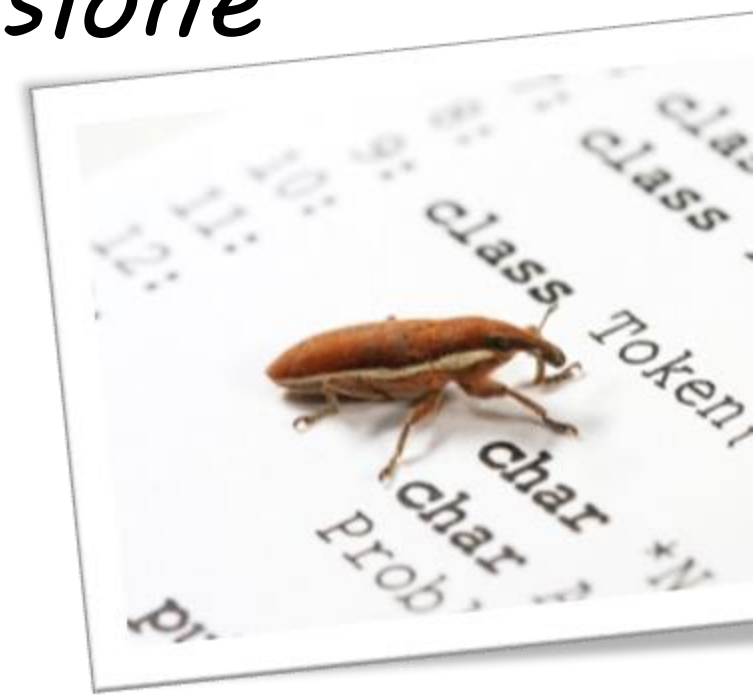
Testing di Regressione



Content

- Ottimizzazione di una test suite per il testing di regressione

- Problemi
  - Test Suite Minimization
  - Test Case Selection
  - Test Case Prioritization

Testing di Regressione

Il testing di regressione consiste nel ritestare le parti non modificate di un programma/software. I test case sono rieseguiti con lo scopo di verificare se le parti modificate (aggiunte) hanno introdotto o meno nuovi bug.

Version 1	**Version 2**
1. Develop **P**	**4.** Modify P to **P'**
2. Test **P**	**5.** Test **P'** for new functionality
3. Realease **P**	**6.** Perform regression testing on **P'** to ensure that the code corried over from P behave correctly
	7. Release **P'**

Testing di Regressione

Al termine di un task di manutenzione, ci sono due tipi di testing da effettuare:

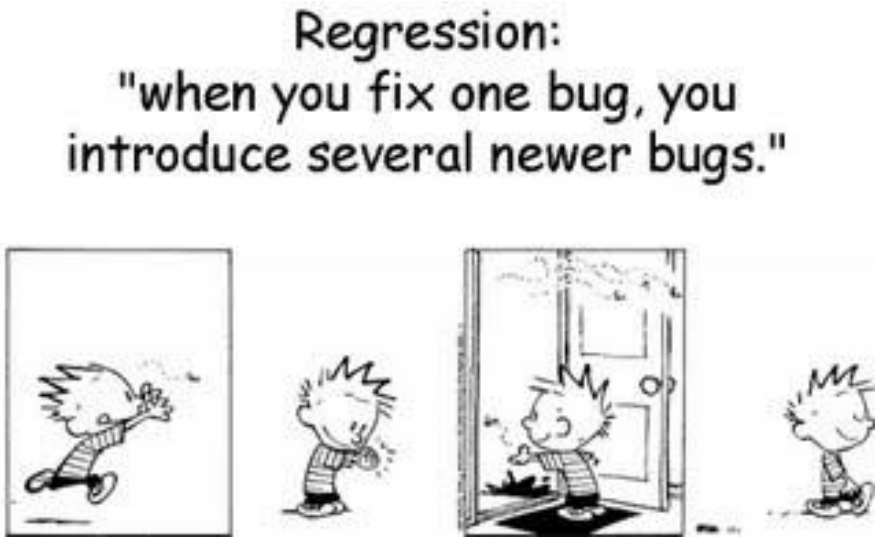
1) Testing funzionale sulle funzionalità aggiunte/modificate. Lo scopo consiste nel testare le funzionalità aggiunte e verificarne il corretto funzionamento.

2) Testing di regressione sulle funzionalità non modificate. L'obiettivo consiste nel verificare se le funzionalità esistenti funzionano correttamente e che i nuovi cambiamenti non hanno introdotto nuovi bug per le funzionalità pre-esistenti.

Perché Testing di Regressione?

Il testing di regressione "inizia" quando uno sviluppatore modifica il codice per eliminare un bug esistente; quando aggiunge nuovo codice per implementare nuove funzionalità; ecc.

Problema: possono esserci delle dipendenze tra le nuove e le vecchie funzionalità.

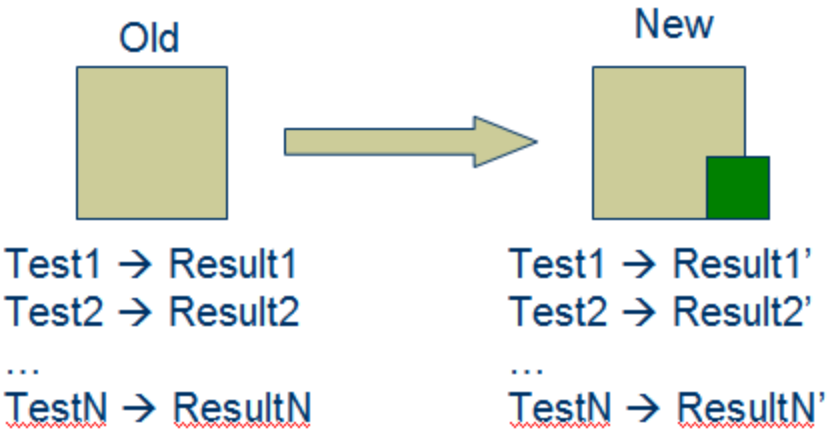


Testing di Regressione

Le **fasi** del testing di regressione sono due:

1. Rieseguire i test case relativi alle vecchie funzionalità.

2. Confrontare il comportamento del programma prima e dopo il task di manutenzione.



$Result_i = Result_i'$

forall i

Strategie

- TEST-ALL: rieseguire tutti i test case validi realizzati nelle versioni precedenti del programma **P**.

Vantaggi:
La strategia TEST-ALL è la migliore strategia da applicare per verificare se P' produce gli stessi risultati di P per tutti i test case.

Svantaggi:
Rieseguire tutti i test case può essere impraticabile:
-Limitazioni di tempo (deadline da rispettare)
-Risorse disponibili limitate.

Strategie

- Ottimizzazione della test suite: rieseguire un **sottosinsieme** ottimale dei test case validi realizzati nelle versioni precedenti del programma **P**.

Perché ottimizzare?

- Più test case possono eseguire/testare le stesse parti di codice (test case ridondanti)
- Test case ridondati possono avere diversi tempi di esecuzione.
- L'ordine con cui si eseguono i test case può influire sui tempi necessari per identificare i difetti.

Strategie

- Ottimizzazione della test suite: rieseguire un **sottosinsieme** ottimale dei test case validi realizzati nelle versioni precedenti del programma **P**.

Perché ottimizzare?

- Più test case n… …odice
- …
- …cessari per

Soluzione Ottimale:

1) Eseguire il minor numero di test case
2) Con il minor costo di esecuzione
3) Eseguendo per prima i test case in grado di rilevare bug

Ottimizzazione della test suite

Test suite minimization: trovare la test suite di dimensione minima in grado di avere lo stesso grado di "copertura" della test suite di partenza, rispetto ad un dato criterio di copertura.

Regression test selection: selezionare un sottoinsieme della test suite sulla base delle informazioni relative al programma, alla versione modificata e alla test suite.

Test case prioritization: I test case sono ordinati in modo tale da eseguire per prima quelli che hanno maggiore probabilità di testare parti di codice con maggiore probabilità ad avere difetti.

Test Suite Minimization

- **Input**:
  - Un insieme di test case $T = \{t_1, t_2, ..., t_n\}$
  - Un programma P
- **Problema**: Trovare un **sottoinsieme minimale** $T^k \subseteq T$ che testi/esegua le stesse parti di codice eseguite da T.

Test Suite Minimization

- **Input**:
  - Un insieme di test case $T = \{t_1, t_2, ..., t_n\}$
  - Un programma P
- **Problema**: Trovare un **sottoinsieme minimale** $T^k \subseteq T$ che testi/esegua le stesse parti di codice eseguite da T.

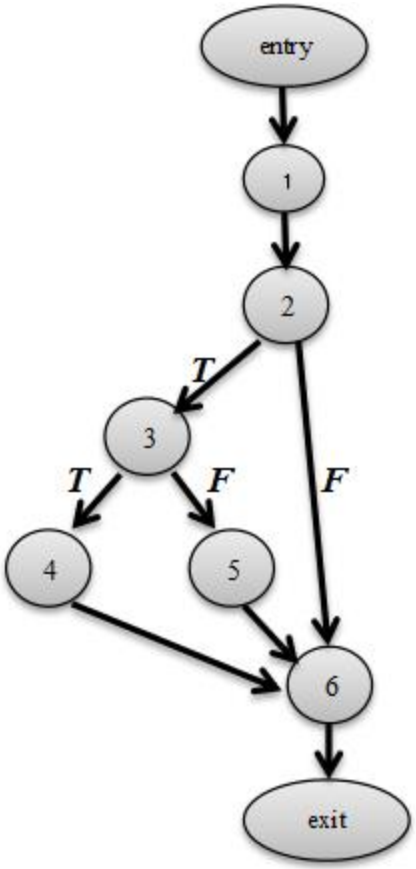
Criteri di copertura del codice:

1. Statement Coverage
2. Branch Coverage

…

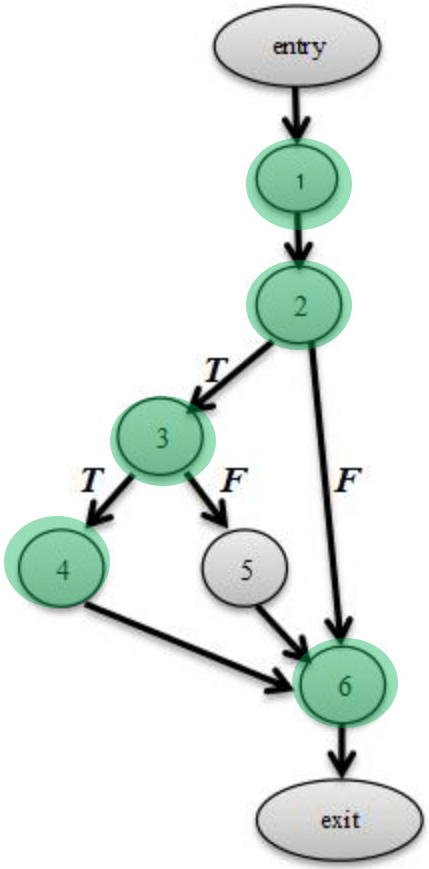
Criteri di copertura

```
Program Example
integer i, j, k
1.  read i, j, k
2.  if(i < j)
3.      if(j < k)
4.          i = k;
        else
5.          k = i;
        endif
    endif
6.  print i, j, k
    end Example
```



Statements Coverage

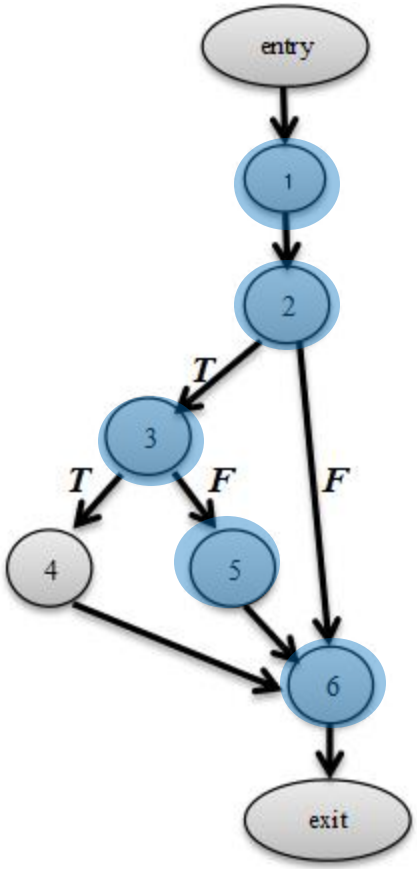
```
    Program Example
    integer i, j, k
1.  read i, j, k
2.  if(i < j)
3.      if(j < k)
4.          i = k;
        else
5.          k = i;
        endif
    endif
6.  print i, j, k
    end Example
```



TC1
Input: i=1 j=2 k=3
Statements: 1,2,3,4, 6 **(83%)**

Statements Coverage

```
    Program Example
    integer i, j, k
1.  read i, j, k
2.  if(i < j)
3.      if(j < k)
4.          i = k;
        else
5.          k = i;
        endif
    endif
6.  print i, j, k
    end Example
```



TC1

Input:	i=1	j=2	k=3

Statements: 1,2,3,4,6 **(83%)**

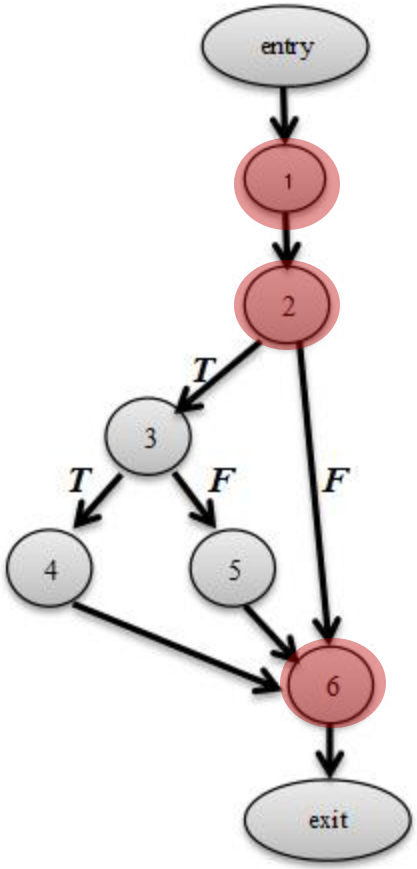
TC2

Input:	i=1	j=4	k=3

Statements: 1,2,3,5,6 **(83%)**

Statements Coverage

```
     Program Example
     integer i, j, k
1.   read i, j, k
2.   if(i < j)
3.        if(j < k)
4.             i = k;
          else
5.             k = i;
          endif
     endif
6.   print i, j, k
     end Example
```



TC1

Input: i=1 j=2 k=3

Statements: 1,2,3,4,6 **(83%)**

TC2

Input: i=1 j=4 k=3

Statements: 1,2,3,5,6 **(83%)**

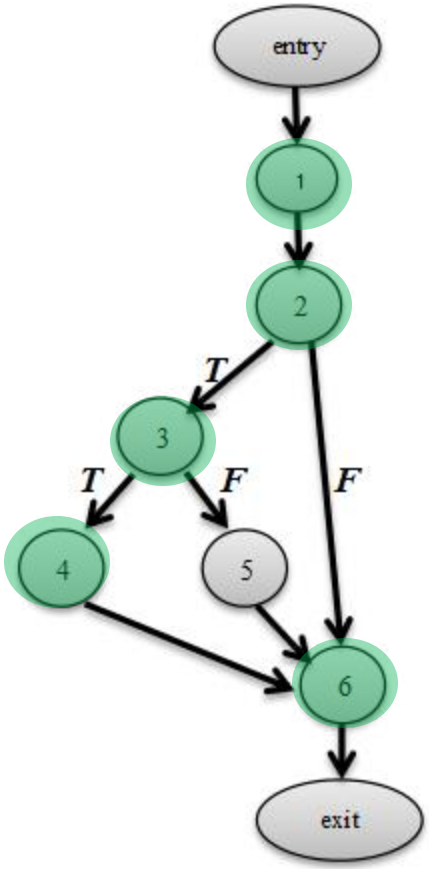
TC3

Input: i=10 j=2 k=3

Statements: 1,2,6 **(50%)**

Branches Coverage

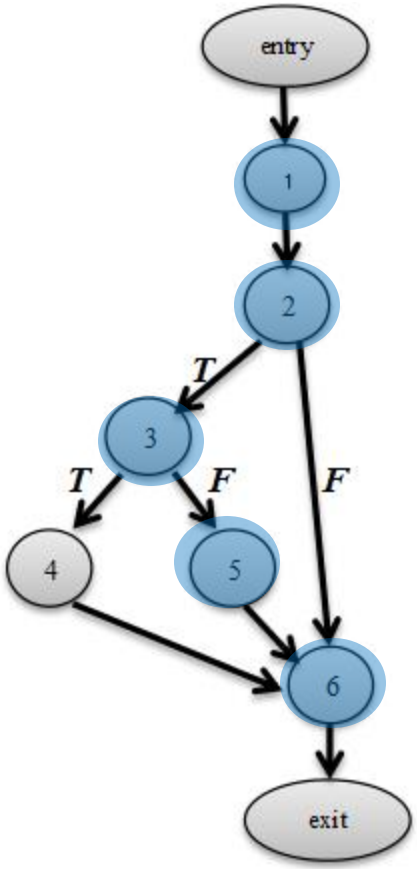
```
Program Example
integer i, j, k
1.  read i, j, k
2.  if(i < j)
3.      if(j < k)
4.          i = k;
        else
5.          k = i;
        endif
    endif
6.  print i, j, k
    end Example
```



TC1			
Input:	i=1	j=2	k=3
Branches:	67%		

Branches Coverage

```
Program Example
integer i, j, k
1.  read i, j, k
2.  if(i < j)
3.      if(j < k)
4.          i = k;
        else
5.          k = i;
        endif
    endif
6.  print i, j, k
    end Example
```



TC1

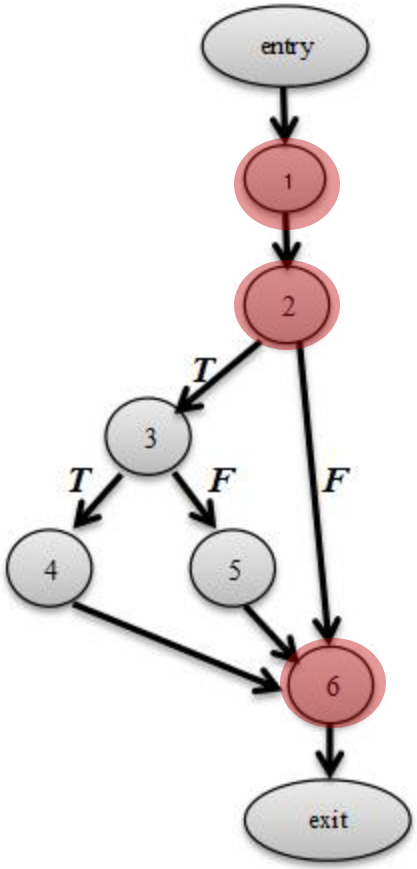
Input:	i=1	j=2	k=3
Branches:	**67%**		

TC2

Input:	i=1	j=4	k=3
Branches:	**67%**		

Branches Coverage

```
Program Example
integer i, j, k
1.  read i, j, k
2.  if(i < j)
3.      if(j < k)
4.          i = k;
        else
5.          k = i;
        endif
    endif
6.  print i, j, k
    end Example
```



TC1

Input:	i=1	j=2	k=3
Branches:	67%		

TC2

Input:	i=1	j=4	k=3
Branches:	67%		

TC3

Input:	i=10	j=2	k=3
Branches:	44%		

Test Suite Minimization (2)

- **Input**:
  - Un insieme di test case $T = \{t_1, t_2, ..., t_n\}$
  - Un insieme di elementi del codice da coprire (statement, ecc.)
  - Un matrice di copertura

	t1	t2	…	tn
E1	1	0	…	1
E2	0	0	…	0
…	…	…	…	…
Ek	1	1	…	0

$$M_{ij} = \begin{cases} 1 & \text{se l'elemento } E_i \text{ è coperto dal tc } t_j \\ 0 & \text{altrimenti} \end{cases}$$

- **Problema**: Trovare un **sottoinsieme minimale** $T^k \subseteq T$ che testi/esegua le stesse parti di codice eseguite da T.

Test Suite Minimization (2)

- **Input**:
  - Un insieme di test case $T = \{t_1, t_2, ..., t_n\}$
  - Un insieme di elementi del codice da coprire (statement, ecc.)
  - Un matrice di copertura

	t1	t2	...	tn
E1	1	0	...	1
E2	0	0	...	0
...	...	...	...	...
Ek	1	1	...	0

$$M_{ij} = \begin{cases} 1 & \text{se l'elemento } E_i \text{ è coperto dal tc } t_j \\ 0 & \text{altrimenti} \end{cases}$$

- **Problema**: trovare un vettore $X = \{x_1, x_2, ..., x_n\}$

$$x_i = \begin{cases} 1 & \text{se il tc } t_i \text{ è selez.} \\ 0 & \text{altrimenti} \end{cases}$$

$$\min \; size(X) = \sum_{i=1}^{n} x_i \qquad \text{s.v.} \qquad \text{cov}(X) = M \cdot X^T \geq 1$$

Test Suite Minimization

- Il problema di **Test Suite Minimimazion** è **NP-completo**

Test Suite Minimimazion

Input:
- Programma $P = \{e_1, e_2, ..., e_n\}$
- una collezione di test case che coprono parti di P, $T = \{t_1, t_2, ..., t_n\}$

Problema:
Trovare il piu piccolo insieme $T^* \subseteq T$ tale che $\bigcup_{t \in T^*} t(P) = P$

Hitting set

Input:
- Insieme universo $P = \{e_1, e_2, ..., e_n\}$
- una collezione di sottoinsiemi di P, $T = \{t_1, t_2, ..., t_n\}$

Problema:
Trovare il piu piccolo insieme $T^* \subseteq T$ tale che $\bigcup_{t \in T^*} t_i = P$

Algoritmi di Ottimizzazione

- Il problema di **Test Suite Minimimazion** è **NP-completo**

- Utilizzare algoritmi di approssimazione:
  - Algoritmo **Greedy** dell'**hitting set**
  - Algoritmo **Additional Greedy** dell'**hitting set**

- Utilizzare altre euristiche
  - **Algoritmi Genetici**
  - Algoritmi di ricerca casuale (algoritmi random)

Algoritmo Greedy

L'algoritmo greedy parte da un insieme vuoto di test case ed incrementalmente aggiunge il test case che copre il maggior numero di elementi (e.g. istruzioni) del programma da testare.

	Greedy (P, T)
(1)	C ←Ø insieme di elementi coperti di P
(2)	S ←Ø insieme TC selezionati
(3)	W ←T Worklist inizialmente contenente T
(4)	**repeat**
(5)	Tj← test case in W con massima copertura
(6)	C ← C ∪ Cov(Tj)
(7)	Rimuovi Tj da W
	S ← S ∪ {Tj}
(8)	**until** C=Cov(T)

Algoritmo Greedy (1)

	T1	T2	T3	T4
Istruzione 1	X			
Istruzione 2	X	X		
Istruzione 3	X	X		X
Istruzione 4		X	X	X
Istruzione 5			X	X
Istruzione 6	X			
Istruzione 7	X	X		
Istruzione 8				X
Istruzione 9	X	X		
Istruzione 10			X	

Algoritmo Greedy (2)

	T1	T2	T3	T4
Istruzione 1	X			
Istruzione 2	X	X		
Istruzione 3	X	X		X
Istruzione 4		X	X	X
Istruzione 5			X	X
Istruzione 6	X			
Istruzione 7	X	X		
Istruzione 8				X
Istruzione 9	X	X		
Istruzione 10			X	

TC	Copertura
T1	60%
T2	50%
T3	30%
T4	40%

Inizializzazione

$C = \emptyset \quad S = \emptyset$

$T = \{T1, T2, T3, T4\}$

Algoritmo Greedy (3)

	T1	T2	T3	T4
Istruzione 1	X			
Istruzione 2	X	X		
Istruzione 3	X	X		X
Istruzione 4		X	X	X
Istruzione 5			X	X
Istruzione 6	X			
Istruzione 7	X	X		
Istruzione 8				X
Istruzione 9	X	X		
Istruzione 10			X	

TC	Copertura
T1	60%
T2	50%
T3	30%
T4	40%

Inizializzazione

C = Ø S = Ø

T = {T1, T2, T3, T4}

Iterazione 1

T1 ha la massima coverage

C = C U {1,2,3,6,7,9} = {1,2,3,6,7,9}

W = {T2, T3, T4}

S = {T1}

Algoritmo Greedy (4)

	T1	T2	T3	T4
Istruzione 1	X			
Istruzione 2	X	X		
Istruzione 3	X	X		X
Istruzione 4		X	X	X
Istruzione 5			X	X
Istruzione 6	X			
Istruzione 7	X	X		
Istruzione 8				X
Istruzione 9	X	X		
Istruzione 10			X	

TC	Copertura
T1	60%
T2	50%
T3	30%
T4	40%

Inizializzazione

C = Ø S = Ø

T = {T1, T2, T3, T4}

Iterazione 2

T2 ha la massima coverage

C = C U {2,3,4,7,8} = {1,2,3,4,6,7,8,9}

W = {T3, T4}

S = {T1, T2}

Algoritmo Greedy (5)

	T1	T2	T3	T4
Istruzione 1	X			
Istruzione 2	X	X		
Istruzione 3	X	X		X
Istruzione 4		X	X	X
Istruzione 5			X	X
Istruzione 6	X			
Istruzione 7	X	X		
Istruzione 8				X
Istruzione 9	X	X		
Istruzione 10			X	

TC	Copertura
T1	60%
T2	50%
T3	30%
T4	40%

Inizializzazione

C = Ø S = Ø

T = {T1, T2, T3, T4}

Iterazione 3

T4 ha la massima coverage

C = C U {3,4,5,8} = {1,2,3,4,5,6,7,8,9}

W = {T3}

S = {T1, T2, T4}

Algoritmo Greedy (6)

	T1	T2	T3	T4
Istruzione 1	X			
Istruzione 2	X	X		
Istruzione 3	X	X		X
Istruzione 4		X	X	X
Istruzione 5			X	X
Istruzione 6	X			
Istruzione 7	X	X		
Istruzione 8				X
Istruzione 9	X	X		
Istruzione 10			X	

TC	Copertura
T1	60%
T2	50%
T3	30%
T4	40%

Inizializzazione

C = Ø S = Ø

T = {T1, T2, T3, T4}

Iterazione 4

T3 ha la massima coverage

C = C U {4,5,10} = {1,2,3,4,5,6,7,8,9,10}

W = Ø

S = {T1, T2, T3, T4}

Algoritmo Greedy (7)

	T1	T2	T3	T4
Istruzione 1	X			
Istruzione 2	X	X		
Istruzione 3	X	X		X
Istruzione 4		X	X	X
Istruzione 5			X	X
Istruzione 6	X			
Istruzione 7	X	X		
Istruzione 8				X
Istruzione 9	X	X		
Istruzione 10			X	

TC	Copertura
T1	60%
T2	50%
T3	30%
T4	40%

Inizializzazione

C = Ø S = Ø

T = {T1, T2, T3, T4}

Iterazione 4

T3 ha la massima coverage

C = C U {4,5,10} = {1,2,3,4,5,6,7,8,9,10}

W = Ø

S = {T1, T2, T3, T4}

Test suite Minima

Additional Greedy

L'algoritmo AdditionalGreedy parte da un insieme vuoto di test case ed incrementalmente aggiunge il test case con la massima copertura "addizionale" rispetto alle istruzioni già coperte.

	AdditionalGreedy (P, T)
(1)	C ←Ø insieme di elementi coperti di P
(2)	S ←Ø insieme TC selezionati
(3)	W ←T Worklist inizialmente contenente T
(4)	**repeat**
(5)	Tj← test case in W con massimo **|Cov(Tj) - C|**
(6)	C ← C ∪ Cov(Tj)
(7)	Rimuovi Tj da W
	S ← S ∪ {Tj}
(8)	**until** C=Cov(T)

Additional Greedy(1)

	T1	T2	T3	T4
Istruzione 1	X			
Istruzione 2	X	X		
Istruzione 3	X	X		X
Istruzione 4		X	X	X
Istruzione 5			X	X
Istruzione 6	X			
Istruzione 7	X	X		
Istruzione 8				X
Istruzione 9	X	X		
Istruzione 10			X	

TC	Copertura	Copertura Addizionale
T1	60%	60%
T2	50%	50%
T3	30%	30%
T4	40%	40%

Inizializzazione

C = Ø S = Ø

T = {T1, T2, T3, T4}

Additional Greedy(2)

	T1	T2	T3	T4
Istruzione 1	X			
Istruzione 2	X	X		
Istruzione 3	X	X		X
Istruzione 4		X	X	X
Istruzione 5			X	X
Istruzione 6	X			
Istruzione 7	X	X		
Istruzione 8				X
Istruzione 9	X	X		
Istruzione 10			X	

TC	Copertura	Copertura Addizionale
T1	60%	60%
T2	50%	50%
T3	30%	30%
T4	40%	40%

Inizializzazione

C = Ø S = Ø

T = {T1, T2, T3, T4}

Iterazione 1

T1 ha la massima copertura addizionale

C = C U {1,2,3,6,7,9} = {1,2,3,6,7,9}

W = {T2, T3, T4}

S = {T1}

Additional Greedy(3)

	T1	T2	T3	T4
Istruzione 1	X			
Istruzione 2	X	X		
Istruzione 3	X	X		X
Istruzione 4		X	X	X
Istruzione 5			X	X
Istruzione 6	X			
Istruzione 7	X	X		
Istruzione 8				X
Istruzione 9	X	X		
Istruzione 10			X	

TC	Copertura	Copertura Addizionale
T1	60%	-
T2	50%	10%
T3	30%	30%
T4	40%	30%

Inizializzazione

C = Ø S = Ø

T = {T1, T2, T3, T4}

Iterazione 2

T3 ha la massima copertura addizionale

C = C U {4,5,10} = {1,2,3,4,5,6,7,9,10}

W = {T2, T4}

S = {T1, T3}

Additional Greedy(4)

	T1	T2	T3	T4
Istruzione 1	X			
Istruzione 2	X	X		
Istruzione 3	X	X		X
Istruzione 4		X	X	X
Istruzione 5			X	X
Istruzione 6	X			
Istruzione 7	X	X		
Istruzione 8				X
Istruzione 9	X	X		
Istruzione 10			X	

TC	Copertura	Copertura Addizionale
T1	60%	-
T2	50%	-
T3	30%	-
T4	40%	10%

Inizializzazione

C = Ø S = Ø

T = {T1, T2, T3, T4}

Iterazione 3

T4 ha la massima copertura addizionale

C = C U {8} = {1,2,3,4,5,6,7,8,9,10}

W = {T2,}

S = {T1, T3, T4}

Additional Greedy(5)

	T1	T2	T3	T4
Istruzione 1	X			
Istruzione 2	X	X		
Istruzione 3	X	X		X
Istruzione 4		X	X	X
Istruzione 5			X	X
Istruzione 6	X			
Istruzione 7	X	X		
Istruzione 8				X
Istruzione 9	X	X		
Istruzione 10			X	

TC	Copertura	Copertura Addizionale
T1	60%	-
T2	50%	-
T3	30%	-
T4	40%	10%

Inizializzazione

C = Ø S = Ø

T = {T1, T2, T3, T4}

Iterazione 3

T3 ha la massima copertura addizionale

C = C U {8} = {1,2,3,4,5,6,7,8,9,10}

W = {T2,}

S = {T1, T3, T4}

Test suite Minima

Test Suite Minimization

- Il problema di **Test Suite Minimimazion** è **NP-completo**

- Gli algoritmi Greedy sono algoritmi di approssimazione: *possono fornire soluzioni sub-ottimali*

- **Problemi**:
  - Trovare algoritmi di approssimazione migliori
  - *Una test suite minimizzata può rilevare meno difetti della test suite originale* (**Rothermel** et al. 1989, **Wong** et al. 1998)

Test Case Selection

- **Input**:
  - Un insieme di test case $T = \{t_1, t_2, ..., t_n\}$
  - Un programma P
- **Problema**: Trovare un **sottoinsieme** $T^k \subseteq T$ che (i) massimizzi il numero di elementi in P testati/coperti da T^k e (ii) minimizzi il costo di esecuzione

Copertura del codice e costo di esecuzione sono due obiettivi contrastanti

Test Case Selection

- **Input**:
  - Un insieme di test case $T = \{t_1, t_2, \ldots, t_n\}$
  - Un insieme di elementi del codice da coprire (statements, ecc.)
  - Un matrice di copertura M $M_{ij} = \begin{cases} 1 & \text{se l'elemento } E_i \text{ è coperto dal tc } t_j \\ 0 & \text{altrimenti} \end{cases}$
  - Informazioni sul costo/tempo di esecuzione dei test case

$$C = \{c_1, c_2, \ldots, c_n\}$$

- **Problema**: trovare un vettore $X = \{x_1, x_2, \ldots, x_n\}$ $x_i = \begin{cases} 1 & \text{se } t_i \text{ è selez.} \\ 0 & \text{altrimenti} \end{cases}$ tale che

$$\min \ cost(X) = \sum_{i=1}^{n} (x_i \cdot c_i)$$

$$\max \ cov(X) = M \cdot X^T$$

Test Case Selection

- Il problema di **Test Case Selection** è **NP-completo**

Test Case Selection

Input:
- Programma $P = \{e_1, e_2, ..., e_n\}$
- una collezione di test case che coprono parti di P, $T = \{t_1, t_2, ..., t_n\}$
- insieme dei costi $C = \{c_1, c_2, ..., c_n\}$

Problema:
Trovare l'insieme di costo minimo $T^* \subseteq T$ tale che

$$\bigcup_{t \in T^*} t(P) = P$$

Weighted Set Cover

Input:
- Insieme universo $P = \{e_1, e_2, ..., e_n\}$
- una collezione di sottoinsiemi di P, $T = \{t_1, t_2, ..., t_n\}$
- Insieme di costi $C = \{c_1, c_2, ..., c_n\}$

Problema:
Trovare l'insieme di costo minimo $T^* \subseteq T$ tale che

$$\bigcup_{t \in T^*} t_i = P$$

Test Case Selection

- Il problema di **Test Case Selection** è **NP-completo**

- Utilizzare algoritmi di approssimazione:
  - Algoritmo **Greedy** del problema di **weighted set cover**
  - Algoritmo **Additional Greedy** del problema di **weighted set cover**

- Utilizzare altre euristiche
  - **Algoritmi Genetici**
  - Algoritmi di ricerca casuale (algoritmi random)

Algoritmo Greedy

L'algoritmo greedy parte da un insieme vuoto di test case ed incrementalmente aggiunge il test case con la massima **copertura per unità di tempo**

	Greedy (P, T)
(1)	C ←Ø insieme di elementi coperti di P
(2)	S ←Ø insieme TC selezionati
(3)	W ←T Worklist inizialmente contenente T
(4)	**repeat**
(5)	Tj← test case in W con massimo |Cov(Tj)| / Cj
(6)	C ← C U Cov(Tj)
(7)	Rimuovi Tj da W
	S ← S U {Tj}
(8)	**until** C=Cov(T)

Coverage per unità di tempo

Additional Greedy

L'algoritmo AdditionalGreedy parte da un insieme vuoto di test case ed incrementalmente aggiunge il test case con la massima **copertura "addizionale" per unità di tempo** rispetto alle istruzioni già coperte.

	AdditionalGreedy (P, T)
(1)	C ←Ø insieme di elementi coperti di P
(2)	S ←Ø insieme TC selezionati
(3)	W ←T Worklist inizialmente contenente T
(4)	**repeat**
(5)	Tj← test case in W con max **|Cov(Tj) - C|/ Cj**
(6)	C ← C U Cov(Tj)
(7)	Rimuovi Tj da W
	S ← S U {Tj}
(8)	**until** C=Cov(T)

Coverage "addizionale" per unità di tempo

Additional Greedy(1)

	T1	T2	T3	T4
Istruzione 1	X	X	X	X
Istruzione 2	X	X	X	X
Istruzione 3	X		X	X
Istruzione 4	X	X		X
Istruzione 5	X	X		X
Istruzione 6	X	X		
Istruzione 7	X	X		
Istruzione 8	X	X		
Istruzione 9		X	X	
Istruzione 10		X		
Costo (sec.)	4	5	3	3

TC	Copertura	Copertura Add / Costo
T1	80%	80/4 = 20
T2	90%	90 / 5 =18
T3	40%	40/3 = 13
T4	50%	50/3 = 17

Inizializzazione

C = Ø S = Ø

T = {T1, T2, T3, T4}

Additional Greedy(2)

	T1	T2	T3	T4
Istruzione 1	X	X	X	X
Istruzione 2	X	X	X	X
Istruzione 3	X		X	X
Istruzione 4	X	X		X
Istruzione 5	X	X		X
Istruzione 6	X	X		
Istruzione 7	X	X		
Istruzione 8	X	X		
Istruzione 9		X	X	
Istruzione 10		X		
Costo (sec.)	4	5	3	3

TC	Copertura	Copertura Add / Costo
T1	80%	80/4 = 20
T2	90%	90 / 5 =18
T3	40%	40/3 = 13
T4	50%	50/3 = 17

Inizializzazione

C = Ø S = Ø

T = {T1, T2, T3, T4}

Iterazione 1

T1 ha la massima **Copertura Add / Costo**

C = C U {1,2,3,4,5,6,7,8} = {1,2,3,4,5,6,7,8}

W = {T2, T3, T4}

S = {T1}

Additional Greedy(3)

	T1	T2	T3	T4
Istruzione 1	X	X	X	X
Istruzione 2	X	X	X	X
Istruzione 3	X		X	X
Istruzione 4	X	X		X
Istruzione 5	X	X		X
Istruzione 6	X	X		
Istruzione 7	X	X		
Istruzione 8	X	X		
Istruzione 9		X	X	
Istruzione 10		X		
Costo (sec.)	4	5	3	3

TC	Copertura	Copertura Add / Costo
T1	80%	-
T2	90%	20 / 5 = 4
T3	40%	10 / 3 = 3
T4	50%	10 / 3 = 3

Inizializzazione

C = Ø S = Ø

T = {T1, T2, T3, T4}

Iterazione 2

T2 ha la massima **Copertura Add / Costo**

C = C U {1,2,4,5,6,7,8,9,10}= {1,2,3,4,5,6,7,8,9,10}

W = {T3, T4}

S = {T1, T2}

Additional Greedy(4)

	T1	T2	T3	T4
Istruzione 1	X	X	X	X
Istruzione 2	X	X	X	X
Istruzione 3	X		X	X
Istruzione 4	X	X		X
Istruzione 5	X	X		X
Istruzione 6	X	X		
Istruzione 7	X	X		
Istruzione 8	X	X		
Istruzione 9		X	X	
Istruzione 10		X		
Costo (sec.)	4	5	3	3

TC	Copertura	Copertura Add / Costo
T1	80%	-
T2	90%	20 / 5 = 4
T3	40%	10 / 3 = 3
T4	50%	10 / 3 = 3

Soluzione Finale

T = {T1, T2}

Cov = 100%

Cost = 9

Additional Greedy(5)

	T1	T2	T3	T4
Istruzione 1	X	X	X	X
Istruzione 2	X	X	X	X
Istruzione 3	X		X	X
Istruzione 4	X	X		X
Istruzione 5	X	X		X
Istruzione 6	X	X		
Istruzione 7	X	X		
Istruzione 8	X	X		
Istruzione 9		X	X	
Istruzione 10		X		
Costo (sec.)	4	5	3	3

TC	Copertura	Copertura Add / Costo
T1	80%	-
T2	90%	20 / 5 = 4
T3	40%	10 / 3 = 3
T4	50%	10 / 3 = 3

Soluzione Finale

T = {T1, T2}

Cov = 100%

Cost = 9

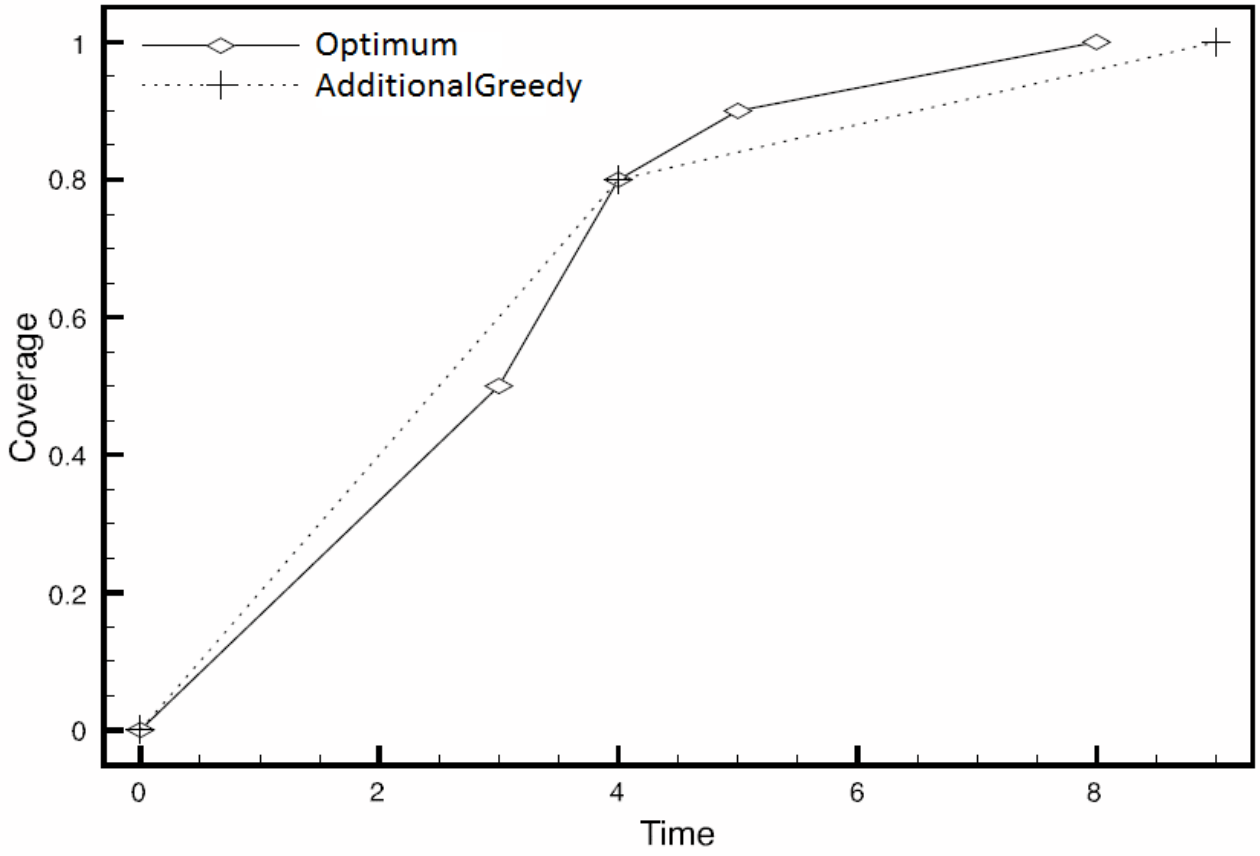
Soluzione Ottima

T = {T2, T3}

Cov = 100%

Cost = 8

Additional Greedy(6)



Test Case Selection

- Il problema di **Test Case Selection** è **NP-completo**

- Gli algoritmi Greedy sono algoritmi di approssimazione: *possono fornire soluzioni sub-ottimali*

- **Problemi**:
  - Trovare algoritmi di approssimazione migliori
  - Trovare il maggior numero di compromessi ottimali tra (i) costo e (ii) coverage
  - *Una test suite ottimizzata può rilevare meno difetti della test suite originale*

Test Case Prioritization

- **Input**:
  - Un insieme di test case $T = \{t_1, t_2, ..., t_n\}$
  - Un programma P

- **Problema**: Trovare un **ordinamento** ottimale dei test case **T***, in modo tale da eseguire per prima i test case che testano parti di codice difettose.

Motivazioni

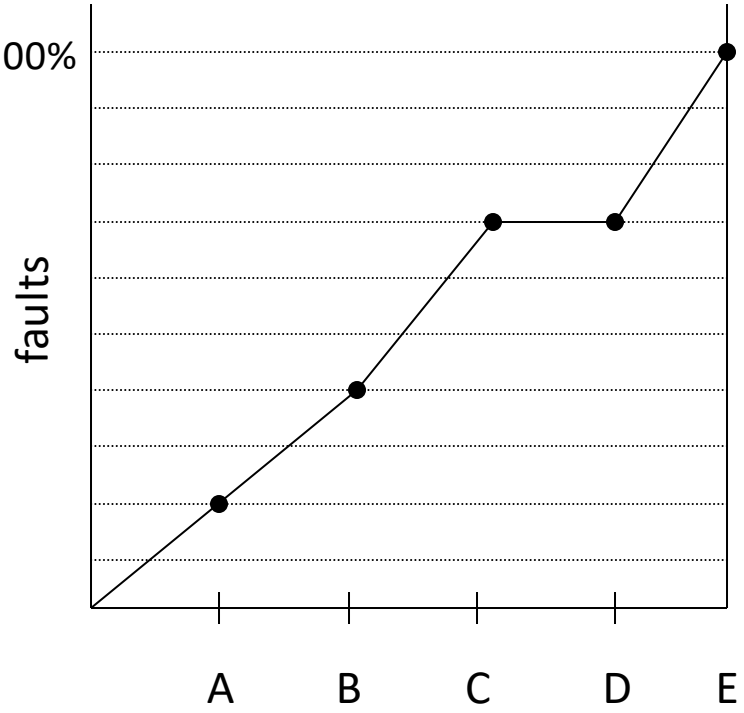
L'**ordine** con cui sono eseguiti i test case influisce:

- Sulla **percentuale di fault identificati**: un buon ordinamento rileva fault prima di altri, oppure
  - ✓ Rivelano per prima **fault più critici**
  - ✓ Rilevano per prima fault di **parti di codice più critiche**
- Sulla percentuale di codice testato: un buon ordinamento consente di raggiungere buoni livelli di **copertura** prima di altri ordinamenti.

Esempio

test	fault 1	2	3	4	5	6	7	8	9	10
A	X				X					
B	X				X	X	X			
C	X	X	X	X	X	X	X			
D										
E								X	X	X

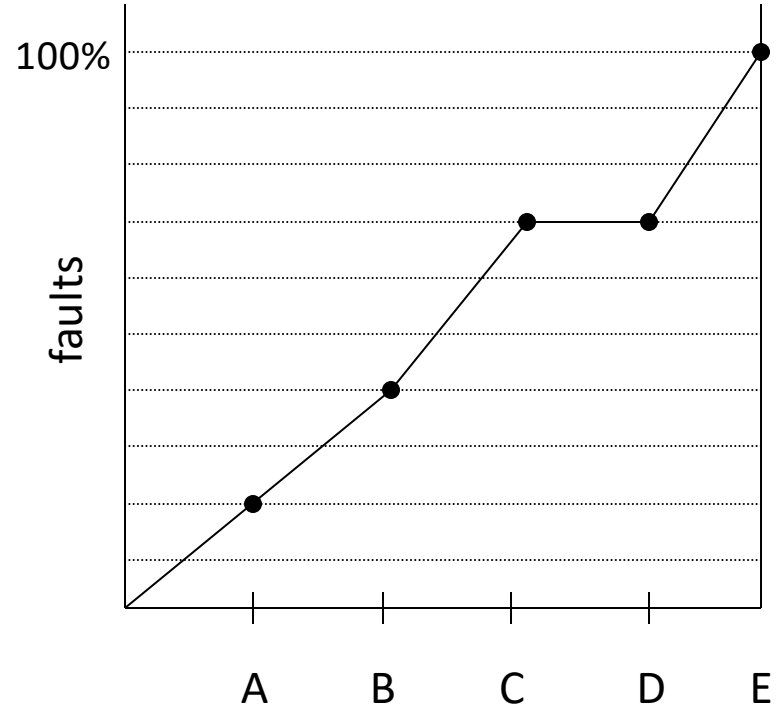
Ordinamento = A,B,C,D,E



Ordinamento dei test case

Esempio

test	fault 1	2	3	4	5	6	7	8	9	10
A	X				X					
B	X				X	X	X			
C	X	X	X	X	X	X	X			
D										
E								X	X	X



Ordinamento = A,B,C,D,E

Ordinamento dei test case

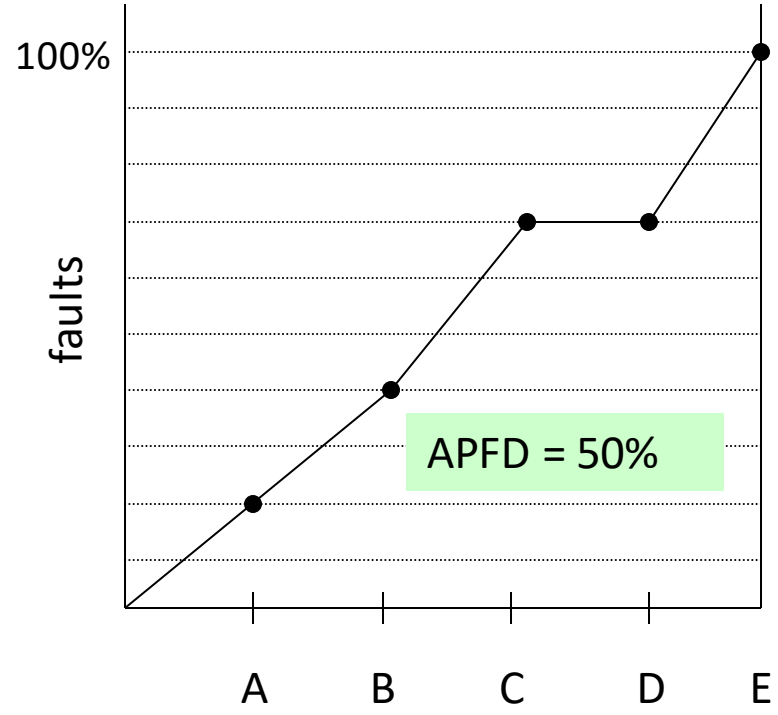
APFD: Average Percentage of Faults Detected.

$$APFD = 1 - \frac{TF_1 + TF_2 + \ldots + TF_m}{n \times m} + \frac{1}{2 \times n}$$

n = # test case
m = # fault
TFi = posizione del primo test case che rileva il fault Fi

Esempio

test	fault 1	2	3	4	5	6	7	8	9	10
A	X				X					
B	X				X	X	X			
C	X	X	X	X	X	X	X			
D										
E								X	X	X



Ordinamento = A,B,C,D,E

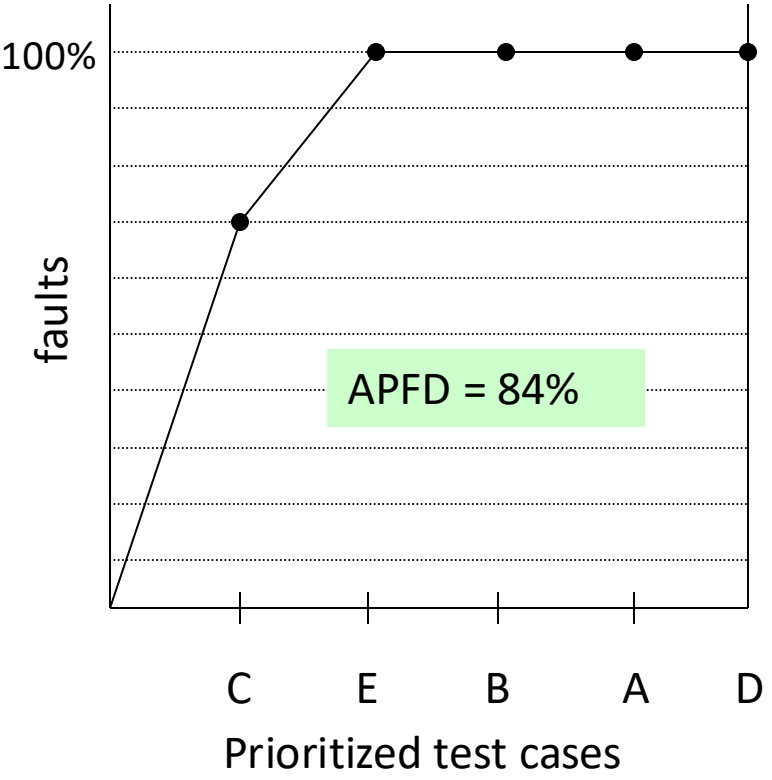
Ordinamento dei test case

APFD: Average Percentage of Faults Detected.

$$APFD = 1 - \frac{1+3+3+3+1+2+2+5+5+5}{5 \cdot 10} + \frac{1}{2 \cdot 5} = 0.5$$

Esempio (2)

test	fault 1	2	3	4	5	6	7	8	9	10
A	X				X					
B	X				X	X	X			
C	X	X	X	X	X	X	X			
D										
E								X	X	X



APFD: Average Percentage of Faults Detected.

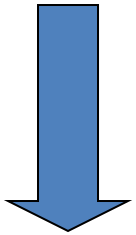
$$APFD = 1 - \frac{1+1+1+1+1+1+1+2+2+2}{5 \cdot 10} + \frac{1}{2 \cdot 5} = 0.84$$

Problema!

*Il numero di fault identificati da un test case è **sconosciuto** fino a quando il test case non viene eseguito e il suo output non viene valutato sulla base di un oracolo.*

Problema!

*Il numero di fault identificati da un test case è **sconosciuto** fino a quando il test case non viene eseguito e il suo output non viene valutato sulla base di un oracolo.*



Si usano euristiche per derivare quali test case hanno maggiore probabilità di rilevare un bug.

- *Code Coverage* (Test case che coprono più codice)
- *Past fault Coverage* (Test case che in passato hanno rivelato fault)

Misure di copertura del codice per Test Case Prioritization

Così come per la copertura dei fault, anche per confrontare la copertura degli elementi di codice di due permutazioni di casi di test bisogna usare la Average Percentage of Element Coverage (APEC)

APEC: Average Percentage of Element Coverage.

$$APEC = 1 - \frac{\sum_{i=1}^{m} TE_i}{n \times m} + \frac{1}{2 \times n}$$

n = # test case
m = # fault
TEi = posizione del primo test case che copre l'elemento Ei

Test Case Prioritization (2)

- **Input**:
  - Un insieme di test case $T = \{t_1, t_2, ..., t_n\}$
  - Un insieme di elementi del codice da coprire (statements, ecc.)
  - Una matrice di copertura del codice $M_{ij} = \begin{cases} 1 & \text{se } E_i \text{ è coperto da } t_j \\ 0 & \text{altrimenti} \end{cases}$
  - Una matrice di copertura dei fault passati

- **Problema**: trovare un ordinamento T* di T tale che

$$\max \text{CodeCoverage}(T^*) = \text{APEC}(T^*)$$
$$\max \text{PastFaultCoverage}(T^*) = \text{APFD}_{\text{past}}(T^*)$$

Misure di copertura del codice per Test Case Prioritization

Nella formulazione precedente, il costo di esecuzione di ogni caso di test è considerato unitario.

Se vogliamo considerare anche il costo di esecuzione c_i di ciascun caso di test t_i (costo da minimizzare), allora APEC e APFD devono essere riformulate nella versione cost cognizant

$$APFD_c = \frac{\sum_{i=1}^{m}\left(\sum_{j=TF_i}^{n} c_j - \frac{1}{2}c_{TF_i}\right)}{\sum_{i=1}^{n} c_i \times m}$$

$$APEC_c = \frac{\sum_{i=1}^{m}\left(\sum_{j=TE_i}^{n} c_j - \frac{1}{2}c_{TE_i}\right)}{\sum_{i=1}^{n} c_i \times m}$$

Nel caso in cui $c_i = 1$ per ogni i, APECc = APEC e APFDc = APFD

Test Case Prioritization (3)

- **Input**:
  - Un insieme di test case $T = \{t_1, t_2, \ldots, t_n\}$
  - Un insieme di elementi del codice da coprire (statements, ecc.)
  - Una matrice di copertura del codice $M_{ij} = \begin{cases} 1 & \text{se } \mathrm{E_i} \text{ è coperto da } t_j \\ 0 & \text{altrimenti} \end{cases}$
  - Una matrice di copertura dei fault passati

$$PF_{ij} = \begin{cases} 1 & \text{se il fault } \mathrm{F_i} \text{ è coperto da } t_j \\ 0 & \text{altrimenti} \end{cases}$$

- **Problema**: trovare un ordinamento T* di T tale che

$$\max \mathrm{CodeCoverage}(T^*) = \mathrm{APECc}(T^*)$$
$$\max \mathrm{PastFaultCoverage}(T^*) = \mathrm{APFDc}_{\mathrm{past}}(T^*)$$

Test Case Prioritization (4)

- Il problema di **Test Case Prioritization** è **NP-completo**

	AdditionalGreedy (P, T)
(1)	C ←Ø insieme di elementi coperti di P
(2)	PastFault←Ø insieme fault coperti
(3)	S ←Ø lista dei TC selezionati
(4)	W ←T Worklist inizialmente contenente T
(5)	**repeat**
(6)	Tj← test case in W con max**|Cov(Tj) - Cov| * |PF(Tj) - PastFault|/|Cj|**
(7)	C ← C ∪ Cov(Tj)
(8)	PastFault ← PastFault ∪ PF(Tj)
(9)	Rimuovi Tj da W
(10)	Inserisci Tj nella lista S
(11)	**until** C=Cov(T)

In breve…

Ottimizzazione di una test suite per il testing di regressione

Problemi:

1) Test Suite Minimization

(i) minimizzare numero di test case

2) Test Case Selection

(i) massimizzare la copertura del codice

(ii) minimizzare il costo di esecuzione

3) Test Case Prioritization

(i) massimizzare la copertura del codice

(ii) minimizzare il costo di esecuzione

(iii) massimizzare APFD dei fault passati