Tesztelés

- fogalmak + elvek
- statikus tesztelés
- dinamikus tesztelés
 - fekete doboz módszerek
 - szürke doboz módszerek
 - fehér doboz módszerek
- technika: futtatás adatfájlal – C++
- szabályos tesztek, véletlen tesztek

➤ Hibakeresés

- elvek + eszközök
- módszerek

➤ Hibajavítás





Tesztelés fogalmak



Célja:

a hibás működés kimutatása.

Tesztelési fogalmak:

- Teszteset = **bemenet** + **kimenet**
- Próba = teszteset-halmaz
- Jó teszteset: nagy valószínűséggel felfedetlen hibát mutat ki
- Ideális próba: minden hibát kimutat
- Megbízható próba: nagy valószínűséggel minden hibát kimutat

A „hiba” helyett jobb lenne „problémát” mondani, mivel a tesztelés nemcsak a hibakeresés, hanem a hatékonyságvizsgálat eszköze is.



Tesztelés elvek



Tesztelési elvek:

- Érvényes (megengedett) és érvénytelen (hibás) bemenetre is kell tesztelni.
- Minden teszteset által nyújtott információt maximálisan ki kell használni (a következő tesztesetek kiválasztásánál).
- Csak más (mint a szerző) tudja jól tesztelni a programot.
- A hibák nagy része a kód kis részében van.
- Rossz a meg nem ismételhető teszteset.
(Ez nem a tesztelés, hanem a tesztelendő program vonása.)



Tesztelési módszerek



Tesztelési módszerek:

- Statikus tesztelés: a programszöveget vizsgáljuk, a program futtatása nélkül.
- Dinamikus tesztelés: a programot futtatjuk különböző bemenetekkel és a kapott eredményeket vizsgáljuk.

A tesztelés eredménye:

- hibajelenséget találtunk;
- nem találtunk – még – hibát.

A tesztelés egyik alapkérdése:

- meddig teszteljünk?



Statikus tesztelés



➤ KódelLENőrzés:

- algoritmus↔kód megfeleltetés
 - kódolási hibák kimutatásra
- algoritmus+kód elmagyarázása másnak
 - algoritmikus+kódolás hibák kimutatására

➤ Szintaktikus ellenőrzés:

- fordítóprogram esetén automatikus
 - bár nem mindig kellően szigorú
- értelmező esetén sok futtatással jár
 - tehát tulajdonképpen dinamikus



Statikus tesztelés

➤ Szemantikus ellenőrzés, ellentmondás keresés:

- o felhasználatlan változó/érték

```
i=1;  
for (i=2;...)  
{ ... }
```

- o „gyanús” változóhasználat

```
i=...; //ez esetleg jóval előbb található  
for (int i=2;...)  
{ ... i ... }  
... i ...
```

- o „identikus” transzformáció

```
i=1*i+0; //n1? K0? 0?
```



Statikus tesztelés

➤ Szemantikus ellenőrzés, ellentmondás keresés (folytatás):

- o inicializálatlan változó

```
int n;    //meggondolatlan kódolása
int k[n]; //a specifikációbeli adatléírásnak
```

- o definiálatlan (?) értékű kifejezés:

```
int n; ← globális n
...
int fv()
{
    for (int n=0; n<9; ++n) ← lokális n
    {
        ... n ... ; ← a lokális n felhasználása
    }
    return ...n...; ← a globális n-et tartalmazó formula
}
```



Statikus tesztelés

➤ Szemantikus ellenőrzés, ellentmondás keresés (folytatás):

- o érték nélküli függvény

szintaktikusan **hibás** a C++ nyelvben:

```
int fv()  
{  
    ...  
    return; ← ... error: return-statement with no value, in  
              function returning 'int' |  
}
```

... de az alábbi csak **figyelmeztetés**t vált ki:

```
int fv()  
{  
    ...  
} ← ... warning: control reaches end of non-void function |
```



Statikus tesztelés

➤ Szemantikus ellenőrzés, ellentmondás keresés (folytatás):

- azonosan igaz/**hamis** feltétel

$(N > 1 \ || \ N < 100) \ / \ (N < 1 \ \&\& \ N > \text{maxN})$

Gyakori hiba a **beolvasás-ellenőrzés** kódolásakor.

- végtelen számlálós ciklus

```
for (int i=0; i<N; ++i)
{
    ...
    --i; ← nem vált ki fordítási hibaiüzenetet
}
```

Talán egy **while**-os ciklus átírása során maradhatott benn.



Statikus tesztelés

➤ Szemantikus ellenőrzés, ellentmondás keresés (folytatás):

- o pontatlan ciklus-szervezés

```
for (int i=0; i<N; ++i)
{
    ...
    ++i; ← nem okoz fordítási hibaiüzenetet
}
```

Egy **while**-os ciklus következtelen átírása során maradhatott benn.

- o konstans értékű, (bár) változókat tartalmazó kifejezés

```
y=sin(x)*cos(x)-sin(2*x)/2
```



Statikus tesztelés

➤ Szemantikus ellenőrzés, ellentmondás keresés

(folytatás):

- o végtelen feltételes ciklus ($i < N$ feltételű ciklusban sem i , sem N nem változik, vagy „szinkronban” változik)

```
i=1;  
while (i<=N)  
{  
    ...  
    i=+1; ← talán i+=1 akart lenni?  
}
```



Statikus tesztelés

➤ Szemantikus ellenőrzés, ellentmondás keresés

(folytatás):

- o végtelen feltételes ciklus ($i < N$ feltételű ciklusban sem i , sem N nem változik, vagy „szinkronban” változik)

```
i=1;  
while (i<=N)  
{  
    ...  
    i=i++; ← talán i++ vagy i=i+1 akart lenni?  
}
```



Dinamikus tesztelés



Tesztelési módszerek:

- **Fekete doboz** módszerek (← *nincs kimerítő bemenet* – nem lehet minden lehetséges bemenetre kipróbálni): a teszteseteket a program **specifikációja** alapján **optimálisan** választjuk.
- **Fehér doboz** módszerek (← *nincs kimerítő út* – nem lehet minden végrehajtási sorrendre kipróbálni): a teszteseteket a **program struktúrája** alapján **optimálisan** választjuk.
- **Szürke doboz** módszerek – a konkrét algoritmust nem ismerjük, de a típusát igen, a tesztelést erre alapozzuk.



Dinamikus tesztelés: fekete doboz módszerek



- **Ekvivalencia-osztályok módszere:** a bemeneteket (vagy a kimeneteket) soroljuk olyan diszjunkt osztályokba, amelyekre a program várhatóan egyformán működik; ezután osztályonként egy tesztesetet válasszunk!
- **Határeset elemzés módszere:** az ekvivalencia-osztályok határáról válasszunk tesztesetet (be- és kimeneti osztályokra is)!
- **Ok-hatás analízis:** ekvivalencia osztályt leíró bemeneti feltételek (ok) és kimeneti feltételek (hatás) kombinálása.



Ekvivalencia-osztályok módszere – osztályozás



- Ha a bemeneti feltétel értéktartományt definiál, az érvényes ekvivalencia osztály legyen a megengedett bemenő értékek halmaza, az érvénytelen ekvivalencia osztályok pedig az alsó és a felső határoló tartomány. Pl. ha az adatok osztályzatok (értékük 1 és 5 között van), akkor ezek az ekvivalencia osztályok rendre: $\{1 \leq i \leq 5\}$, $\{i < 1\}$ és $\{i > 5\}$.
- Ha a bemeneti feltétel értékek számát határozza meg, akkor az előzőhöz hasonlóan járjunk el. Pl. ha be kell olvassunk legfeljebb 6 karaktert, akkor az érvényes ekvivalencia osztály: 0-6 karakter beolvasása, az érvénytelen ekvivalencia osztály: 6-nál több karakter beolvasása. (0-nál kevesebb nem fordulhat elő.)



Ekvivalencia-osztályok módszere – osztályozás



- Ha a bemenet feltétele azt mondja ki, hogy a bemenő adatnak valamilyen meghatározott jellemzővel kell rendelkezni, akkor két ekvivalencia osztályt kell felvenni: egy érvényeset és egy érvénytelent.
- Ha okunk van feltételezni, hogy a program valamelyik ekvivalencia osztályba eső elemeket különféleképpen kezeli, akkor a feltételezésnek megfelelően bontsuk az ekvivalencia osztályt további osztályokra.
- Alkalmazzuk ugyanezeket az elveket a kimeneti ekvivalencia osztályokra is!



Ekvivalencia-osztályok módszere – tesztesetek



A teszteseteket a következő két elv alapján határozhatjuk meg:

- Amíg az érvényes ekvivalencia osztályokat le nem fedtük, addig készítsünk olyan teszteseteket, amelyek minél több érvényes ekvivalencia osztályt lefednek!
- Minden érvénytelen ekvivalencia osztályra írjunk egy-egy, az osztályt lefedő tesztesetet. Több hiba esetén ugyanis előfordulhat, hogy a hibás adatok lefedik egymást, a második hiba kijelzésére az első hibajelzés miatt már nem kerül sor.

Megjegyzés: mindegyikhez 1-1 hibajelzésnek kell tartoznia a programban.



Határeset elemzés módszere



- Ha a bemeneti feltétel egy értéktartományt jelöl meg, írjunk teszteseteket az érvényes tartomány alsó és felső határára és az érvénytelen tartománynak a határ közelébe eső elemére! Pl.: ha a bemeneti tartomány a $(0,1)$ nyílt intervallum, akkor a 0, 1, 0.01, 0.99 értékekre érdemes kipróbálni a programot.
- Ha egy bemeneti feltétel értékek számosságát adja meg, akkor hasonlóan járjunk el, mint az előző esetben. Pl.: ha rendeznünk kell 1-128 nevet, akkor célszerű a programot kipróbálni 0, 1, 128, 129 névvel.



Ok-hatás elemzés módszere



Ok-hatás analízis: ekvivalencia osztályt leíró bemeneti feltételek (ok) és kimeneti feltételek (hatás) kombinálása – azaz a bemenetek és kimenetek kapcsolatának vizsgálata.

Készítsünk egy gráfot, ami a bemeneti feltételeket összeköti a kimeneti feltételekkel, esetenként VAGY, ÉS, illetve NEM kapcsolatokkal.

A tesztelési terv ennek a gráfnak a teljes bejárása.



Dinamikus tesztelés: fekete doboz módszerek



Feladat: Adjuk meg egy N természetes szám valódi (1-től és önmagától különböző) osztóját!

Ekvivalencia osztályok (bemenet alapján):

Specifikáció:

- Bemenet: $N \in \mathbb{N}$
- Kimenet: $O \in \mathbb{N}, V \in \mathbb{L}$
- Előfeltétel: $N > 1$
- Utófeltétel: $V \wedge \exists i (2 \leq i < N): i \mid N$
 $V \wedge \rightarrow 2 \leq O < N$ és $O \mid N$ és
 $\forall i (2 \leq i < O): i \nmid N$

1. N prímszám: 3

2. N -nek egy(-féle) valódi osztója van: $25 = 5 * 5$

Érvényes adatokra

3. N -nek több, különböző valódi osztója is van: $77 = 7 * 11$

4. N páros : 2, $4 = 2 * 2$, $6 = 2 * 3$

$\subset 1. \cup 2. \cup 3.$

5. $N = 1$, vagy bármi, ami nem természetes szám

Érvénytelen adatokra

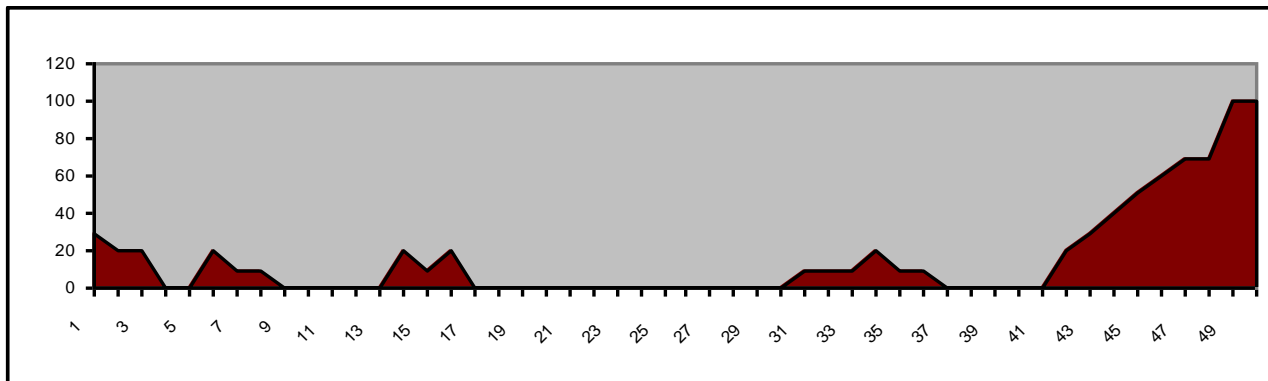


Dinamikus tesztelés: fekete doboz módszerek



Feladat:

Egy repülőgéppel Európából Amerikába repültünk. Az út során bizonyos kilométerenként mértük a felszín tengerszint feletti magasságát (≥ 0). 0 magasságot ott mértünk, ahol tenger van, >0 -t pedig ott, ahol szárazföld. Adjuk meg a legszélesebb szigetet!



Dinamikus tesztelés: fekete doboz módszerek



Feladat:

Egy repülőgéppel Európából Amerikába repültünk. Az út során bizonyos kilométerenként mértük a felszín tengerszint feletti magasságát (≥ 0). 0 magasságot ott mértünk, ahol tenger van, >0 -t pedig ott, ahol szárazföld. Adjuk meg a legszélesebb szigetet!

Specifikáció:

- Bemenet: $N \in \mathbb{N}$, $\text{Mag}_{1..N} \in \mathbb{N}^N$
- Kimenet: $V \in \mathbb{L}$, $K, V \in \mathbb{N}$
- **Előfeltétel:** $\text{Mag}_1 > 0$ és $\text{Mag}_N > 0$ és
 $\forall i(1 < i < N): \text{Mag}_i \geq 0$ és
 $\exists i(1 < i < N): \text{Mag}_i = 0 \quad [\rightarrow N \geq 3]$

Érvénytelen ekvivalencia osztályok:

- $N < 3$
- $\text{Mag}_1 \leq 0$
- $\text{Mag}_N \leq 0$
- $\exists i(1 < i < N): \text{Mag}_i < 0$
- $\forall i(1 < i < N): \text{Mag}_i > 0$



Dinamikus tesztelés: fekete doboz módszerek



Érvényes ekvivalencia osztályok (a kimenet alapján):

- nincs sziget
- van sziget
 - egy sziget van
 - több sziget van
 - egyforma szélességűek
 - nem egyforma szélességűek
 - az első a legszélesebb
 - az utolsó a legszélesebb
 - egy közbülső a legszélesebb

Feladat:

Egy repülőgéppel Európából Amerikába repültünk. Az út során bizonyos kilométerenként mértük a felszín tengerszint feletti magasságát (≥ 0). 0 magasságot ott mértünk, ahol tenger van, >0 -t pedig ott, ahol szárazföld. Adjuk meg a legszélesebb szigetet!



Dinamikus tesztelés: fekete doboz módszerek



Határesetek:

- Európa 1 szélességű, nem 1 szélességű;
- Amerika 1 szélességű, nem 1 szélességű;
- a legszélesebb sziget 1 szélességű, nem 1 szélességű;
- a legszélesebb szigetet (bal, ill. jobb) szomszédjától 1 szélességű tenger választja el, nem 1 szélességű tenger választja el.

Ez hány tesztet? Az első kettő négyféleképpen kombinálható. A harmadik ezek számát megduplázza. A negyedik pedig négyszerezi.

32 tesztet!!!

Feladat:

Egy repülőgéppel Európából Amerikába repültünk. Az út során bizonyos kilométerenként mértük a felszín tengerszint feletti magasságát (≥ 0). 0 magasságot ott mértünk, ahol tenger van, >0 -t pedig ott, ahol szárazföld. Adjuk meg a legszélesebb szigetet!



Dinamikus tesztelés: fekete doboz módszerek



Feladat:

Egy repülőgéppel Európából Amerikába repültünk. Az út során bizonyos kilométerenként mértük a felszín tengerszint feletti magasságát (≥ 0). 0 magasságot ott mértünk, ahol tenger van, >0 -t pedig ott, ahol szárazföld. Adjuk meg a legszélesebb szigetet!

Megtaláljuk ezzel az **összes** hibát?

Tételezzük fel, hogy a megoldásban először megkeressük Európa utolsó és Amerika első pontját, majd e két pont között keressük a szigeteket.

Ha az Amerika első pontját tartalmazó változót elrontjuk, pl. $N/2$ -re vagy $N-10$ -re állítjuk, akkor mi a garancia arra, hogy a korábbi tesztek ezt felfedezik?

Mi van, ha a programunk Európát és Amerikát is szigetként veszi számításba, és adja legszélesebb szigetnek?



Dinamikus tesztelés: szürke doboz módszerek



Határok vizsgálata sorozatoknál

- Első elem feldolgozásra kerül-e
- Utolsó elem feldolgozásra kerül-e
- Közbenső elem feldolgozásra kerül-e

Sorozat mérete szerint:

- Üres sorozat kezelése
- Egy elemű sorozat kezelése
- (Két elemű sorozat kezelése)
- Több elemű sorozat kezelése



Dinamikus tesztelés: szürke doboz módszerek



Összegzés

- Két különböző elemet tartalmazó sorozat.
- Terheléses teszt, túlcsordulás vizsgálat.

Megszámolás

- Két elemű sorozat, ahol mindkét elem kielégíti a számlálás feltételét.
- Az adott tulajdonságnak a sorozatban nulla, egy, kettő vagy több elem tesz eleget.

Kiválasztás

- A kiválasztandó elem a sorozat első eleme.
- A kiválasztandó elem a sorozatnak **nem** az első eleme.



Dinamikus tesztelés: szürke doboz módszerek



Keresés

- A keresett elem a sorozat első eleme.
- A keresett elem a sorozat utolsó eleme.
- Létezik a keresett tulajdonságnak megfelelő közbenső elem.
- Nem létezik a keresett tulajdonságnak megfelelő elem.

Maximum-kiválasztás

- Két elemű sorozat, első eleme a nagyobb.
- Két elemű sorozat, második eleme a nagyobb.
- Több elemű sorozat közbenső eleme a legnagyobb.
- Több elemű sorozatban több maximális elem van.



Dinamikus tesztelés: fehér doboz módszerek



Fehér doboz módszerek

- egy kipróbálási stratégiát választunk a program szerkezete alapján,
- a stratégia alapján megadott tesztutakhoz tesztpredikátumokat rendelünk,
- a tesztpredikátumok ekvivalencia osztályokat jelölnek ki, amelyekből egy-egy tesztesetet választunk.



Dinamikus tesztelés: fehér doboz módszerek



Kipróbálási stratégiák:

- utasítás lefedés: minden utasítást legalább egyszer hajtsunk végre!
- feltétel lefedés: minden feltétel legyen legalább egyszer igaz, illetve hamis!
- részfeltétel lefedés: minden részfeltétel legyen legalább egyszer igaz, illetve hamis!



Dinamikus tesztelés: fehér doboz módszerek



Teszteset-generálás

Bázisútnak nevezzük a programgráf olyan útját, amely

- a kezdőponttól a legelső elágazás- vagy ciklusfeltétel kiértékeléséig tart,
- elágazás- vagy ciklusfeltételtől a következő elágazás- vagy ciklusfeltétel helyéig vezet,
- elágazás- vagy ciklusfeltételtől a program végéig tart, s közben más feltétel kiértékelés nincs.



Dinamikus tesztelés: fehér doboz módszerek



Tesztutaknak nevezzük a programgráfon átvezető, a kezdőponttól a végpontig haladó olyan utakat, amelyek minden bennük szereplő élt pontosan egyszer tartalmazznak.

Tesztpredikátumnak nevezzük azokat a bemenetre vonatkozó feltételeket, amelyek teljesülése esetén pontosan egy tesztúton kell végighaladni.

A teszteset-generálás első lépése a minimális számú olyan tesztút meghatározása, amelyek lefedik a kipróbálási stratégiának megfelelően a programgráfot.



Dinamikus tesztelés: fehér doboz módszerek



A tesztpredikátum előállítása:

Ehhez a program **szimbolikus végrehajtására** van szükség. Induljunk ki az előfeltételből! Haladjunk a programban az első elágazás- vagy ciklusfeltételig, s a formulát a közbülső műveleteknek megfelelően transzformáljuk! A tesztútnak megfelelő ág feltételét **és** kapcsolattal kapcsoljuk hozzá a tesztpredikátumhoz, majd folytassuk a szimbolikus végrehajtást egészen a program végpontjáig!



Dinamikus tesztelés: fehér doboz módszerek



Feladat: Egy N természetes szám valódi (1-től és önmagától különböző) osztója...

Utasítás lefedés:

- $i:=i+1$ végrehajtandó: $N=3$
- $O:=i$ végrehajtandó: $(\leftarrow Van=Igaz)$ $N=4$

$i:=2$	
$i < N$ és nem $i \mid N$	
$i:=i+1$	
$Van:=i < N$	
i	N
Van	
$O:=i$	—

Feltétel lefedés:

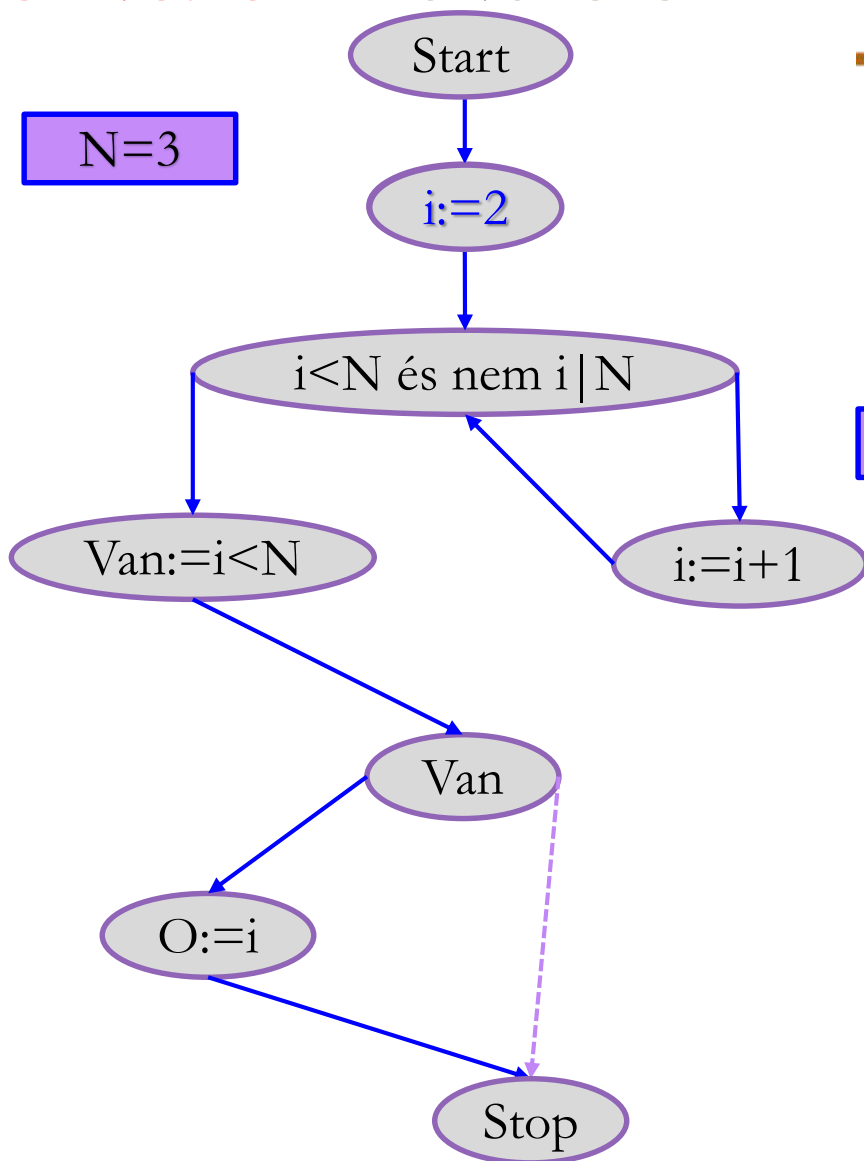
- Ciklusfeltétel igaz: $N=3$
- Ciklusfeltétel hamis: $N=2$ (be sem lép)
- Elágazásfeltétel igaz: $(\leftrightarrow Van=Igaz)$ $N=4$
- Elágazásfeltétel hamis: $(\leftrightarrow Van=Hamis)$ $N=2$



Dinamikus tesztelés: fehér doboz módszerek



Tesztút₁



2 < N és nem 2 | N

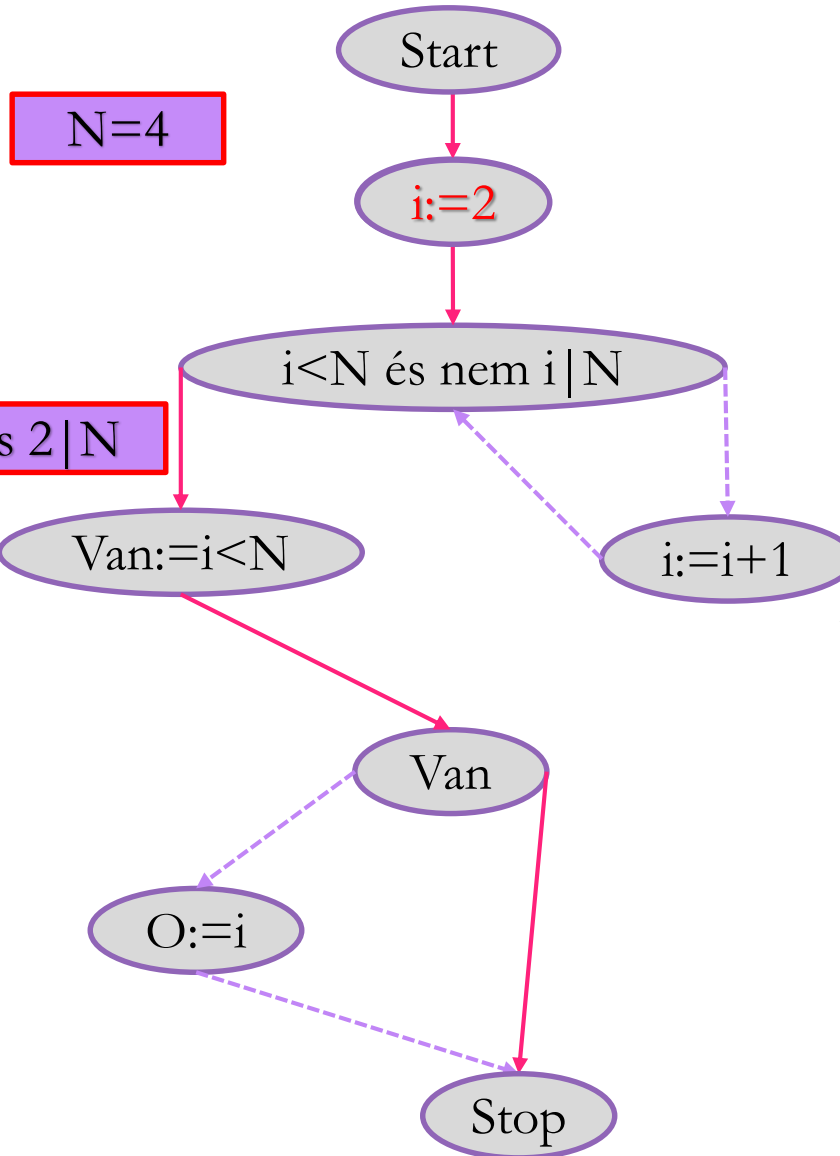
i:=2	
i < N és nem i N	
i:=i+1	
Van:=i < N	
Van	
O:=i	—



Dinamikus tesztelés: fehér doboz módszerek



Tesztút₂



$2 \geq N$ vagy $2 < N$ és $2 \mid N$

i:=2	
i < N és nem i N	
i:=i+1	
Van:=i < N	
Van	
O:=i	—

Automatikus tesztbemenet-
előállításához:

[http://people.inf.elte.hu/szla
vi/PrM1felev/Pdf/PrTea7.p
df](http://people.inf.elte.hu/szla
vi/PrM1felev/Pdf/PrTea7.p
df) 1.3.2. fejezetében



Speciális tesztelések

- Biztonsági teszt: ellenőrzések vannak?
- Hatékonysági teszt

Speciális programokhoz

- Funkcióteszt: tud minden funkciót?
- Stressz-teszt: gyorsan jönnek a feldolgozandók, ...
- Volumen-teszt: sok adat sem zavarja



Tesztelés **automatizálása**



Teszt-generálás:

- kézi
- automatikus (generáló program)
 - szabályos
 - véletlenszerű

Teszt-futtatás

- kézi
- automatikus (parancsfájl, be- és kimeneti állományok, automatikus értékelés)



Futtatás **adatifjll**al (C++)



Elv:

A standard input/output átirányítható fájlba. Ekkor a program **fájl**t használ az inputhoz és az outputhoz.

Következmény: **szerkezetileg a konzol inputtal/outputtal megegyező kell legyen / lesz a megfelelő fájl.**

„Technika”:

A lefordított kód mögé kell paraméterként írni a megfelelő fájlok nevét:

prog.exe **<inputfájl >outputfájl**

Figyelem! Ha van outputfájl, akkor a kérdés szövege is abban „jelenik meg”.

Nyereség:

Kényelmes és adminisztrálható tesztelés.

prog.exe **>>outputfájl**
outputfájl**hoz** írás!
45/61



Futtatás adatfájllal (C++)



Demo:

1. *Készítsünk néhány bemeneti adatot tartalmazó fájlt (a konzol inputnak megfelelő szerkezetben)!*
2. *Futtassuk ezekkel az előbb elmondottak szerint:*
 1. `prog.exe <1.be >1.ki`
 2. `prog.exe <2.be >2.ki`
 3. ...
3. *Ellenőrizzük a kimeneti fájlok tartalmát: olyan-e, amilyennek vártuk!*

Megjegyzés: tovább egyszerűsíthetjük a tesztelést, ha egy batch állománnyal automatizáljuk a 2.-at!

Valahogy így: próba₁, próba₂ ...

L. a csatolt
könyvtárban!

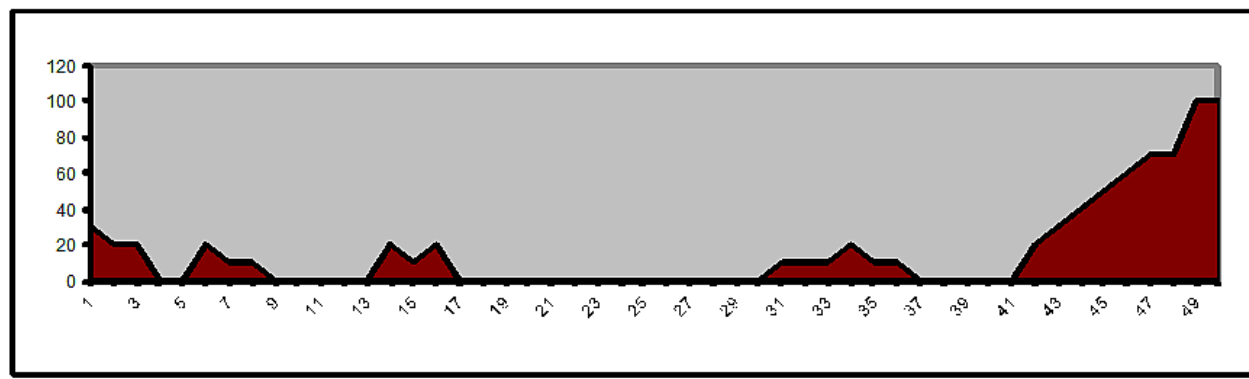
Kód
jegyzet-
ként

Tesztek előállítása



Feladat (teszteléshez):

Egy repülőgéppel Európából Amerikába repültünk. Az út során X kilométerenként mértük a felszín tengerszint feletti magasságát (≥ 0). 0 magasságot ott mértünk, ahol tenger van, >0 -t pedig ott, ahol szárazföld. Adjuk meg a szigeteket!



Tesztek előállítása

Specifikáció:

- Bemenet: $N \in \mathbb{N}$, $\text{Mag}_{1..N} \in \mathbb{N}^N$
- Kimenet: $\text{Db} \in \mathbb{N}$, $K_{1..N}, V_{1..N} \in \mathbb{N}^{\text{Db}}$
- ...

Tesztelés:

- **Kis** tesztek a tesztelési elveknek megfelelően, például:
 - $N=3$, $\text{Mag}=(1,0,1)$ → nincs sziget
 - $N=5$, $\text{Mag}=(1,0,1,0,1)$ → egy „rövid” sziget
 - $N=7$, $\text{Mag}=(1,0,1,0,1,0,1)$ → több „rövid” sziget
 - $N=7$, $\text{Mag}=(1,0,1,1,1,0,1)$ → hosszabb sziget
- Hogyan készítünk **nagy** (hatékonysági) tesztek?



Szabályos tesztek

Generálhatunk „szabályos” tesztek (egyszerű ciklusokkal).
Például így:

N:=1000	Változó i:Egész
i=1..10	
Mag[i]:=11-i	⇐ Európa
i=11..900	
Mag[i]:=0	⇐ tenger
i=901..N	
Mag[i]:=i-900	⇐ Amerika



Véletlen tesztek

(alapok – véletlenszámok)



A véletlenszámokat a számítógép egy algoritmussal állítja elő egy kezdőszámból kiindulva.

$$x_0 \rightarrow f(x_0)=x_1 \rightarrow f(x_1)=x_2 \rightarrow \dots$$

A „véletlenszerűséghez” megfelelő függvény és jó kezdőszám szükséges.

- **Kezdőszám:** (pl.) a belső órából vett érték.
- **Függvény** (az ún. lineáris kongruencia módszernél):
 $f(x) = (A \cdot x + B) \text{ Mod } M$,
ahol A, B és M a függvény belső konstansai.



Véletlen tesztek (alapok – C++)



C++: `rand()` véletlen egész számot ad `0` és egy maximális érték (`RAND_MAX`) között. `srand(szám)` kezdőértéket állít be.

➤ Véletlen($a..b$) $\in \{a, \dots, b\}$

```
v=rand() % (b-a+1)+a
```

➤ Véletlen(N) $\in \{1, \dots, N\}$

```
v=rand() % N+1
```

➤ véletlenszám $\in [0, 1) \subset \mathbb{R}$

```
v=rand() / (RAND_MAX+1.0)
```


A generátor használata kockadobásra:

```
#include <time.h>
...
srand(time(NULL));
i=rand() % 6 + 1;
```



Véletlen tesztek

Véletlen tesztekhez használjunk véletlenszámokat! Például így:

N:=1000	Változó i:Egész						
M:=Véletlen(9)							
i=1..M							
Mag[i]:=Véletlen(4..10)	⇐ Európa						
i=M+1..900							
<table><tr><td>I</td><td>véletlenszám<0.5</td><td>N</td></tr><tr><td>Mag[i]:=0</td><td>Mag[i]:=1</td><td></td></tr></table>	I	véletlenszám<0.5	N	Mag[i]:=0	Mag[i]:=1		⇐ tenger és szigetek
I	véletlenszám<0.5	N					
Mag[i]:=0	Mag[i]:=1						
i=901..N	⇐ Amerika						
Mag[i]:=Véletlen(2..8)							



Hibakeresés

Hibajelenségek a tesztelés során...

- hibás az eredmény,
- futási hiba keletkezett,
- nincs eredmény,
- részleges eredményt kaptunk,
- olyat is kiír, amit nem vártunk,
- túl sokat (sokszor) ír,
- nem áll le a program,
- ...



Hibakeresés

Célja:

a felfedett hibajelenség okának, helyének megtalálása.

Elvek:

- Eszközök használata előtt alapos végiggondolás.
- Egy megtalált hiba a program más részeiben is okozhat hibát.
- A hibák száma, súlyossága a program méretével nemlineárisan (annál gyorsabban!) nő.
- Egyformán fontos, hogy *miért nem csinálja* a program, amit *várunk*, illetve, hogy *miért csinál* olyat, amit *nem várunk*.
- Csak akkor javítani, ha megtaláltuk a hibát!



Hibakeresés

Hibakeresési eszközök (folytatás):

- Változó-, memória-kiírás (feltételes fordítás)
- Töréspont elhelyezése
- Lépésenkénti végrehajtás
- Adat-nyomkövetés
- Állapot-nyomkövetés (pl. paraméterekre vonatkozó előfeltételek, ciklus-invariánsok)
- Postmortem nyomkövetés: hibától visszafelé
- Speciális ellenőrzések (pl. indexhatár: `.at(.)↔[.]`)



Hibakeresés (C++)



Hibakeresési eszközök:



```
K:\proba\main.exe
Fuggveny: main, sor: 8
Assertion failed: a==0&&b==0, file K:\proba\main.cpp, line 9
```

- manuálisan – standard makrókkal (`__LINE__`, `__func__`, `assert`). Pl.

```
#include <iostream>
#include <cassert> //assert-hez
using namespace std;
int main()
{
    int a,b;
    cerr<<"Fuggveny: "<<__func__;//akt. függvény neve, most: main
    cerr<<", sor: "<<__LINE__<<endl;//akt. sorszám, most: 8
    assert(a==0&&b==0); //egy elvárás (a==0&&b==0) ellenőrzése;
                          //nem teljesülésekor megállás, hibaüzenet
    ...
}
```



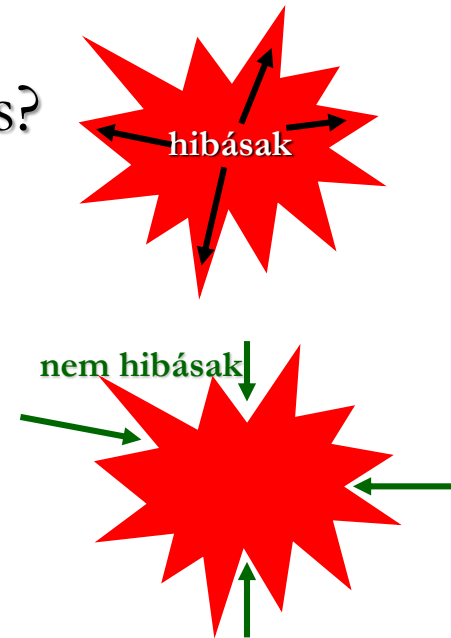
Hibakeresési módszerek

Célja:

- A **bemenetnek mely** része, amelyre hibásan működik a program?
- **Hol** található a **programban** a hibát okozó utasítás?

Módszerfajták:

1. Indukciós módszer (hibásak körének **bővítése**)
2. Dedukciós módszer (hibásak körének **szűkítése**)
3. Hibakeresés hibától visszafelé
4. Teszteléssel segített hibakeresés (olyan teszteset kell, amely az ismert hiba helyét fedí fel)



Hibakeresési módszerek

Példa az **indukciós** módszerre:



Feladat: *1 és 99 közötti N szám kiírása betűkkel.*

- Tesztesetek: $N=8 \Rightarrow$ jó, $N=17 \Rightarrow$ jó, $N=30 \Rightarrow$ hibás.
- Próbáljunk a hibásakból általánosítani: tegyük fel, hogy minden 30-cal kezdődőre rossz!
- Ha beláttuk (teszteléssel), akkor próbáljuk tovább általánosítani, pl. tegyük fel, hogy minden 30 felettire rossz!
- Ha nem lehet tovább általánosítani, akkor tudjuk mit kell keresni a hibás programban.
- Ha nem ment az általánosítás, próbáljuk másképp: hibás-e minden 0-ra végződő számra!
- ...



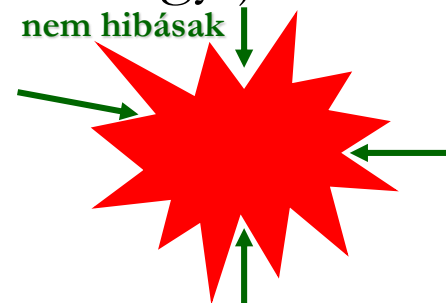
Hibakeresési módszerek

Példa a **dedukciós** módszerre:



Feladat: *1 és 99 közötti N szám kiírása betűkkel.*

- Tesztesetek: $N=8 \Rightarrow$ jó, $N=17 \Rightarrow$ jó, $N=30 \Rightarrow$ hibás.
- Tegyük fel, hogy minden nem jóra hibás!
- Próbáljunk a hibás esetek alapján szűkíteni:
tegyük fel, hogy a 20-nál kisebbekre jó!
- Ha beláttuk (teszteléssel), akkor szűkítsünk tovább, jó-e minden 39-nél nagyobbra?
- Ha nem szűkíthető tovább, akkor megtaláltuk, mit kell keresni a hibás programunkban.
- Ha nem, szűkítsünk másképp: tegyük fel, hogy jó minden nem 0-ra végződő számra!
- ...



Hibajavítás

Célja:

a megtalált hiba kijavítása.

Elvek:

- A hibát kell javítani és nem a tüneteit.
- A hiba kijavítása a program más részében hibát okozhat (rosszul javítunk, illetve korábban elfedett más hibát).
- Javítás után a tesztelés megismételendő!
- A jó javítás valószínűsége a program méretével fordítva arányos.
- A hibajavítás a tervezési fázisba is visszanyúlhat (a módszertan célja: lehetőleg ne nyúljon vissza).



➤ Tesztelés

- fogalmak + elvek
- statikus tesztelés
- dinamikus tesztelés
 - fekete doboz módszerek
 - szürke doboz módszerek
 - fehér doboz módszerek
- technika: futtatás adatfájlal – C++
- szabályos tesztek, véletlen tesztek

➤ Hibakeresés

- elvek + eszközök
- módszerek

➤ Hibajavítás

