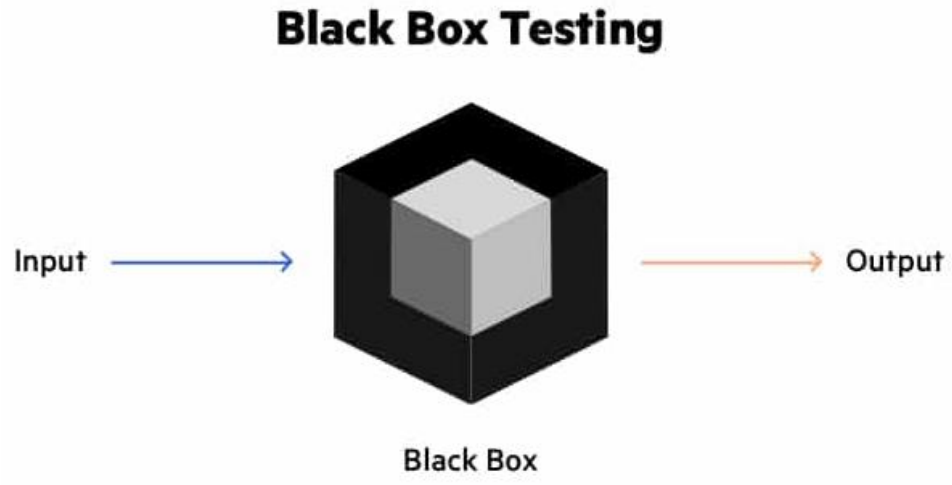
Tehnike crne kutije

FUNKCIONALNO TESTIRANJE

Funkcionalno testiranje

- Program se posmatra kao funkcija koja mapira vrednosti iz ulaznog domena u izlazni. Implemenacija nije poznata, program predstavlja "crnu kutiju" (kao što se tako kaže za neuronske mreže).
- Jedina informacija koja se koristi za određivanje test primera je specifikacija programa (dokumentacija).



Testiranje tehnikama crne kutije

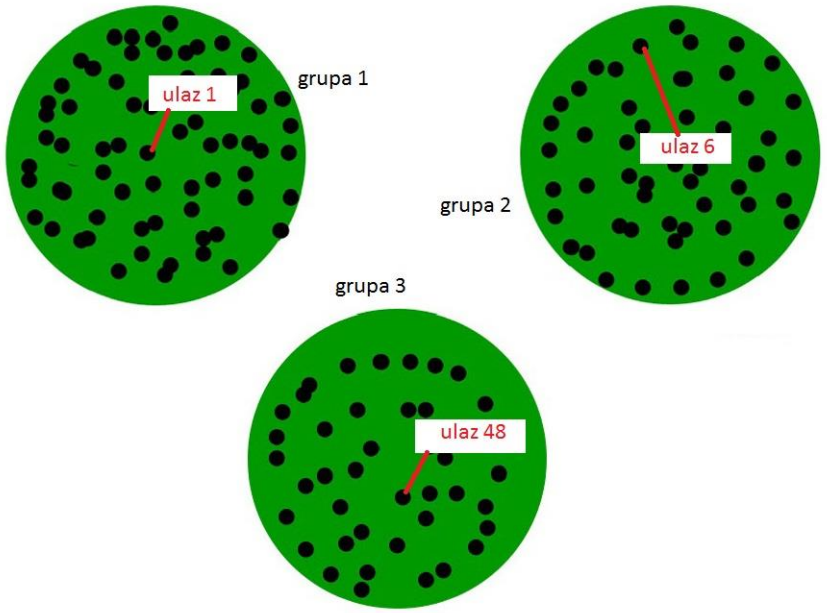
- Podela na klase ekvivalencije
- Analiza graničnih vrednosti
- Tabela odlučivanja i uzročno-posledični graf
- Testiranje zasnovano na modelu stanja (preskočićemo ove godine)
- Nagađanje grešaka
- Testiranje sintakse
- Kombinatorno testiranje
  - Zasnovano na ortogonalnim nizovima

Podela na klase ekvivalencije

Ulazne vrednosti programa se dele u klase ekvivalencije.

Podela se vrši tako da važi:

- program se ponaša na sličan način za sve ulazne vrednosti koje pripadaju istoj klasi ekvivalencije.
- Grupe međusobno ekvivalentnih ulaznih podataka proizvode isti rezultat iz kojih se bira po jedan predstavnik grupe koji se koristi u tokom testiranja.



Zašto definišemo klase ekvivalencije?

- Da bismo testirali program sa po jednom reprezentativnom vrednošću ulaza iz svake klase ekvivalencije
  - Jednako delotvorno kao testiranje bilo kojim drugim vrednostima iz istih klasa ekvivalencije (naći će se iste greške).
- U idealnom slučaju podskupovi su međusobno disjunktni i pokrivaju ceo skup ulaza (relacija "sličnosti" ulaza je relacija ekvivalencije)

Kako odrediti klase ekvivalencije?

- Posmatraju se svi uslovi vezani za ulaze i izlaze programa koji proizilaze iz specifikacije (tipovi promenljivih, eksplicitna ograničenja).
- **Za svaki uslov se posmatraju dva grupe klasa prema zadovoljenosti uslova**:
  - *legalne klase* - obuhvataju dozvoljene situacije.
  - *nelegalne klase* - obuhvataju sve ostale situacije.

Saveti za podelu na klase ekvivalencije

Ako je ulazni uslov programa definisan u opsegu vrednosti:

- Na primer, broj između **18** i **56**.
- Definišu se:
  - **jedna legalna** (18 ≤ broj ≤ 56) i
  - **dve nelegalne klase ekvivalencije**: (broj < 18) i (broj > 56)

AGE: [ ] (accepts 18 to 56)

EQUIVALANCE PARTITIONING		
INVALID	VALID	INVALID
<=17	18-56	>=57

Saveti za podelu na klase ekvivalencije

Ako ulazni uslov programa definiše neki fiksni broj:

- Na primer, matični broj građanina ima 13 cifara.
- Definišu se:
  - jedna legalna (mbr_c=13) i
  - dve nelegalne klase ekvivalencije: (mbr_c < 13) i (mbr_c > 13)

Ako ulazni podatak uzima vrednosti iz nabrojivog skupa, pri čemu se program različito ponaša za svaku vrednost:

- Na primer, {a, b, c}
- Po jedna klasa za svaku dozvoljenu vrednost ulaza (tri klase)
- Jedna klasa za ulazne vrednosti van dozvoljenog skupa (npr. d).

Table 2.1 Guidelines for generating equivalence classes for variables: range and strings

Kind	Equivalence classes	Example	
		Constraint	Class representatives[a]
Range	One class with values inside the range and two with values outside the range	$speed \in [60 \ldots 90]$	{{50}↓,{75}↑, {92}↓}
		area : float; $area \geq 0$	{{−1.0}↓,{15.52}↑}
		age : int; $0 \leq age \leq 120$	{{−1}↓,{56}↑, {132}↓}
		letter:char;	{{J}↑,{3}↓}

String	At least one containing all legal strings and one containing all illegal strings. Legality is determined based on constraints on the length and other semantic features of the string	*fname*: *string*;	{{ϵ}↓,{Sue}↑, {Sue2}↓,{Too Long a name}↓}
		vname: string;	{{ϵ}↓,{shape}↑, {address1}↑}, {Long variable}↓

Table 2.2 Guidelines for generating equivalence classes for variables: Enumeration and arrays

Kind	Equivalence classes	Example[a] Constraint	Example[a] Class representatives[b]
Enumeration	Each value in a separate class	`auto_color` ∈ {red, blue, green}	{{red}↑, {blue}↑, {green}↑}
		`up`: boolean	{{true}↑, {false}↑}
Array	One class containing all legal arrays, one containing only the empty array, and one containing arrays larger than the expected size	*Java array*: int[] *aName* = new int[3];	{{]}↓, {[−10, 20]}↑, {[−9, 0, 12, 15]}↓}

Saveti za podelu na klase ekvivalencije

- Za složene tipove podataka (zapise)
  - Odrediti klase ekvivalencije za svaku komponentu posebno, a po potrebi razmotriti kombinacije klasa ekvivalencije
- Generalno, ako postoji sumnja da se program ne ponaša isto za svaki element već definisane klase ekvivalencije, treba klasu razbiti na više manjih.

- Ako razmatramo svaki uslov izolovano od drugih i definišemo njegove klase ekvivalencije, tada je reč je o tzv. **jednodimenzionalnom particionisanju.**
- Ako razmatramo kombinacije klasa ekvivalencije parova ili većeg broja uslova, radi se **multidimenzionalnom particionisanju.**

Multidimenzionalno particioniranje

Razmotrimo program koji prima dva cela broja **x** i **y**, uz ograničenja:

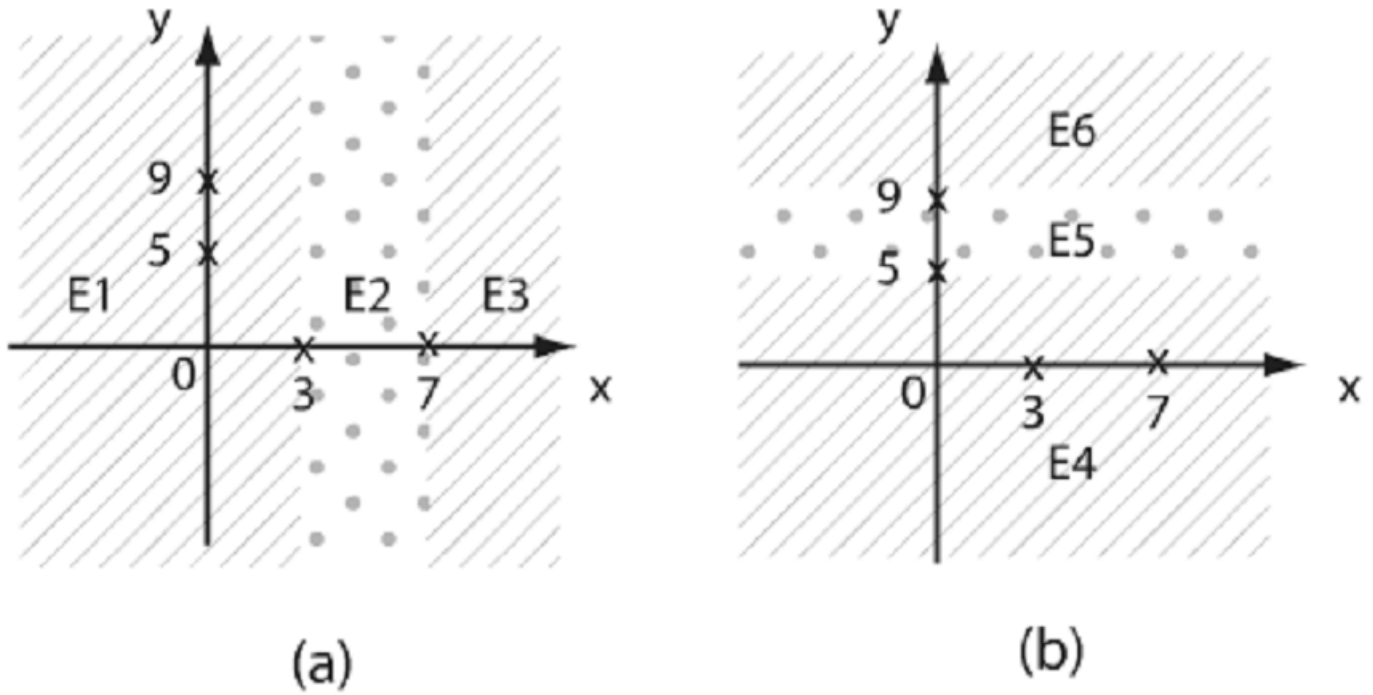
$3 \leq x \leq 7$ $5 \leq y \leq 9$

Jednodimenziono particionisanje (posmatramo **x** i **y** svaki za sebe) daje sledeće klase ekvivalencije:

E1: $x < 3$	E4: $y < 5$
E2: $3 \leq x \leq 7$	E2: $5 \leq y \leq 9$
E3: $x > 7$	E3: $y > 9$

Multidimenzionalno particioniranje (primer)

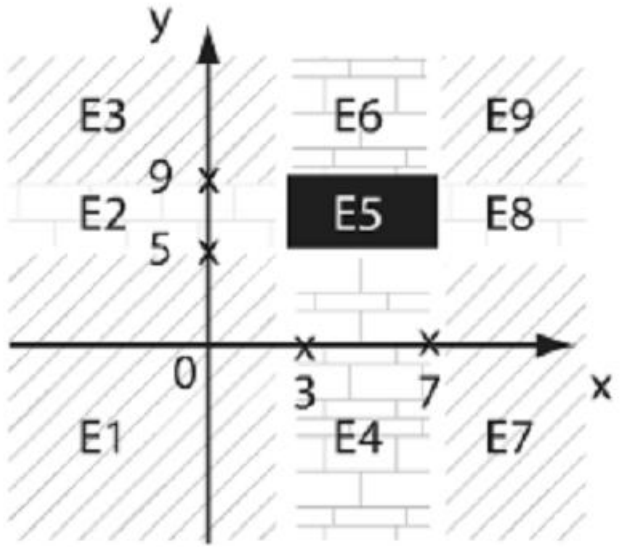
Grafička predstava



Multidimenzionalno particioniranje (primer)

Ako se uzmu kombinacije klasa ekvivalencija za X i Y:

E1: $x < 3, y < 5$	E2: $x < 3, 5 \leq y < 9$	E3: $x < 3, y > 9$
E4: $3 \leq x \leq 7, y < 5$	E5: $3 \leq x \leq 7, 5 \leq y \leq 9$	E6: $3 \leq x \leq 7, y > 9$
E7: $x > 7, y < 5$	E8: $x > 7, 5 \leq y \leq 9$	E9: $x > 7, y > 9$



Ako se primeni multidimenziono particionisanje program će biti detaljnije testiran.

Mana je što u slučaju N uslova dolazi do kombinatorne eksplozije. To se može **rešavati tehnikama kombinatornog testiranja.**

Identifikacija test primera iz klasa

1. Dodeliti jedinstveni broj svakoj klasi.
2. Napisati test primere za sve legalne klase uključujući u jedan test što više klasa.
3. Napisati po jedan test primer za svaku nelegalnu klasu zasebno (da bi se izbeglo da jedan nelegalan ulaz maskira drugi zbog ugrađenih provera u programu).

Primer podele na klase ekvivalencije

Dat je potprogram IME koji ispituje ispravnost imena datoteke. Sintaksa imena je data sa:

<ime datoteke> ::= <ime> [.<tip>],

<ime>: dužine 1-6, sadrži samo alfanumeričke vrednosti, počinje slovom

<tip>: dužine 0-3, sadrži samo alfanumeričke vrednosti,

Ukoliko se izostave tačka i tip, podrazumeva se .DAT.

Potprogram postavlja kod ishoda sa značenjem:

- 0 - neispravno ime datoteke,
- 1 - ispravno ime datoteke, ima tačku ali nema tip;
- 2 - ispravno ime datoteke, sa podrazumevanim tipom .DAT,
- 3 - ispravno ime sa zadatim tipom.

Primer (nastavak)

Legalne klase ekvivalencije	Nelegalne klase ekvivalencije
Ime:	**Ime:**
1. Ima 1-6 znakova	2. Ima 0 znakova
4. Sadrži samo alfanumeričke karaktere	3. Ima više od 6 znakova
7. Počinje slovom	5. Ime sadrži spec. znakove
	6. Ime sadrži blank, tabove
Tačka i tip:	8. Ime počinje cifrom
10. Naveden je iza imena	9. Ime počinje spec. znakom
12. Postoji tačka, a nema tipa	
13. Ne postoje ni tačka ni tip	**Tačka i tip:**
14. Tip ima od 0-3 znaka	11. Postoji ime i tip, a nema tačke
16. Tip je alfanumerički	15. Tip ima više od tri znaka
	17. Tip sadrži i druge spec. znakove

Primer (nastavak)

Testovi za legalne klase:

- Proba.txt pokriva 1, 4, 7, 10, 14, 16
- Proba. pokriva još 12
- Proba pokriva još 13

Za svaku nelegalnu klasu treba poseban test:

2. .bla
3. Dugackoime.log
5. %#,,$^$#.grr
6. Ime i prezime.dat
8. 14343
9. ..doc
11. ime tip
15. tra.lalala
17. dokument.$$$

Legalne klase ekvivalencije

Ime:

1. Ima 1-6 znakova
4. Sadrži samo alfanumeričke karaktere
7. Počinje slovom

Tačka i tip:

10. Naveden je iza imena
12. Postoji tačka, a nema tipa
13. Ne postoje ni tačka ni tip
14. Tip ima od 0-3 znaka
16. Tip je alfanumerički

Nelegalne klase ekvivalencije

Ime:

2. Ima 0 znakova
3. Ima više od 6 znakova
5. Ime sadrži spec. znakove
6. Ime sadrži blank, tabove
8. Ime počinje cifrom
9. Ime počinje spec. znakom

Tačka i tip:

11. Postoji ime i tip, a nema tačke
15. Tip ima više od tri znaka
17. Tip sadrži i druge spec. znakove

Analiza graničnih vrednosti

Programeri često previđaju posebnu obradu koja je potrebna na granicama klasa ekvivalencije

Na primer, programer može nepravilno napisati < umesto <=

Analiza graničnih vrednosti:

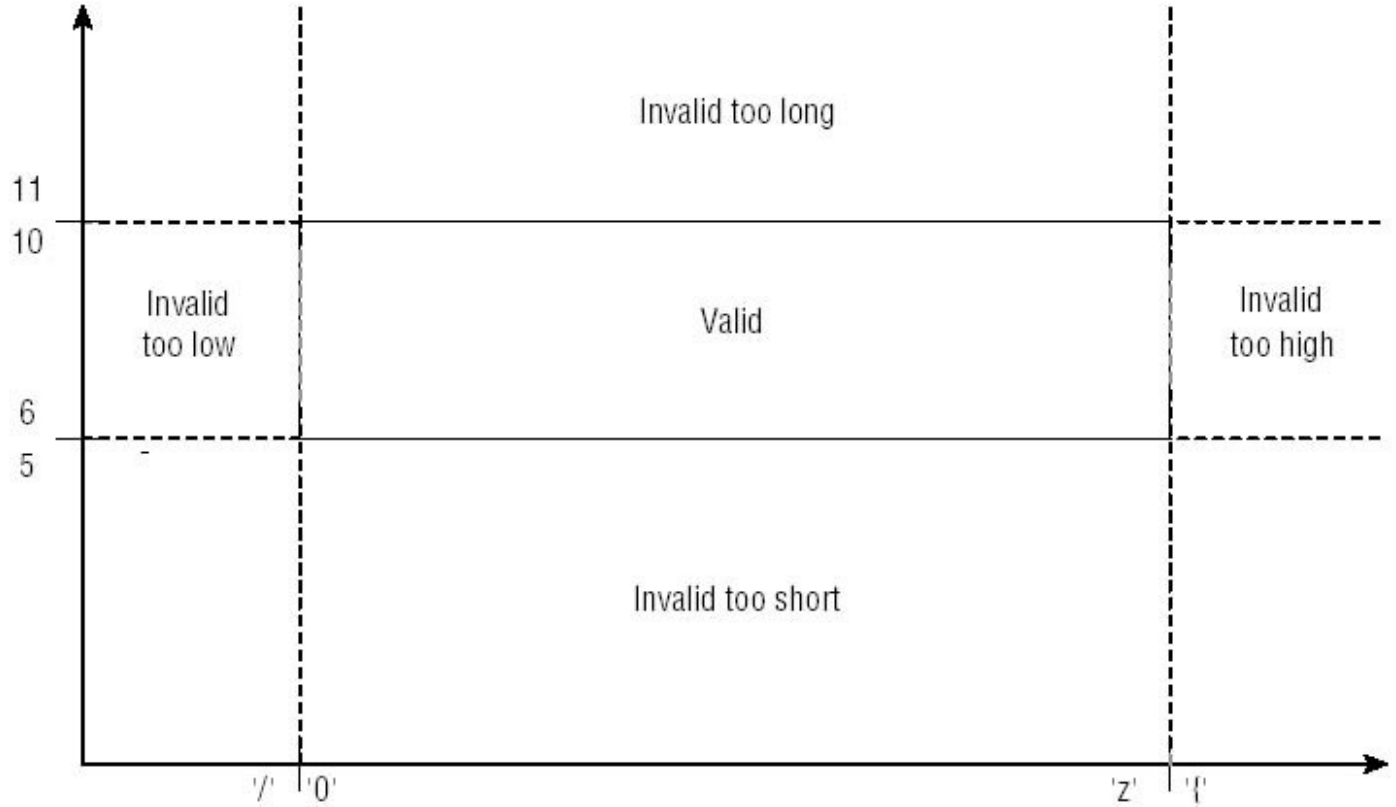
- Izabrati test primere na granicama različitih klasa ekvivalencije (jedno ili multidimenzionalnih), ili na osnovu specificiranih veza među ulazima
- Po jedan test primer za svaku graničnu vrednost

Majersove preporuke za granične vrednosti

1. Ako ulazni uslov precizira opseg vrednosti, napisati testove za same krajeve opsega i testove nevažećih ulaza za situacije odmah iza legalnih krajeva. PRIMER: Ako je ulaz u opsegu -1.0 do 1.0 testirati treba za -1.0, 1.0, -1.0001, 1.0001 (ako se pretpostavi da je najmanja delta između dva broja koja pravi neku razliku u programu .0001).
2. Ako ulazni uslov precizira ukupan broj vrednosti, napisati testove za minimalni i maksimalni broj vrednosti i jedan ispod i iznad ovih vrednosti. Na primer, ako ulazni fajl može da sadrži 1-255 zapisa, napisati testove za 0, 1, 255, i 256 zapisa.
3. Primeniti preporuku 1. na izlazne uslove.
4. Primeniti preporuku 2. na izlazne uslove. PRIMER: ako na stranu izveštaja staje 65 redova, napisati testove za 64, 65 i 66 redova.
5. Ako je ulaz (ili izlaz) programa uređen skup (sekvencijalni fajl, na primer, ili linearna lista ili tabela), fokusirati pažnju na prvi i poslednji element skupa.
6. Upotrebiti sopstvenu pamet za nalaženje drugih graničnih uslova.

Dalji primer – BVA za stringove

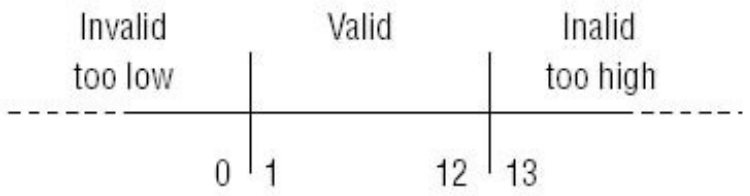
- npr. ako se traži lozinka dužine od 6 do 10 karaktera:

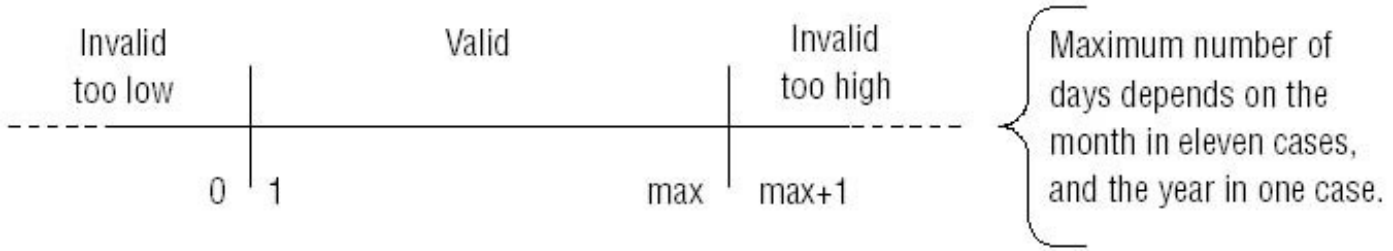


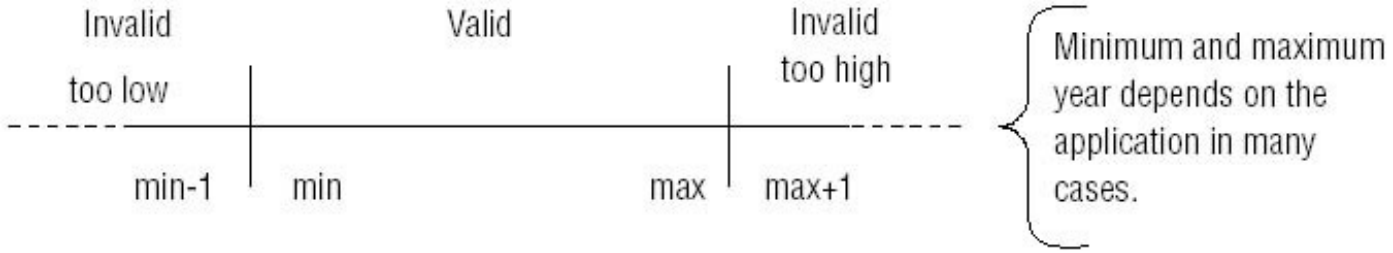
Datumi

Treba voditi računa i o formatu datuma, internoj reprezentaciji (Y2K problem, Unix brojač ide od Jan 1, 1970.)

MM/DD/YY



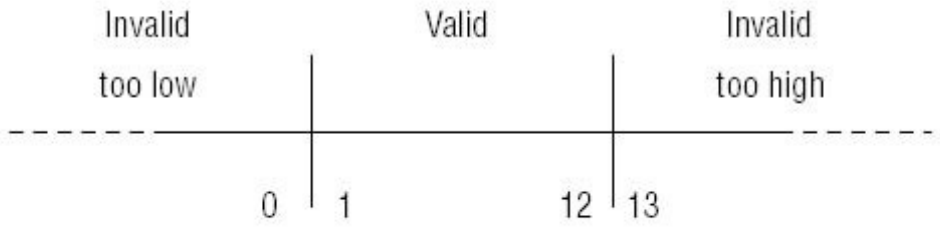


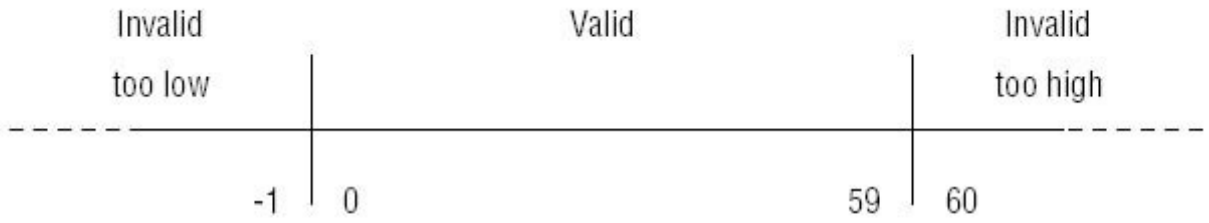


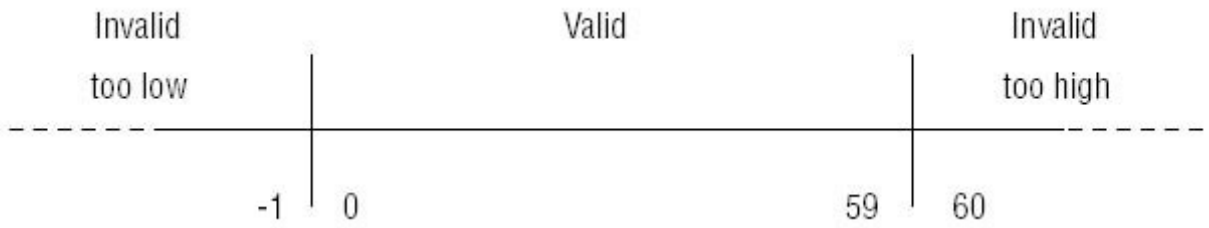
HH:MM:SS

Vreme

Treba voditi računa i o formatima, vremenskim zonama, pomeranju vremena...



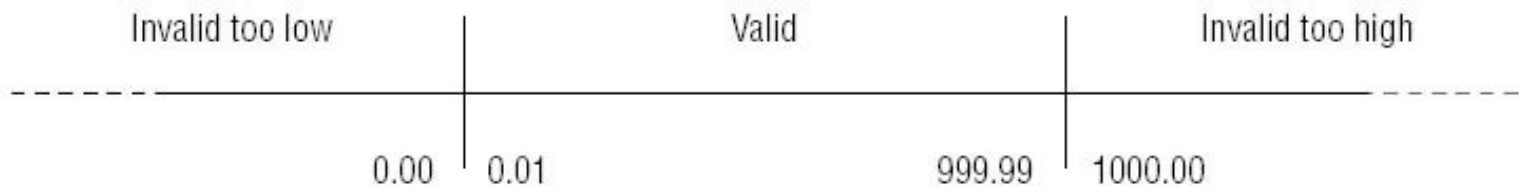




Novčane sume

Često se reprezentuju kao decimalni brojevi sa fiksnim zarezom, znači imaju dobro definisan epsilon.

Treba voditi računa i o valuti, formatu pisanja.



Primer analize graničnih vrednosti

- Treba testirati metod *textSearch* koji vrši pretragu nepraznog stringa ***s*** u tekstu ***txt.***
- Pozicije znakova u *txt* počinju od 0.
- Ulazni parameteri u *textSearch* su *s* i *txt.*
- Metod vraća ceo broj *x.*
- Ako je $x \geq 0$ onda *x* označava startnu poziciju stringa *s* unutar *txt.*
- Negativna vrednost *x* označava da *s* nije nađen unutar *txt.*

Primer

- Identifikujemo klase ekvivalencije na osnovu ulaza
- Min. dužina stringova je 0 i to je donja granica (gornja nije specifikacijom ograničena)

Klase ekvivalencije za **s**: E_1: prazan string, E_2: neprazan string
Klase ekvivalencije za **txt**: E_3: prazan string, E_4: neprazan string

- Na osnovu izlaza programa dobijamo još dve klase ekvivalencije:
- Graničnu vrednost x=0 dobijamo kada se s nalazi u txt-u i to na samom početku.

$E_5: x < 0$, $E_6 : x \geq 0$

Klase ekvivalencije za **s**: E_1: prazan string, E_2: neprazan string
Klase ekvivalencije za **txt**: E_3: prazan string, E_4: neprazan string
E_5: $x < 0$, E_6 : $x \geq 0$

Primer...

- Sada možemo osmisliti testove za klase ekvivalencije i do sada uočene granične vrednosti:

$$
\begin{aligned}
T = \{ & t_1 : (s = \epsilon, \\
& \quad txt = \text{“Laughter is good for the heart.”}), \\
& t_2 : (s = \text{“Laughter”}, \ txt = \epsilon), \\
& t_3 : (s = \text{“good for”}, \\
& \quad txt = \text{“Laughter is good for the heart.”}), \\
& t_4 : (s = \text{“Laughter”}, \\
& \quad txt = \text{“Laughter is good for the heart.”})\}
\end{aligned}
$$

- t_1 i t_2 pokrivaju klase ekvivalecije E_1 do E_5, t_3 pokriva E_6.
- t_4 eksplicitno pokriva granični uslov x=0 koji smo identifikovali.

Pokrili smo sve klase, ali nastavljamo sa dodatnim testovima!

Klase ekvivalencije za **s**: E_1: prazan string, E_2: neprazan string
Klase ekvivalencije za **txt**: E_3: prazan string, E_4: neprazan string
E_5: $x < 0$, E_6 : $x \geq 0$

Primer...

- Možemo kao granični uslov dodati test kada se traženi string nalazi na samom kraju:

t_5 : (*s* = "heart.",
txt = "Laughter is good for the heart.")

- Sada smišljamo testove blizu granica:

- *s* starts at position 1 in *txt*. Expected output: $p = 1$.
- *s* ends at one character before the last character in *txt*. Expected output: $p = k$, where k is the position from where *s* starts in *txt*.
- All but the first character of *s* occur in *txt* starting at position 0. Expected output: $p = -1$.
- All but the last character of *s* occur in *txt* at the end. Expected output: $p = -1$.

Klase ekvivalencije za **s**: E_1: prazan string, E_2: neprazan string
Klase ekvivalencije za **txt**: E_3: prazan string, E_4: neprazan string
E_5: $x < 0$, E_6 : $x \geq 0$

Primer...

t_6 : (s = "Laughter",
txt = "Laughter is good for the heart.",)
t_7 : (s = "heart",
txt = "Laughter is good for the heart.",)
t_8 : (s = "gLaughter",
txt = "Laughter is good for the heart."),
t_9 : (s = " heard.",
txt = "Laughter is good for the heart.".)

Klase ekvivalencije za **s**: E_1: prazan string, E_2: neprazan string
Klase ekvivalencije za **txt**: E_3: prazan string, E_4: neprazan string
E_5: $x < 0$, E_6 : $x \geq 0$

Primer...

Možemo još probati da kombinujemo klase za **s** i **txt**:

$$E_1 \times E_3,\ E_1 \times E4,\ E_2 \times E3,\ E_2 \times E4.$$

Testovi t_1, t_2 i t_3 pokrivaju sve klase izuzev E_1 x E_3. Dodatni test pokriva i ovo:

$$t_{10}: (s = \epsilon,\ txt = \epsilon)$$

Nagađanje grešaka

- Tehnike crne kutije opisane do sada su formalne, u smislu da postoje jasna pravila kako se testiranje sprovodi i kako se vrši odabir testova.
- Svima im je zajedničko da je specifikacija softvera jedini izvor informacija za formiranje testova, odnosno da ne postoji uvid u kod.
- *Nagađanje grešaka* (*Error guessing*) - neformalna tehnika testiranja koja se često primenjuje u praksi
  - Obično su za nju zaduženi iskusni i dobri testeri
  - Rade je ljudi koji imaju bogato testersko iskustvo ili dugo rade sa određenim sistemom pa lakše mogu da uoče mane tog sistema
  - Tester će onda moći da bolje pretpostavi gde su potencijalne greške ili situacije u kojima se sistem neće ponašati na očekivan način. Na primer,
    - pokušaj deljenja sa nulom,
    - prazno polje za unos,
    - prazni fajlovi i
    - različite vrste pogrešnih podataka – na primer, alfabetski karakteri u polju za unos na mestu gde se očekuju numerički i slično.
  - Neke situacije neće nikada doći – pogrešna pretpostavka!