PROVES

Tema 8

Enginyeria del Programari

ETS Ingenieria Informàtica
DSIC – UPV

Curs 2021-2022

Objetius

- Conèixer les tècniques bàsiques de prova de programes

- Ser capaços de dissenyar casos de prova per a un mòdul o funció utilitzant les tècniques
  - **camí bàsic** i
  - **partició equivalent.**

Continguts

- Introducció a la prova de programari.
- Tècniques de disseny de casos de prova.
- Prova de caixa blanca: el camí bàsic.
- Prova de caixa negra: la partició equivalent.
- Eines automàtiques de prova.

Bibliografia

- SOMMERVILLE, I., Software Engineering, 7ª Edición. Addison Wesley, 2005
- PFLEEGER, S. L., Enginyeria del Programari: Teoría y Práctica. Prentice Hall, 2002.
- PRESSMAN, R. Ingeniería del software. Un enfoque práctico. 6ª Edición, McGraw-Hill, 2006.
- COLLARD, J.F, BURNSTEIN, I, Practical Software Testing: A Process-Oriented Approach, Springer. 2003
- EVERETT, D., McLEOD, R. Software Testing. Testing Across the Entire Software Development Life Cycle, IEEE Press
- BEIZIER, B., Testing and quality assurance, von Nostrand Reinhold, New York, 1984

Introducció

- Proves: factor crític per a determinar la qualitat del programari

- La Prova de Programari pot definir-se com una activitat en la qual un sistema o un dels seus components s'executa en circumstàncies prèviament especificades, els resultats s'observen i registren, i es realitza una avaluació d'algun aspecte: correcció, robustesa, eficiència, etc.

- Cas de prova (test case): *«un conjunt d'entrades, condicions d'execució i resultats esperats desenvolupats per a un objectiu particular»*

Principis bàsics

- **Principi 1**: *Testing* és el procés d'executar un component programari utilitzant un conjunt bàsic de casos de prova, amb la intenció de (1) revelar defectes, i (2) avaluar la qualitat
- **Principi 2**: Quan l'objectiu de la prova és detectar defectes, llavors un bon cas de proves serà aquell amb una major probabilitat de detectar un defecte encara no detectat
- **Principi 3**: Els resultats de les proves haurien de ser meticulosament inspeccionats
- **Principi 4***: Un cas de prova ha de contenir els resultats esperats
- **Principi 5***: Els casos de prova haurien de ser definits tant per a condicions d'entrada vàlides com no vàlides

Principis bàsics

- **Principi 6**: La probabilitat de l'existència addicional de defectes en un component programari és proporcional al nombre de defectes ja trobats en aqueix component
- **Principi 7***: Proves haurien de ser dutes a terme per un grup que siga independent del grup de desenvolupament
- **Principi 8***: Les proves han de ser repetibles i reutilitzables
- **Principi 9***: Les proves han de ser planejades
- **Principi 10:** Les activitats de proves haurien d'estar integrades amb la resta del cicle de vida
- **Principi 11**: *Testing* és una activitat creativa i desafiadora

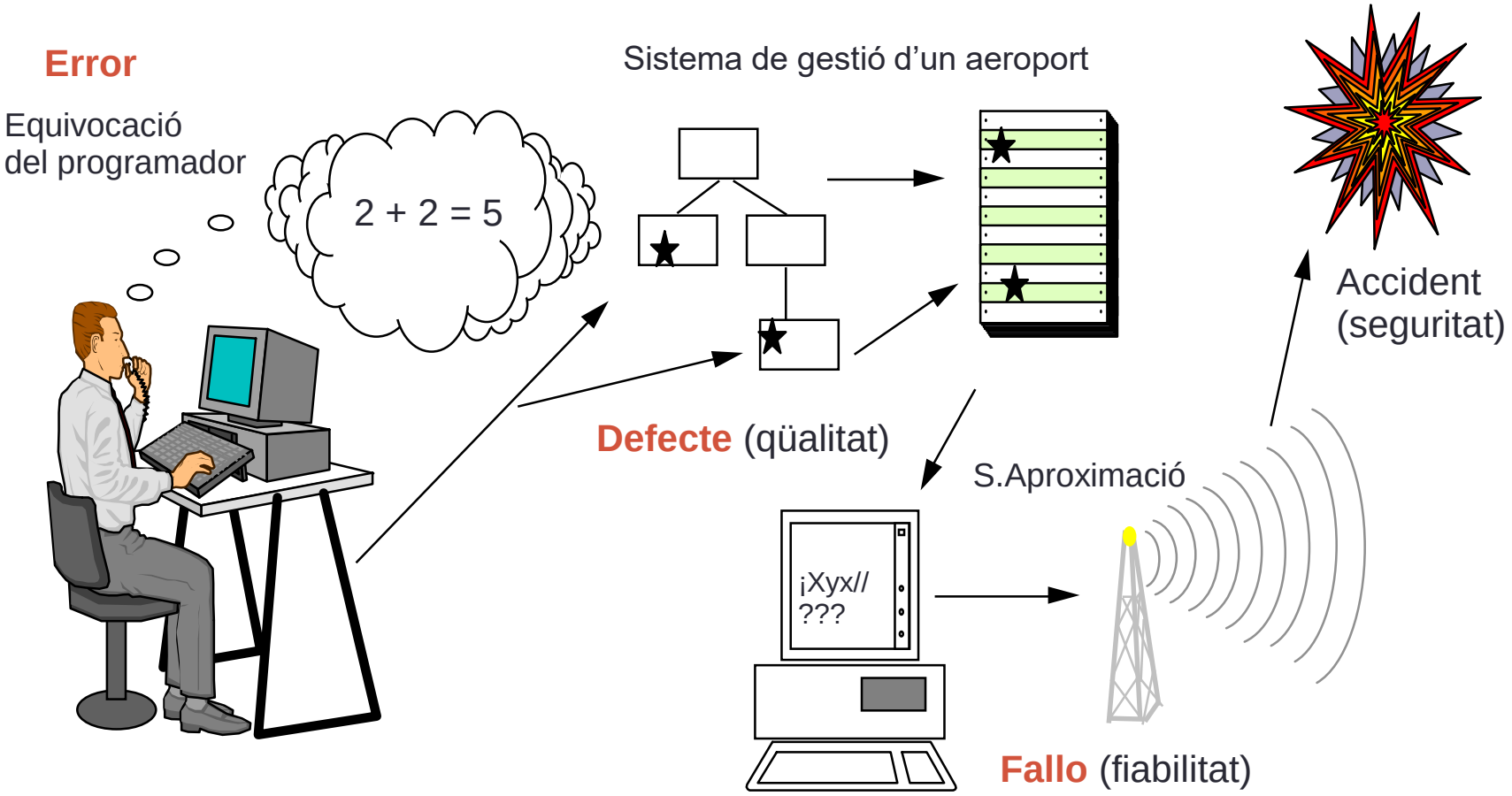
Visió Google de les proves

Definicions

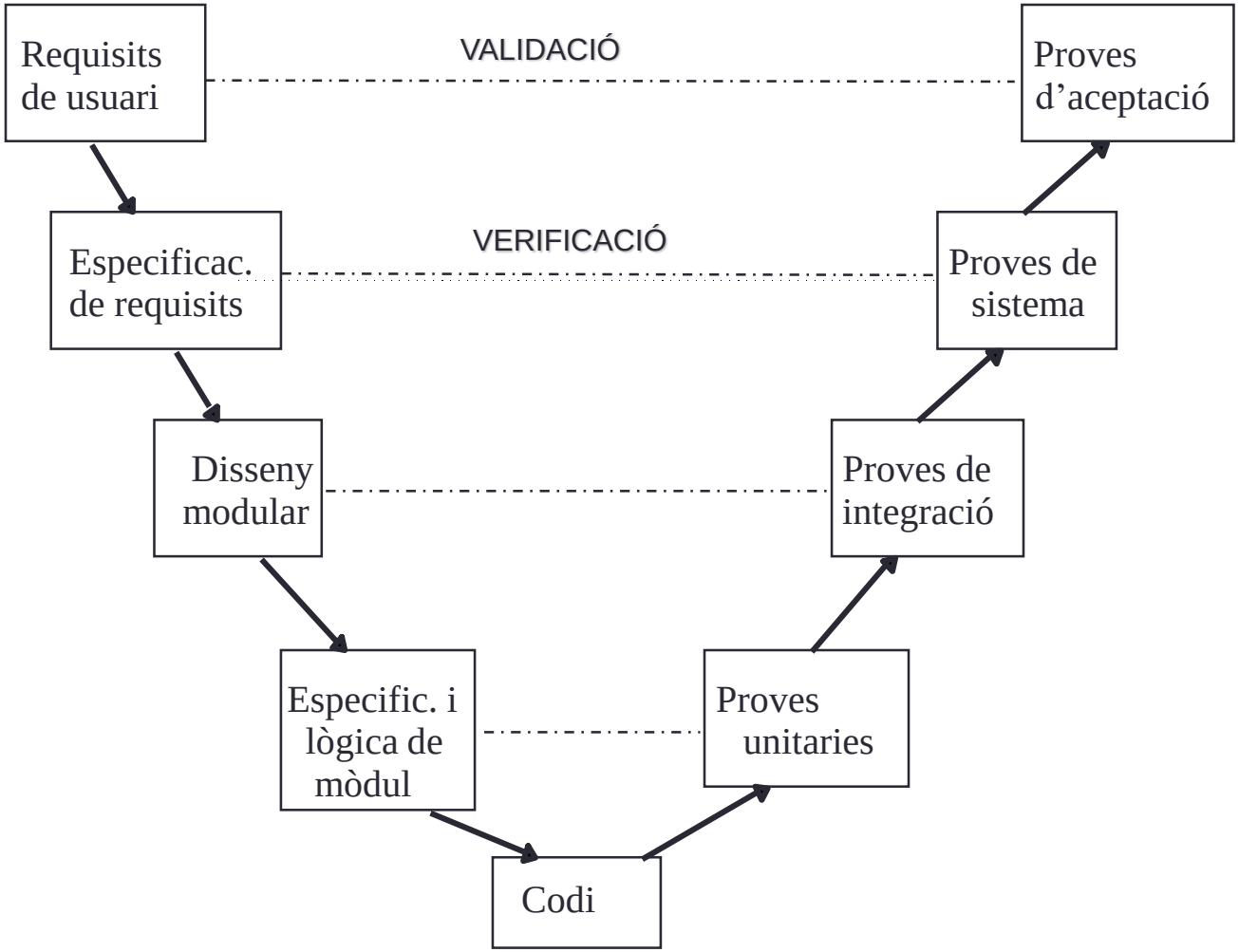
- Verificació: estem construint correctament el producte? el programari s'ha construït d'acord a les especificacions?
- Validació: estem construint el producte correcte? el programari fa lo que el usuari realment requereix?

- Defecte (***bug***): una anomalia en el programari com, per exemple, un procés, una definició de dates o un pas de processament incorrectes en un programa
- Fallo (***failure***): quan el sistema, o algun dels seus components, es incapaç de realitzar les funcions requerides dins dels rendiments especificats
- Error (***error***): acció humana que condueix a un resultat incorrecte (error de programació)

Relació entre error, defecte i fallo



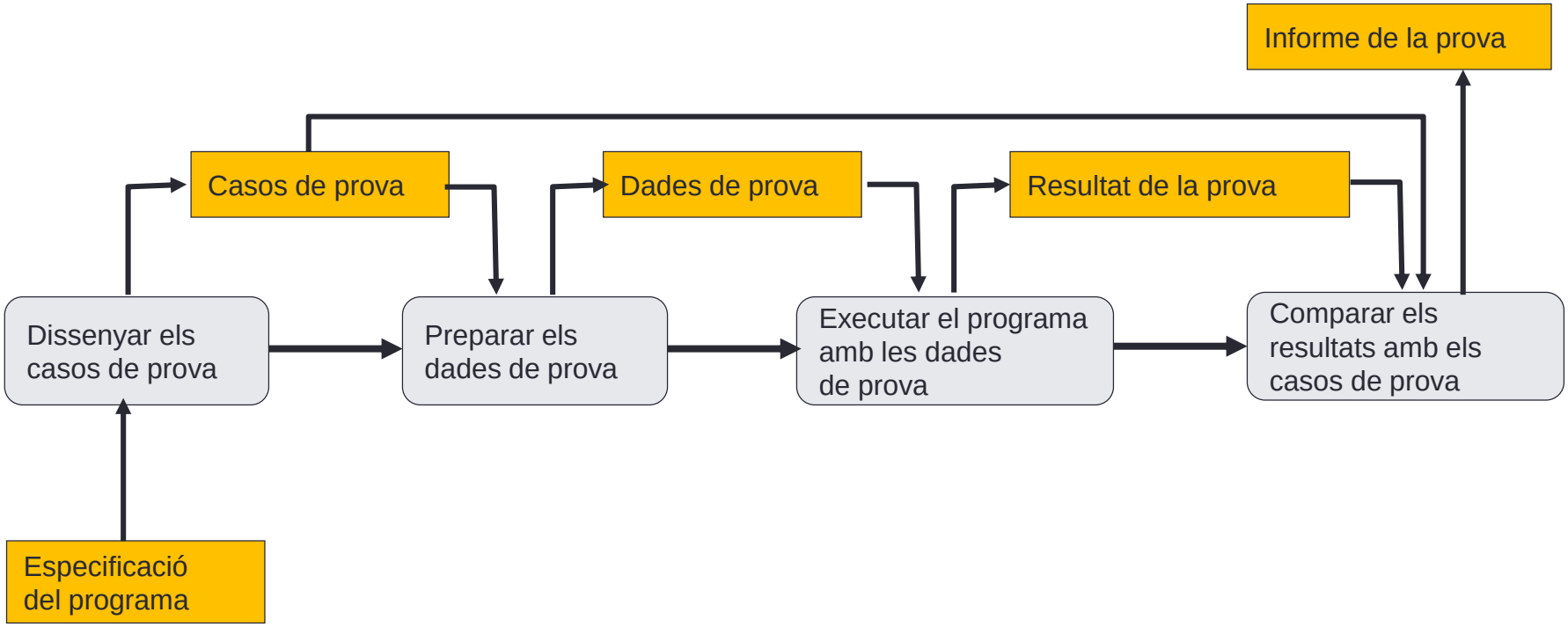
El procés de prova



El procés de prova

- Proves d'unitat: es prova cada mòdul individualment.
- Prova funcional o d'integració: el programari totalment assemblat es prova com un conjunt, per a comprovar si compleix o no tant els requisits funcionals com els requisits de rendiment, seguretat, etc.
- Prova del sistema: el programari ja validat s'integra amb la resta del sistema (per exemple, elements mecànics, interfícies electròniques, etc.) per a provar el seu funcionament conjunt .
- Prova d'acceptació: el producte final es comprova per l'usuari final en el seu propi entorn d'explotació per a determinar si ho accepta com està o no (validació).

Flux d'Informació de la Prova



Procés de depuració

- Quant a la localització de l'error en el codi font:
  - Analitzar la informació i pensar.
  - En arribar a un punt mort, passar a una altra cosa.
  - En arribar a un punt mort, descriure el problema a una altra persona.
  - No solament usar eines de depuració com únic recurs.
  - No experimentar canviant el programa.
  - S'han d'atacar els errors individualment.
  - S'ha de fixar l'atenció també en les dades.

Procés de depuració

- Quant a la correcció de l'error:
  - On hi ha un defecte, sol haver-hi més.
  - Ha de fixar-se el defecte, no els seus símptomes.
  - La probabilitat de corregir perfectament un defecte no és del 100%.
  - Cura amb crear nous defectes.
  - La correcció ha de situar-nos temporalment en la fase de disseny.

TÈCNIQUES DE DISSENY DE CASOS DE PROVA

- ✓ **Caixa Blanca**
- ✓ **Caixa Negra**
- ✓ **Aleatòries**

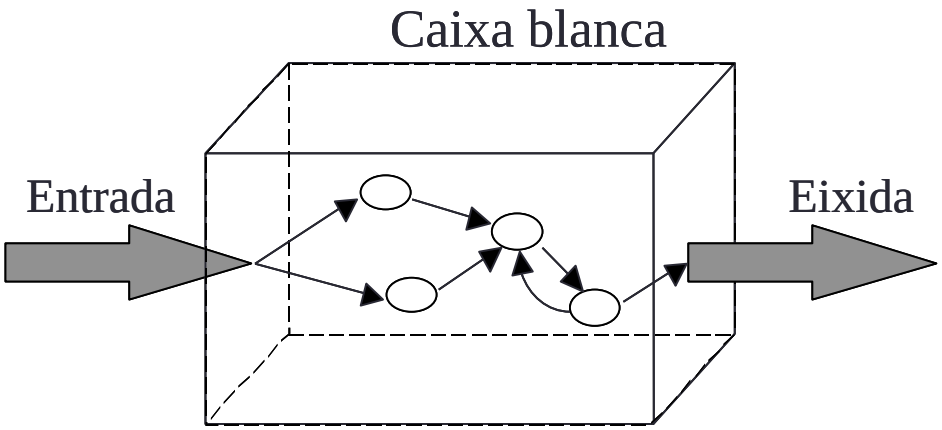
Disseny de Casos de Prova

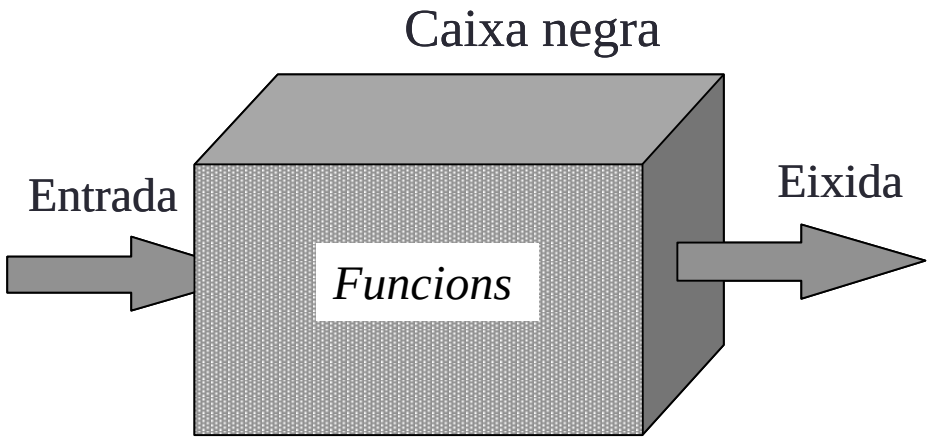
- El disseny de casos de prova per a la verificació del programari pot significar un esforç considerable (prop del 40% del temps total de desenvolupament)

- Existeixen fonamentalment tres enfocaments:
  - **Prova de caixa blanca**
  - **Prova de caixa negra**
  - **Proves aleatòries**

- Combinar tots dos enfocaments permet aconseguir major fiabilitat.

Disseny de Casos de Prova





Disseny de Casos de Prova

- La **prova de caixa blanca** o prova estructural es basa en l'estudi minuciós de tota l'operativitat d'una part del sistema, considerant els detalls procedimentals.
  - Es plantegen diferents camins d'execució alternatius i es duen a terme per a observar els resultats i contrastar-los amb l'esperat.
  - En la pràctica NO es viable la verificació mitjançant la prova de la caixa blanca de la totalitat de camins d'un procediment (el nombre de combinacions creix exponencialment.
    - S'estableixen diferents nivells de cobertura del casos de prova

Disseny de casos de Prova

- La **prova de caixa negra** o prova funcional principalment analitza la compatibilitat quant a les interfícies de cadascun dels components programari entre si.

- La **prova aleatòria** consisteix a utilitzar models que representen les possibles entrades del programa per a crear a partir d'ells els casos de prova.
  - Models estadístics que simulen les seqüències i freqüències habituals de les dades d'entrada al programa.
  - Es fonamenta que la probabilitat de descobrir un error és pràcticament la mateixa si es fan una sèrie de proves aleatòriament triades, que si es fan seguint les instruccions dictades per criteris de cobertura.
  - No obstant açò, poden romandre ocults errors que solament es descobreixen davant entrades molt concretes.
  - Aquest tipus de proves pot ser suficient en programes poc crítics.

PROVES DE CAIXA BLANCA

- ✓ **Tipus de Cobertura**
- ✓ **Tècnica del Camí Bàsic**
  - Camí Independent
  - Graf de Flux
  - Complexitat Ciclomàtica
  - Derivació de Casos de Prova

Proves de Caixa blanca

- La prova de la caixa blanca usa l'estructura de control del disseny procedural per a derivar els casos de prova.

- Idea: no és possible provar tots els camins d'execució diferents, però sí confeccionar casos de prova que garantisquen que es verifiquen tots els camins d'execució anomenats independents.

- Verificacions per a cada camí independent:
  - Provar es seues dues facetes des del punt de vista lògic, és a dir, certa i falsa.
  - Executar tots els bucles en els seus límits operacionals
  - Exercitar les estructures internes de dades.

Proves de Caixa Blanca

- Els errors lògics i les suposicions incorrectes són inversament proporcionals a la probabilitat que s'execute un camí del programa.
- Sovint creiem que un camí lògic té poques possibilitats d'executar-se quan, de fet, es pot executar de forma regular.
- Els errors tipogràfics són aleatoris.
- Tal com va apuntar Beizer,

  "els errors s'amaguen en els racons i s'aglomeren en els límits"

Tipus de cobertura

- **Cobertura de sentències**. Cada sentència o instrucció del programa s'execute almenys una vegada.
- **Cobertura de decisions**. Cada decisió tinga, almenys una vegada, un resultat cert i, almenys una vegada, un de fals.
- **Cobertura de condicions**. Cada condició de cada decisió adopte el valor cert almenys una vegada i el fals almenys una vegada

---

- **Criteri de decisió/condició**. Consisteix en exigir el criteri de cobertura de condicions obligant al fet que es complisca també el criteri de decisions
- **Criteri de condició múltiple**. Ha de garantir totes les combinacions possibles de les condicions dins de cada decisió.

Tipus de cobertura

Exemple 1:

```
si (a>b)
  entonces a=a-b
  si no b=b-a
fin_si
```

Example 2:

```
si (a>b)
  entonces a=a-b
fin_si
```

Example 3:

```
i=1
mientras (v[i]<>b)
y(i<>5)
hacer
   i=i+1;
fin_mientras
```

Example 4:

```
si (a>b) y (b es
primo)
  entonces a=a-b
fin_si
```

Tipus de cobertura

```
1  if (a>b) and (b is prime)
2    then a=a-b
   //without else
   end_if
```

Cobertura de sentencia: 2 fragmentos

```
   if (a>b) and (b is prime)      1
     then a=a-b                   2
   //without else                 3
   end_if
```

Cobertura de Decisión: 3 fragmentos

```
        1, 2          3, 4
   if (a>b) and (b is prime)
5    then a=a-b
6  //without else
   end_if
```

Cobertura Condición: 6 fragmentos

2^2 = {TT, TF, FT, FF} True table

```
        1, 2          3, 4
   if (a>b) and (b is prime)
5    then a=a-b
6  //without else
   end_if
```

Multiple condición: 6 fragmentos

Prova del Camí Bàsic

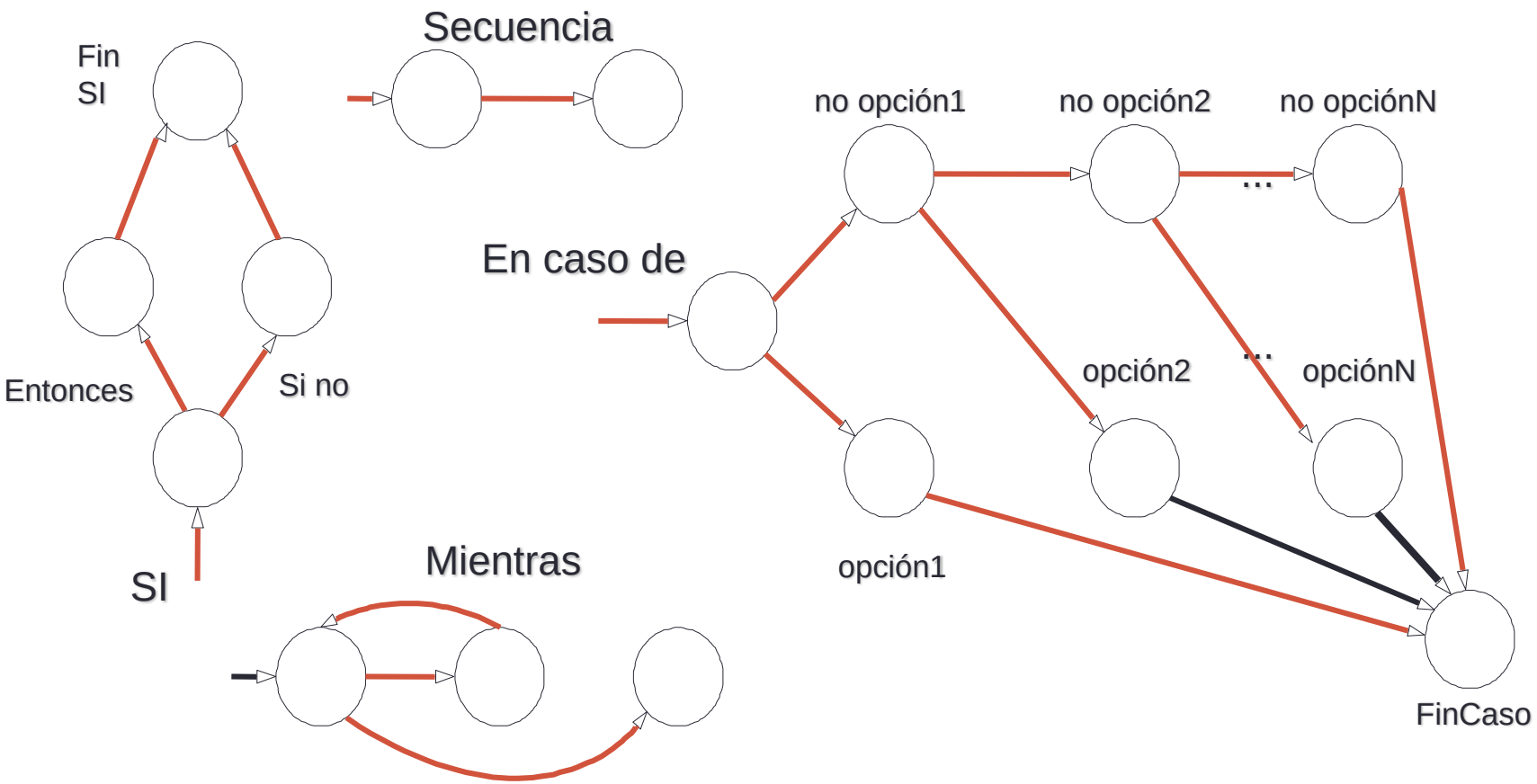
- La prova del camí bàsic és una tècnica de prova de la caixa blanca proposta per Tom McCabe.
- La idea és derivar casos de prova a partir d'un conjunt de camins independents pels quals pot circular el flux de control.
- Camí independent és aquell que introdueix almenys una sentència de processament (o condició) que no estava considerada en el conjunt de camins independents calculats fins a aqueix moment.
- Per a obtenir el conjunt un conjunt de camins independents es construirà el Graf de Flux associat i es calcularà la seua Complexitat Ciclomàtica.

Prova del camí bàsic: Graf de Flux

- El flux de control d'un programa pot representar-se per un **graf de flux**.
- Cada **node** del graf correspon a una o més sentències de codi font.
- Cada node que representa una condició es denomina **node predicat**.
- Qualsevol representació del disseny procedural es pot traduir a una **aresta** del graf de flux.

➢ Un **camí independent** en el graf és el que inclou alguna aresta nova, és a dir, que no estava present en els camins definits prèviament.

Prova del Camí Bàsic: Graf de Flux



Prova del camí bàsic: Graf de Flux

- Si es dissenyen casos de prova que cobrisquen els camins bàsics, es garanteix l'execució de cada sentència almenys una vegada i la prova de cada condició en les seues dues possibilitats lògiques (certa i falsa).

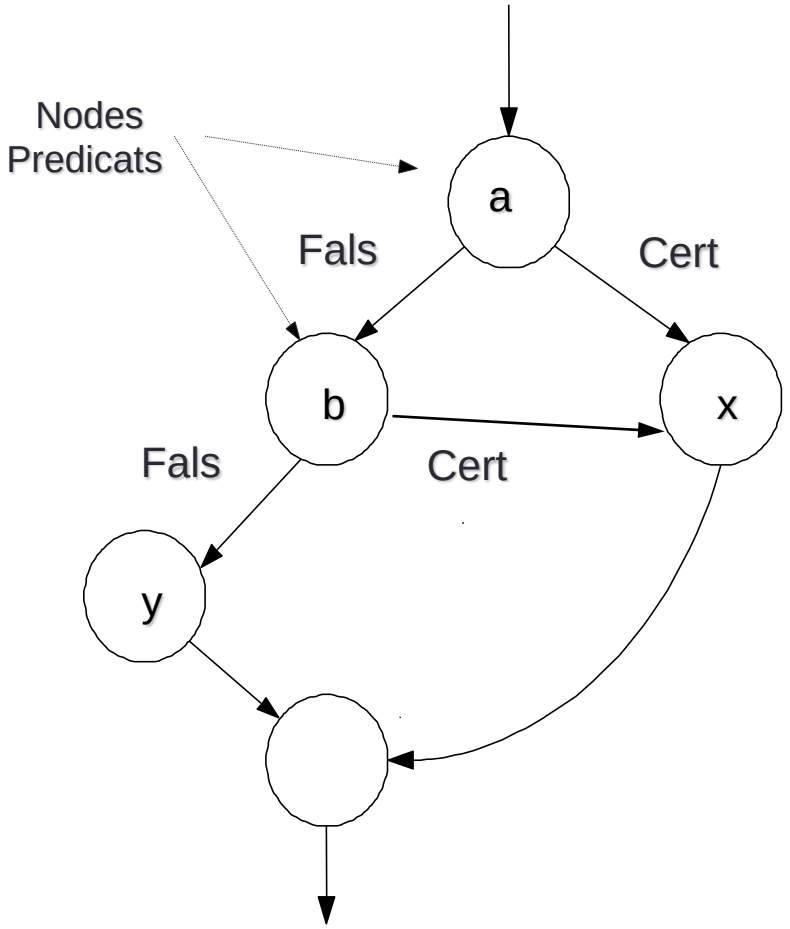
- El conjunt bàsic per a un graf donat pugues no ser únic, depèn de l'ordre en què es van definint els camins.

- Quan apareixen condicions lògiques compostes la confecció del graf és més complexa.

Prova del Camí Bàsic: Graf de Flux

- Example:

SI a O b
Entonces
fer x
Si No
fer y
FinSI



Prova del Camí Bàsic: Complexitat Ciclomàtica

- Complexitat ciclomàtica d'un graf de flux, V(G), indica el nombre màxim de camins independents en el graf.

- Pot calcular-se de tres formes alternatives:

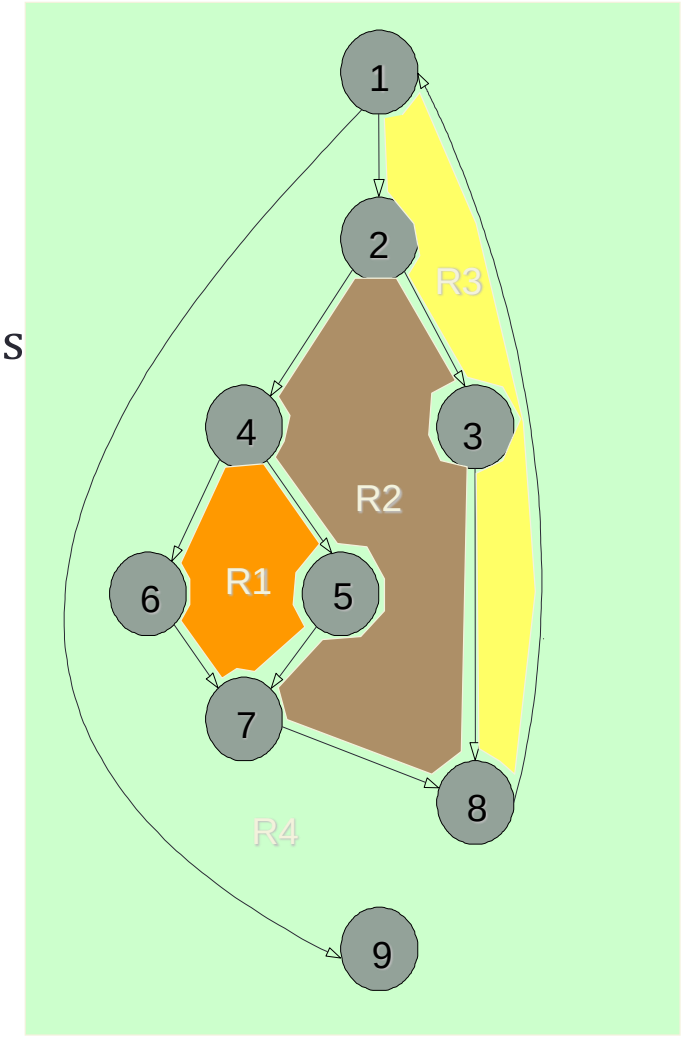
  - El **nombre de regions** en que el graf de flux divideix el plànol.

  - **V(G) = A - N + 2**, on A es el nombre d'arestes i N es el nombre de nodes.

  - **V(G) = P + 1**, on P es el nombre de nodes predicat.

Prova del Camí Bàsic: Complexitat Ciclomàtica

- V(G) = 4
  - El grax de la figura delimita cuatre regions
  - 11 arestes - 9 nodes + 2 = 4
  - 3 nodes predicat + 1 = 4



Prova del Camí Bàsic: Complexitat Ciclomàtica

- El conjunt de camins independents del graf serà 4.
  - Camí 1: 1-9
  - Camí 2: 1-2-4-5-7-8-1-9
  - Camí 3: 1-2-4-6-7-8-1-9
  - Camí 4: 1-2-3-8-1-9

- Qualsevol altre camí no serà un camí independent, p.e., 1-2-4-5-7-8-1-2-3-8-1-2-4-6-7-8-1-9

  ja que és simplement una combinació de camins ja especificats

- Els quatre camins anteriors constitueixen un *conjunt bàsic* per al graf

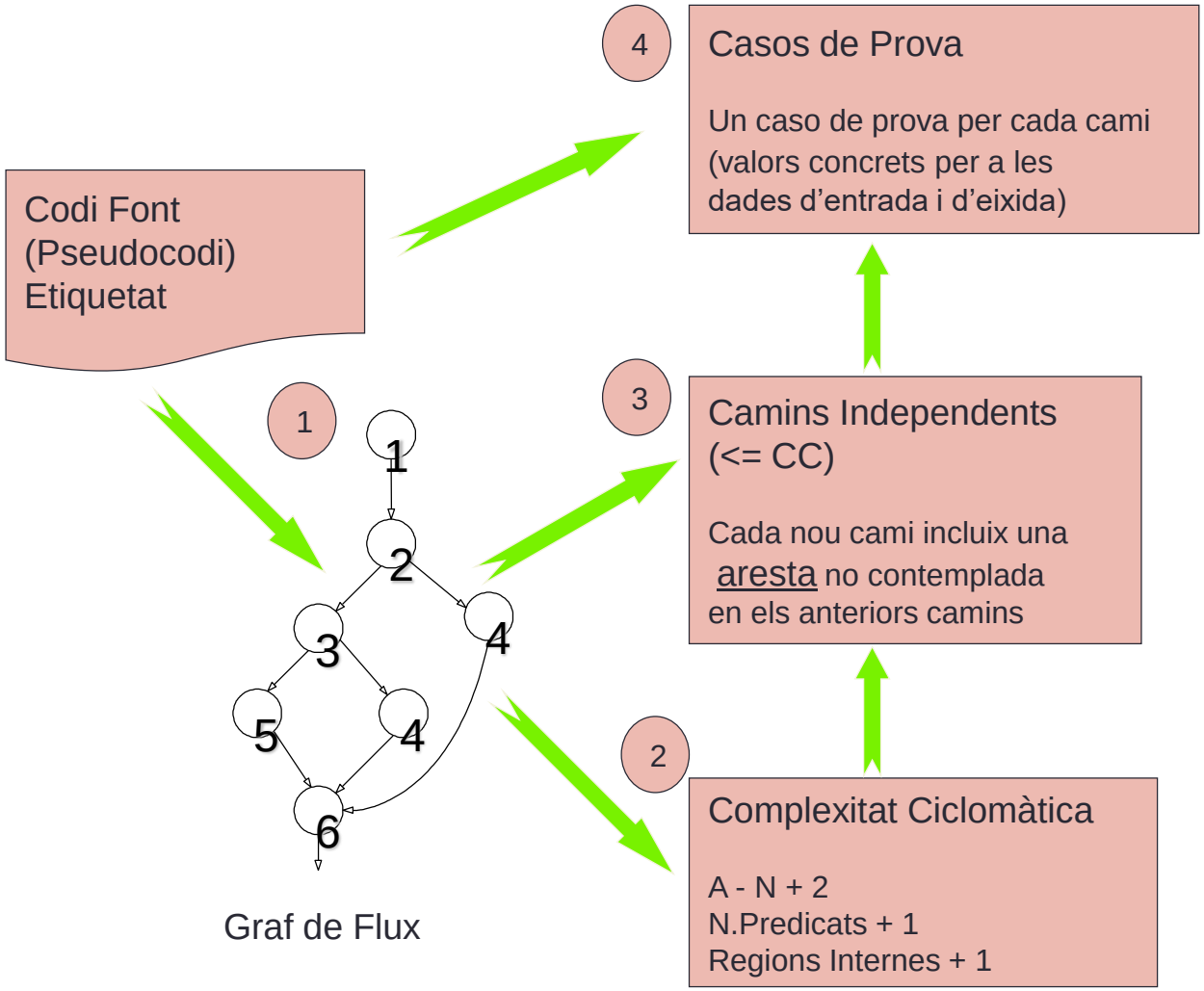
Prova del Camí Bàsic: Derivació de casos de prova

- El mètode de prova del camí bàsic es pot aplicar a un disseny procedural detallat (pseudocodi) o al codi font de l'aplicació.

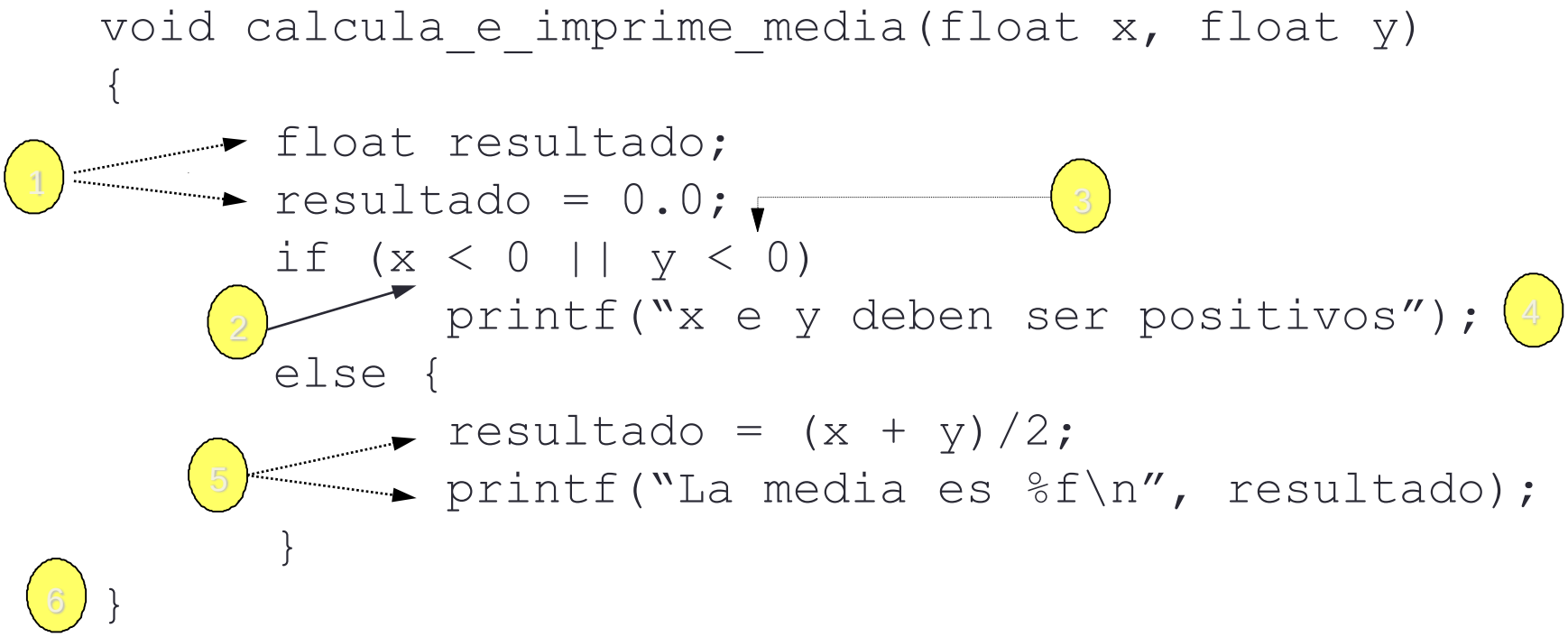
- Passos per a dissenyar els casos de prova:

0. Se etiqueta el codi font, o el pseudocodi, enumerant cada instrucció i cada condició simple.
1. A partir del codi font etiquetat, es dibuixa el graf de flux associat.
2. Es calcula la complexitat ciclomàtica del graf.
3. Es determina un conjunt bàsic de camins independents.
4. Es preparen els casos de prova que obliguen a la execució de cada camí del conjunt bàsic.

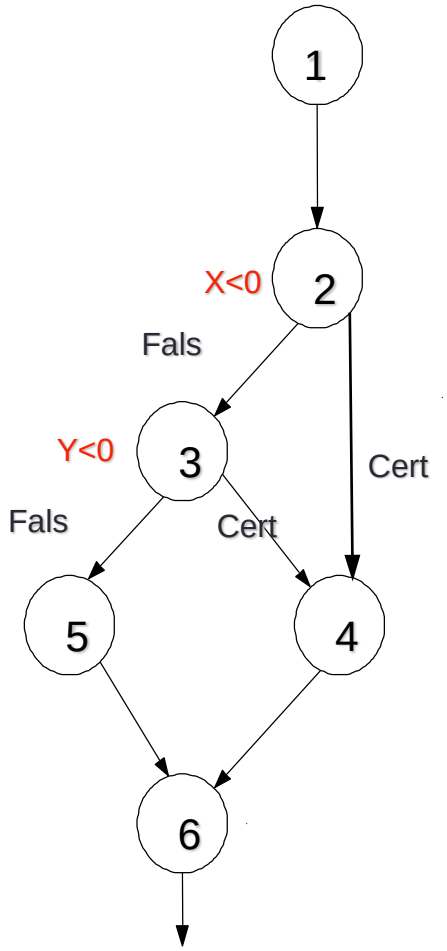
Prova del Camí Bàsic: Derivació dels Casos de Prova



Prova del Camí Bàsic: Exemple



Prova del Camí Bàsic: Exemple



V(G) = 3 regions. Per lo tant, hi ha que determinar tres camins independents.

- Camí 1: 1-2-3-5-6
- Camí 2: 1-2-4-6
- Camí 3: 1-2-3-4-6

Casos de prova per a cada camí:

Camí 1: x=3, y= 5, resultat=4

Camí 2: x=-1, y=3, resultat=0, error

Camí 3: x=4, y=-3, resultat=0, error

Exercici Camí Bàsic: Codi Font

```
int contar_letra(char cadena[10], char letra)
{
  int contador, n,  lon;
  n=0; contador=0;
  lon = strlen (cadena);
  if (lon > 0) {
    do {
      if (cadena[contador] == letra) n++;
      contador++;
      lon--;
    } while (lon > 0);
  }
  return n;
}
```

Exercici Camí Bàsic: Codi Font Etiquetat

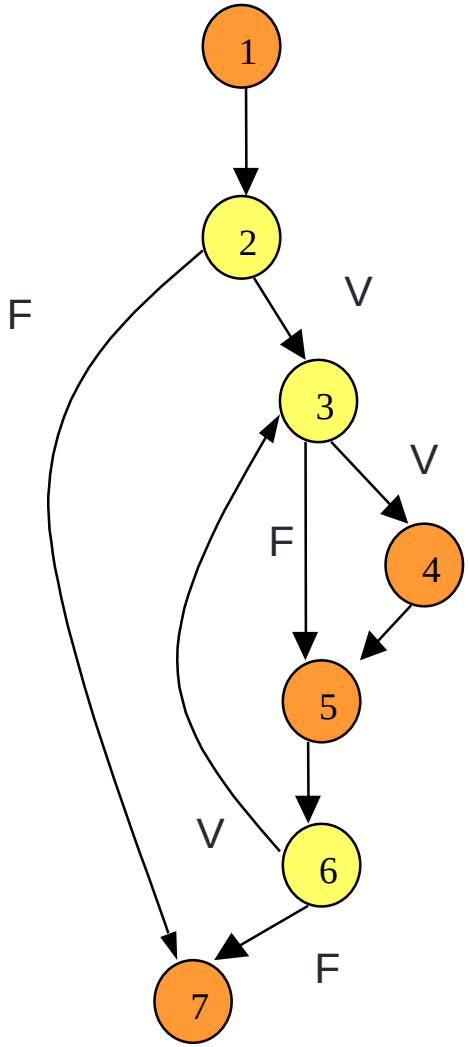
```
int contar_letra(char cadena[10], char letra)
{
  int contador, n,  lon;
  n=0; contador=0;
  lon = strlen (cadena);
  if (lon > 0) {
    do {
      if (cadena[contador] == letra)
            n++;
      contador++;
      lon--;
    } while (lon > 0);
  }
  return n;
}
```

1 2 3 4 5 6 7

Exercici Camí Bàsic: Graf de Flux

```
int contar_letra(char cadena[10], char letra)
{
  int contador, n,  lon;
  n=0; contador=0;
  lon = strlen (cadena);
  if (lon > 0) {
    do {
      if (cadena[contador] == letra)
            n++;
      contador++;
      lon--;
    } while (lon > 0);
  }
  return n;
}
```

1 2 3 4 5 6 7



Exercici Camí Bàsic: Camins Independents

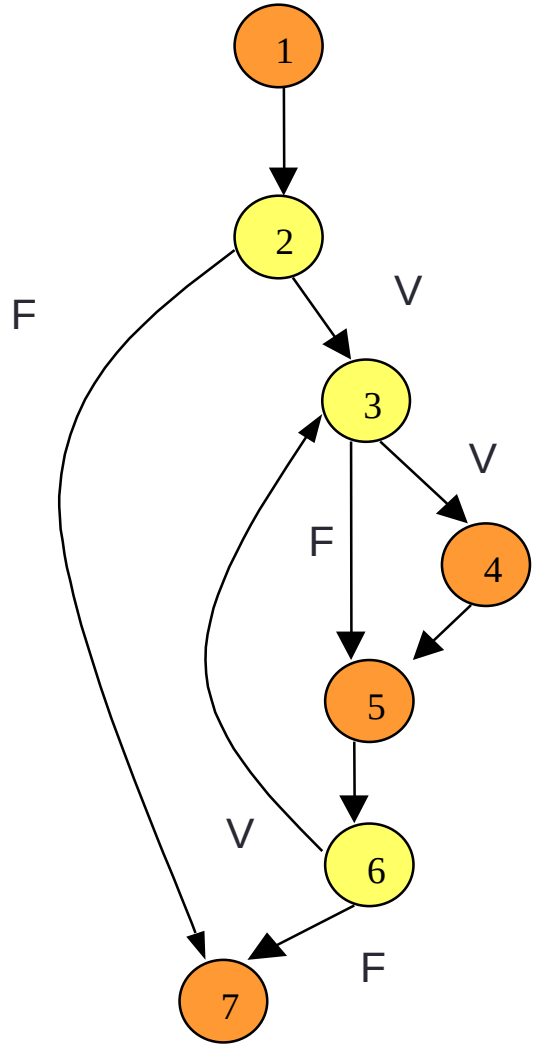
- V(G) = 4;

  Nodes=7; Arestes=9;

  Nodes Predicat=3;

  Regions = 4

- Conjunt de camins independents:
  1. 1-2-7
  2. 1-2-3-4-5-6-7
  3. 1-2-3-5-6-7
  4. 1-2-3-4-5-6-3-5-6-7 (No és l'únic)



Exercici Camí Bàsic: Casos de Prova

1. 1-2-7 cadena = ”” letra = ‘a’ n = 0;
2. 1-2-3-4-5-6-7 cadena = “a” letra = ‘a’ n = 1;
3. 1-2-3-5-6-7 cadena = “b” letra = ‘a’ n = 0;
4. 1-2-3-4-5-6-3-5-6-7 cadena = “ab” letra = ‘a’ n = 1;

PROVES DE CAIXA NEGRA

- ✓ **Tècnica de la Partició Equivalent**
- ✓ **Anàlisi de Valors Límit**

Proves de Caixa Negra

- Els mètodes de la caixa negra se centren sobre els requisits funcionals del programari.

- La prova de la caixa negra intenta trobar errors dels següents tipus:
  - Funcions incorrectes o inexistents.
  - Errors relatius a les interfícies.
  - Errors en estructures de dades o en accessos a bases de dades externes.
  - Errors deguts al rendiment.
  - Errors d'inicialització o terminació.

Partició Equivalent

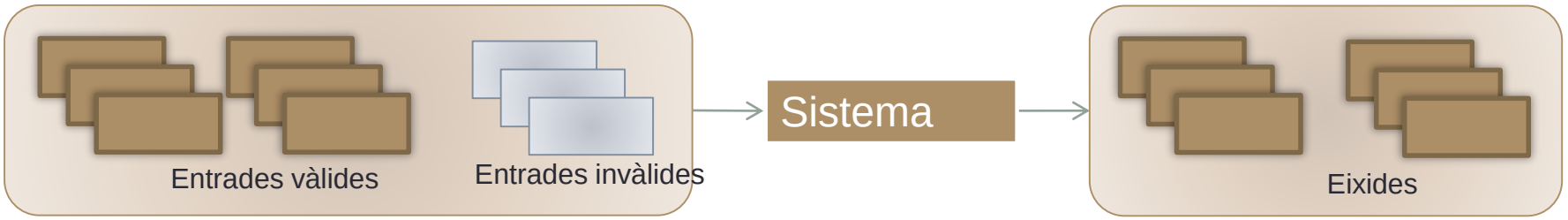
- Consisteix en derivar casos de prova mitjançant **la partició del domini d'entrada** en classes d'equivalència.
  - S'avalua el comportament per a un valor representatiu de cada classe.

- Una **classe d'equivalència** representa el conjunt d'estats vàlids o invàlids per a totes les condicions d'entrada que satisfacin aquesta classe.
  - La prova d'un valor representatiu d'una classe permet suposar «raonablement» que el resultat d'èxit o fallo serà el mateix que l'obtingut provant qualsevol altre valor de la classe

Partició Equivalent

- **Condicions de les entrades** del programari sota proves: restriccions *de format* o *contingut* de les dades d'entrada
  - De dades vàlides
  - De dades no vàlides o erronis

- **Objectiu**: *Descobrir classes d'errors per a cada classe d'equivalència identificada.*
  - Al reduir el nombre de casos de prova a un conjunt mes xicotet podem perdre informació relevant.



Partició Equivalent

- **Tècnica** per a **identificació** de casos de prova:
  1. Per a cada condició d'entrada del programari sota proves, identificar les seues classes d'equivalència utilitzant les **heurístiques**.
  2. Assignar un nombre únic a cada classe d'equivalència identificada.
  3. Fins que totes les classes d'equivalència vàlides hagen sigut cobertes per incorporades a casos de prova, escriure un cas de prova que cobrisca tantes classes vàlides no incorporades com siga possible.
  4. Fins que totes les classes d'equivalència no vàlides hagen sigut cobertes per un cas de prova, escriure un cas de prova per a una cada classe no vàlida sense cobrir.

  El procés anterior també pot aplicar-se per a les condicions d'eixida del programari sota proves

Partició Equivalent

- **Heurístiques d'identificació** de classes d'equivalència:
  a) Si s'especifica un **rang de valors** per a les dades d'entrada, es crearà una classe vàlida i dues classes no vàlides
  b) Si s'especifica un **nombre finit de valors**, es crearà una classe vàlida i dues no vàlides
  c) Si s'especifica una situació del tipus **«ha de ser»** o **booleana** per exemple, «el primer caràcter ha de ser una lletra», s'identifiquen una classe vàlida «és una lletra» i una no vàlida «no és una lletra»
  d) Si s'especifica **un conjunt de valors** admesos i se sap que el programa tracta de forma diferent cadascun d'ells, s'identifica una classe vàlida per cada valor i una no vàlida
  e) Si se sospita que certs elements d'una classe no es tracten igual que la resta de la mateixa, han de dividir-se en **Classes Menors**

Partició Equivalent

Heurístiques	Nombre de classes vàlides	Nombre de classes invàlides
Rango de valors **Ex: [20..30]**	**1:** valor dins del rango (**25**)	**2:** uno bajo el límite inferior y otro sobre el límite superior (**15,41**)
Valor finito **Ex: {2,4,6,8,10}**	**1**: valor dins del conjunt (**4**)	**2**: uno inferior del menor valor i altre superior al valor mes alt (**1,12**)
«deu ser » condició **booleana** **Ex: deu ser una lletra**	**1**: valor que cumplix la condició (**'m'**)	**1**: valor que fa la condició falsa
Conjunt de valors aceptats **Ex: { blue, white, black}**	**Nombre de** valors aceptats **(3: { blue, white, black})**	**1**: Un valor no aceptat (**red**)
Classes menors **0<value<5;** **10<=value<20**	Quan ho podem transformar en subclasses dels exemples previs (**una per a]0,5[i altra per [10,20[), cada subclasse provoca un comportament o eixida diferent**	

Example - Partició Equivalent

Un programa pren com a entrada un fitxer el format del qual de registre consta dels següents camps:

- Id-empleat és un camp de nombres enters positius de 3 dígits exclòs el 000.
- Nom-empleat és un camp alfanumèric de 10 caràcters.
- Mesos-Treballa és un camp que indica el nombre de mesos que porta treballant l'empleat; és un enter positiu inclou el 000 de 3 dígits.
- Directiu és un camp d'un sol caràcter que pot ser «+» per a indicar que l'empleat és un directiu i «-» per a indicar que no ho és.

El programa assigna un valor (P1, P2, P3, P4) que s'imprimeix en un llistat a cada empleat segons les normes següents:

- P1 als directius amb, almenys, 12 mesos d'antiguitat
- P2 als no directius amb, almenys, 12 mesos d'antiguitat
- P3 als directius sense un mínim de 12 mesos d'antiguitat
- P4 als no directius sense un mínim de 12 mesos d'antiguitat

Es demana:

a) Crear una taula de classes d'equivalència les classes hauran de ser numerades en la qual s'indiquen les següents columnes en cada fila: 1 Condició d'entrada que s'analitza; 2 Classes vàlides i 3 Classes no vàlides que es generen per a la condició 4 Regla heurística que s'aplica per a la generació de les classes de la fila

b) Generar els casos de prova

Example - Partició Equivalent

Condició d'entrada	Classes vàlides	Classes no vàlidas	Regla heurística
Id-empleat	[001-999] 1	<=000 2 >999 3 No és un nombre 4	Rang Booleana
Nom-empleat	10 caràcters 5	<10 6 >10 7	Nº finit de valors
Mesos-Treballa	[000-011] 8 [012-999] 9	<000 10 >999 11 No nombre 12	Rang Classes menors Booleana
Directiu	+ 13 - 14	Altre caràcter 15	Conjunt de valors

CP C.Valides	Classes vàlides	Entrada	Eixida
	1 – 5 – 8 - 13	123, gumersindo, 9, +	P4
	1 – 5 – 9 – 14	456, sebastiano, 13, -	P2

Casos de Prova No Vàlids	c. no vàlides	Entrada	Eixida
	2 - 5 - 9 - 13	000, gumersindo, 14, +	
	3 - 5 - 9 - 14	1024, minotauros, 16, -	
	4 - 5 - 8 - 13	abc, sebastiano, 8, +	
	1 - **6** - 8 - 13	123, cobos, 6, +	
	1 - **7** - 8 - 13	123, torreceballos, 3, +	
	1 - 5 - **10** - 13	123, margaritos, -1, +	
	1 - 5 - **11** - 14	123, margaritos, 1024, -	
	1 - 5 - **12** - 14	123, margaritos, abc, -	
	1- 5 – 9 - **15**	123, margaritos, 13, *	

Exercici Partició Equivalent

- Aplicació bancària. Dades d'entrada:
  - **Codi d'àrea**: nombre de 3 dígits que no comença per 0 ni per 1
  - **Nom d'identificació d'operació**: 6 caràcters
  - **Ordres possibles**: “xec”, “dipòsit”, “pagament factura”, “retirada de fons”

Exercici Partició Equivalent

Codi d'àrea:

- Lògica:
  - 1 classe vàlida: nombre
    - Rang:
      - 1 classe vàlida: 200 < codi< 999
      - 2 classes no vàlides: codi < 200; codi> 999
  - 1 classe no vàlida: no es número

Nom d'identificació:

- Valor específico:
  - 1 classe vàlida: 6 caràcters
  - 2 classes no vàlides: més de 6 caràcters; menys de 6 caràcters

Ordres possibles:

- Conjunt de valors:
  - 1 classe vàlida: 4 ordres vàlides
  - 1 classe no vàlida: ordre no vàlida

Exercici Partició Equivalent

Dades d'Entrada	Classes Vàlides	Classes No Vàlides
Codi d'àrea	(1) 200 <= codi <= 999	(2) codi < 200 (3) codi > 999 (4) no es numèric
Identificació	(5) 6 caràcters	(6) menys de 6 caràcters (7) més de 6 caràcters
Ordre	(8) "xec" (9) "dipòsit" (10) "pagament factura" (11) "retirada de fons"	(12) ninguna ordre vàlida

Exercici Partició Equivalent

Codi	Identificació	Ordre	Classes Cubertes
300	Nòmina	“Dipòsit”	(1)^{C} (5)^{C} (9)^{C}
400	Viatges	“Xec”	(1) (5) (8)^{C}
500	Cotxes	“Pagament factura”	(1) (5) (10)^{C}
600	Menjar	“Retirada fonds”	(1) (5) (11)^{C}

Exercici Partició Equivalent: Casos de Prova NO Vàlids

Codi	Identificació	Ordre	Classes Cubertes
180	Viatges	“Pagament factura”	(2) C (5) (10)
1032	Nòmina	“Dipòsit”	(3) C (5) (9)
XY	Compra	“Retirada fonds”	(4) C (5) (11)
350	A	“Dipòsit”	(1) (6) C (9)
450	Regals	“Xec”	(1) (7) C (8)
550	Caseta	&%4	(1) (5) (12) C

Exemple Crear Reserva de Vehicle

- El usuari introduirà les següents dades per a realitzar la reserva d'un vehicle:

  - DNI (DNI: 9 caràcters, 8 dígits i 1 lletra),
  - Data de recollida del vehicle (Dia, Mes i Any),
  - Categoria del vehicle: valor sencer entre 1 i 10
  - Modalitat de lloguer: "per km" o "ilimitat"

Exemple Crear Reserva

Condició d'ntrada	Classes vàlides	Classes no vàlides	Regla heurística
DNI	9 caracters (1) DDDDDDDDL	<9 (2) >9 (3) No son dígitos (4) No es letra (5)	Nº finito de valores Booleana Booleana
Fecha Recogida	dd-mm-aaaa existe (6)	Fecha no existe (7) No son dígitos (8) Día no válido > 31 (9) Mes no válido > 12 (10) Año no válido < 2018 (fecha pasada) (11)	Booleana Booleana Rango Rango Rango
Categoría	1 < num cat <= 10 (12)	Num cat < 1 (13) Num cat > 10 (14)	Rango
Modalidad de Alquiler	Km (15) Ilimitado (16)	No seleccionado (17)	Conjunto de valores

Exemple Crear Reserva

CP C.Vàlides	Classes vàlides	Entrada	Eixida
	1 – 6 – 12 - 15	“12345678A” “15-01-2019” 3 “km”	Lloguer OK
	1 – 6 – 12 – **16**	“12345678A” “15-01-2019” 3 “Ilimitat”	Lloguer OK

Casos de Prova No Vàlids	c. no vàlides	Entrada	Eixida
	2 – 6 – 12 - 15	“1234567A” “15-12-2019” 3 “km”	Error DNI < 9 car.
	3 - 6 – 12 - 15	“12345678A” “15-12-2019” 3 “km”	Error DNI > 9 car.
	4 - 6 – 12 - 15	“Z2345678A” “15-12-2019” 3 “km”	Error DNI No dígits
	5 - 6 – 12 - 15	“123456789” “15-12-2019” 3 “km”	Error DNI No lletra
	1 - **7** – 12 - 15	“12345678A” “30-02-2019” 3 “km”	Error Data no existeix
	1 - **8** – 12 - 15	“12345678A” “3s-02-2019” 3 “km”	Error Data no Dígits
	1 - **9** – 12 - 15	“12345678A” “32-02-2019” 3 “km”	Error Dia incorrecte
	1 - **10** – 12 - 15	“12345678A” “15-13-2019” 3 “km”	Error Mes incorrecte
	1 - **11** – 12 - 15	“12345678A” “15-12-2010” 3 “km”	Error Any passat
	1 – 6 – **13** – 15	“12345678A” “15-12-2019” 0 “km”	Error Categoria <1
	1 – 6 – **14** – 15	“12345678A” “15-12-2019” 11 “km”	Error Categoria > 10
	1 – 6 – 12 – **17**	“12345678A” “15-12-2019” 3 “”	Error Modalitat buida

Anàlisi de valors límit

- La tècnica d'Anàlisi de Valors Límits selecciona casos de prova que exerciten els valores límit donada la tendència de l'aglomeració d'errors en els extrems.

- Complementa la de la partició equivalent. En lloc de realitzar la prova amb qualsevol element de la partició equivalent, s'escullen els valors en les vores de la classe.

- Es deriven tant casos de prova a partir de les condicions d'entrada com amb les d'eixida.

Anàlisi de valors límit

- Directrius:
  - Si una condició d'entrada especifica un **rang delimitat pels valors a i b**, s'han de dissenyar casos de prova per als valors a i b i per als valors just per sota i just per sobre de tots dos.
  - Si la condició d'entrada especifica un **nombre de valors**, s'han de desenvolupar casos de prova que exerciten els valors màxim i mínim a més dels valors just damunt i sota aquells.
  - Aplicar les directrius anteriors a les condicions d'eixida. Compondre casos de prova que produïsquen eixides en els seus valors màxim i mínim.
  - Si les estructures de dades definides internament tenen **límits prefixats** (p.i., un array de 10 entrades), s'ha d'assegurar dissenyar un cas de prova que exercite l'estructura de dades en els seus límits.

Anàlisi de valors límit

Heurístiques	Nombre de classes vàlides	Nombre de classes invàlides
Rango de valors **Ex: [20..30]**	**4:** valors en els límits (**20,21,29,30**)	**2:** un **just per davall** del límit inferior i altre **just per damunt** del límit superior (**19,31**)
Conjunt finito de valors **Ex: {2,4,6,8,10}**	**4**: valores mínims i màxims del conjunt (**2,4, 8,10**)	**2**: valores fora del conjunt, un **just per davall** del valor menor i altre **just por damunt** del valor superior (**1,11**)

ALTRES TIPUS DE PROVES – EINES AUTOMÀTIQUES DE PROVES

Altres tipus de proves

- Recorreguts (“walkthroughs”).
- Proves de robustesa (“robustness testing”)
- Proves d’aguant (“stress”)
- Prestacions (“performance testing”)
- Conformitat o Homologació (“conformance testing”)
- Interoperabilitat (“interoperability tesing”)
- Regressió (“regression testing”)
- Prova de comparació

Eines automàtiques de prova

- Analitzadors estàtics. Aquests sistemes d'anàlisis de programes suporten proves de les sentències que consideren més febles dins d'un programa.
- Auditors de codi. Són filtres amb el propòsit de verificar que el codi font compleix determinats criteris de qualitat (dependrà fortament del llenguatge en qüestió).
- Generadors d'arxius de prova. Aquests programes confeccionen automàticament fitxers amb dades que serviran d'entrada a les aplicacions.
- Generadors de dades de prova. Confeccionen dades d'entrada determinats que comporten un comportament concret del programari.
- Controladors de prova. Confeccionen i alimenten les dades d'entrada i simulen el comportament d'altres mòduls per a restringir l'abast de la prova.

Eines automàtiques de prova

- Analitzadors estàtics:
  - Defectes en les dades: variables usades abans d'inicialitzar-se, variables declarades i mai usades, variables que no s'usen entre dues assignacions d'algun valor, possibles violacions dels límits de vector, etc.
  - Defectes en el control: fragments de codi que mai s'aconsegueix.
  - Defectes en l'entrada/eixida: es retorna una variable d'eixida sense que es modifique el seu valor.
  - Defectes en la interfície: tipus o nombre de paràmetres incorrectes, no utilització del resultat d'una funció, funcions no anomenades.
  - Defectes en el maneig de punters.