17. Testování metodami bílé a černé skřínky. Strukturální, statická a dynamická analýza. Analýza datových toků. Zátěžové testy.

1. Co je testování

= verifikace + validace ~ testování bílé + černé skřínky

- automatizace testování umožňuje:

- běh regresních testů na nové verzi programu
- častější testování
- konzistence opakovatelnosti testů
- vícenásobné použití testu
- rychlý běh testů

Black box testování

- tester nemá informace o vnitřní struktuře testovaného kódu, nezná implementační detaily
- testuje se podle specifikace požadavků a podle požadavků
- např. testování GUI pomocí ručního klikání
- výstup se porovnává oproti očekávanému výstupu

White box testování

- tester má informace o vnitřní struktuře testovaného kódu a má k němu přístup, zná implementační detaily
- tester musí být také schopný programátor
- testuje se podél všech cest v programu (musí být pokryty všechny větve programu)
- tester se může zaměřit lépe na okrajové hodnoty
- např. unit testing
- neodhalí chybějící části SW oproti specifikaci

2. Optimalizace počtu testovacích případů

- nelze testovat vše => optimalizace

Ortogonální pole

- těžké najít, proto existují již hotové katalogy

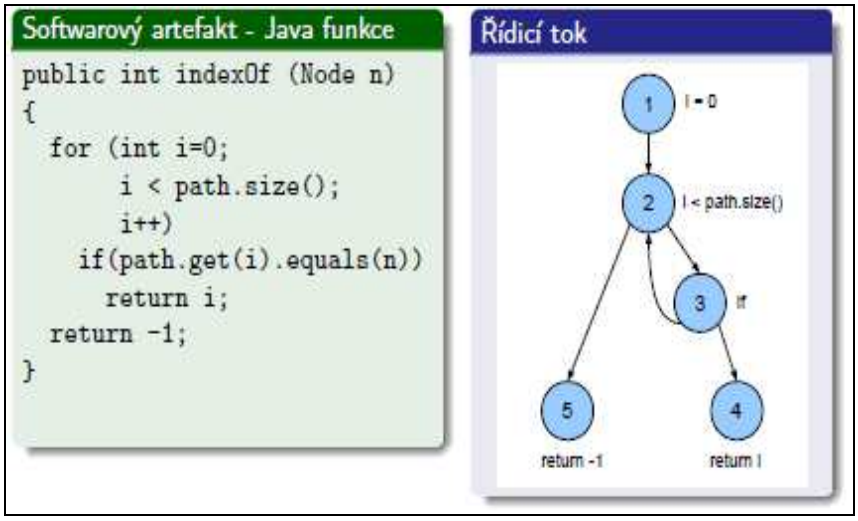
1. identifikovat faktory
2. spočítat úrovně jednotlivých faktorů
3. spočítat četnosti jednotlivých úrovní faktorů (např. 1x2 úrovně a 5x3 úrovně, což je pole 3^5 x 2^1)
4. nalézt co nejbližší větší ortogonální pole (např. 7^3 x 1^2, což je L18)
5. provedeme všech 18 testů s danými kombinacemi vstupů

Latinské čtverce

1. identifikovat faktory
2. zakódování hodnot faktorů
3. konstrukce čtverců
4. provedeme testy s vybranými kombinacemi vstupů

3. Strukturální testování

- tok programu lze převést na model



Obrázek 1 - Model funkce

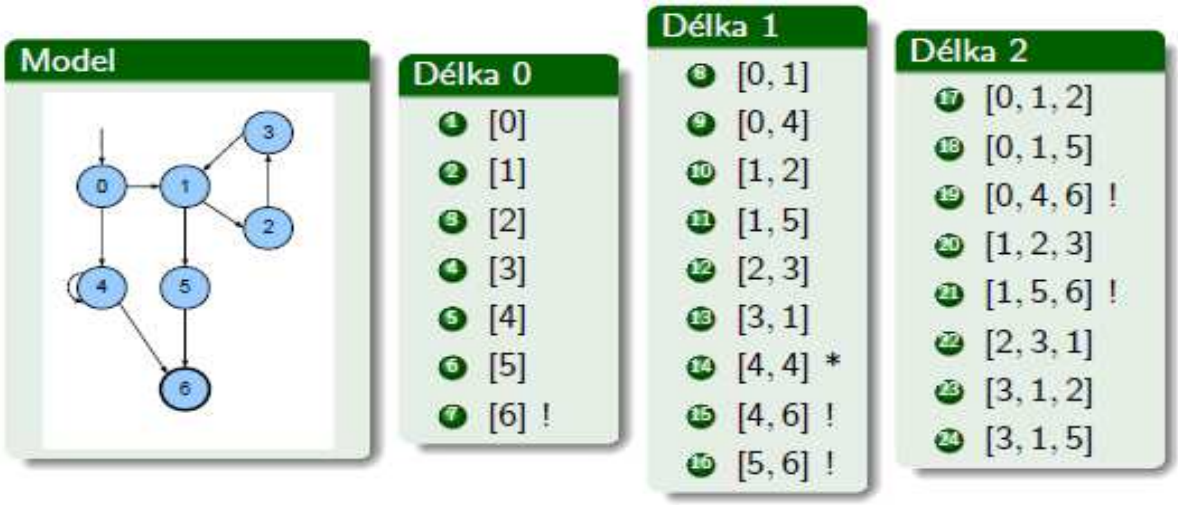
Testování cest

- **cesta** je sekvence operací, které se provedou od začátku běhu programu do jeho ukončení, tzv. úplná cesta
- **kritéria pokrytí** specifikují třídu cest, které by se měly provést v rámci testování, redukují množství testů
- **typy pokrytí**: pokrytí řádek, větví (každá podmínka musí být aspoň jednou pravdivá a aspoň jednou nepravdivá), pokrytí podmínek zkontroluje, úplné pokrytí cest (v praxi neproveditelné)

- **jednoduchá cesta**: cesta z uzlu i do j, na které se žádný uzel neobjevuje více jak jednou (počáteční a koncový uzel však může být totožný, proto jednoduchá cesta může být cyklus)
- **hlavní cesta**: cesta z i do j, jestliže je to jednoduchá cesta a není podcestou žádné jiné jednoduché cesty (je maximální)

Pokrytí řídícího toku (algoritmus nalezení hlavních cest)

1. Nalezni cesty délky 0 (uzly).
2. Kombinuj cesty délky 0 do cest délky 1 (hrany).
3. Kombinuj cesty délky 1 do cest délky 2.
4. ...

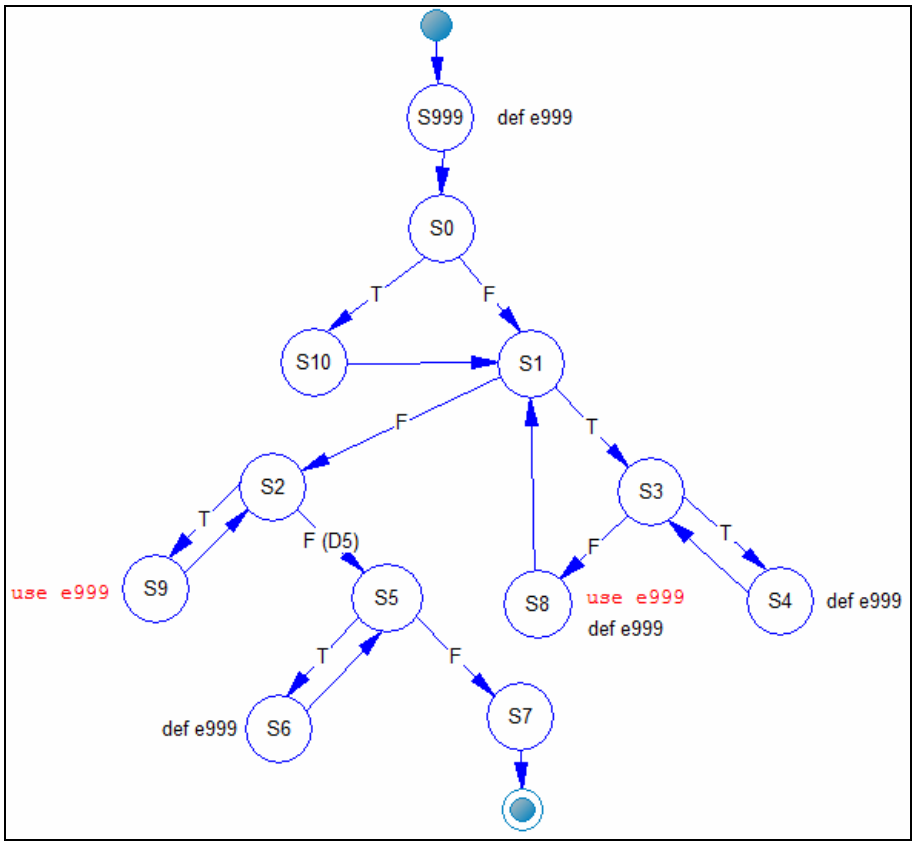


Pokrytí datového toku

- def : místo, kde je hodnota proměnné uložena do paměti
- use: místo, kde se přistupuje k hodnotě proměnné
- def (n) = podmnožina množiny proměnných V, které jsou definovány uzlem n
- use (n) = podmnožina množiny proměnných V, které jsou použity v uzlu n
- du(n, v) je množina všech du-cest vzhledem k proměnné v, která začíná v uzlu n

- pokrytí datového toku je vlastně nalezení všech du-cest pro vybranou proměnnou
- test zajišťuje, že **hodnoty vzniklé na jednom místě jsou použity správně na jiných místech**

- **příklad:**



def(S999)={e999}
def(S10)={e10}

use(S5)={d5}
du(S999, S8, e999)={[S999,S0,S1,S3,S8], [S999,S0,S1,S3,S4,S3,S8],
[S999,S0,S10,S1,S3,S8], [S999,S0,S10,S1,S3,S4,S3,S8]}

4. Statická analýza

= model checking (viz otázka 18) + testování konečného automatu

Testování konečného automatu

- výborný model pro testování **aplikací řízených pomocí menu**, široké použití v objektově orientovaném návrhu
- **pojmy** známé z jiných okruhů: automat, přechodová funkce, množina stavů, počáteční a koncový stav, přechod, vstup, výstup, přechodová tabulka
- množina stavů Q představuje hodnoty důležitých proměnných v systému, mód chování systému
- nedosažitelný stav znamená většinou chybu v návrhu
- většina modelů v praxi je silně souvislá
- **kontrola modelu:**
  - úplnost a konzistence (kontrola chybějících vstupů, nejednoznačnosti, rozpory)
  - jednoznačné kódování vstupů
  - model by měl být silně souvislý
- **pokrytí stavů** je taková minimální množina vstupů, při jejichž přijetí automat projde přes všechny stavy
- **pokrytí přechodů** je taková minimální množina vstupů, při jejichž přijetí automat projde přes všechny přechody

Input = {a; b}
L = {<>; b; b ::a; b ::a ::b}, <> je nulový vstup
T = {<>; a; b; b ::a; b ::b; b ::a ::a; b ::a ::b; b ::a ::b ::a; b ::a ::b ::b}
W = {a; b} ... charakterizační množina
Z = Input * W ∪ W = {a; b::a; b::a; b} = {a; b; a::a; a::b; b::a; b::b}
- L je množina vstupních sekvencí
- konečná množina testů je $T \bullet Z$

5. Dynamická analýza

= test běžící aplikace (AUT – application under test)
- např. metody chybného použití paměti (Rational Purify, Boundchecker)

6. Zátěžové testování

Podává důležité informace o **okrajových podmínkách** systému z hlediska **výkonu**. Výpadek systému může mít obrovské finanční a marketingové důsledky. Je dobré vědět, kde **hranice přetížení** leží. Dle těchto informací je možné dále řídit činnosti jako plánování marketingových kampaní (tak, aby to systém nepoložilo) nebo investice do výkonového rozšiřování systému.

Zátěžové testování je možné realizovat v několika krocích: Deklarovat současnou maximální zátěž. Vytvořit testovací scénáře pro zátěžové testování. Využít existující testovací nástroj. Provést výkonové testování, **monitorovat odezvu systému**. Analyzovat dopady a řešení – optimalizace kódu, optimalizace procesů, db connection pooling, posílení hardwaru, zvětšení propustnosti kanálů. **Nástroje** JMeter, JVisual VM.