|  |
| --- |
| [Versione 1.0] |



MANUALE D’UTILIZZO DEI WISE CRUNCH TOOLS

Versione 1.0

**Data:** 19/02/2014

**Codice:** ?????

**Versione:** 01

**Circolazione:** **Interna, Cliente.**

Contenuti

[Test combinatoriale 3](#_Toc380582144)

[1-wise testing. 4](#_Toc380582145)

[2-wise testing o pairwise testing 5](#_Toc380582146)

[n-wise testing con n>2 8](#_Toc380582147)

[Wise crunch tools 9](#_Toc380582148)

[Procedura di installazione 10](#_Toc380582149)

[Procedura di configurazione 12](#_Toc380582150)

[Panoramica generale dei WiseCrunchTools 13](#_Toc380582151)

[Convenzioni lessicali e tipografiche 16](#_Toc380582152)

[Tool di primo livello – script batch DOS 17](#_Toc380582153)

[Come convenzione, il nome di tutti gli script batch segue lo schema: 17](#_Toc380582154)

[Tool runW 17](#_Toc380582155)

[Tool runCC e runsCC 17](#_Toc380582156)

[Tools runT e runsT 18](#_Toc380582157)

[Tools runTS e runsTS 19](#_Toc380582158)

[Tools runTSF e runsTSF 20](#_Toc380582159)

[Tool runR 21](#_Toc380582160)

[Tool runC 21](#_Toc380582161)

[Tool di secondo livello – eseguibili 23](#_Toc380582162)

[Eseguibili calcolacopertura.exe e calcolaCoperturaSlow.exe 23](#_Toc380582163)

[Eseguibile Combinazioni\_n\_k 23](#_Toc380582164)

[Eseguibili generaTestSet.exe e generaTestSetSlow.exe 24](#_Toc380582165)

[Eseguibile ProdCart.exe 25](#_Toc380582166)

[Eseguibile reduceNple.exe 26](#_Toc380582167)

[Eseguibili di utility 27](#_Toc380582168)

[Evoluzione del documento 29](#_Toc380582169)

# Test combinatoriale

Quando si abbia un’applicazione con molteplici input, ciascuno dei quali può assumere diversi valori, è – in genere – impossibile eseguire il test di tutte le possibili combinazioni dei valori delle variabili d’input, semplicemente perché sono troppe.

Facciamo subito un esempio: consideriamo un’applicazione che accetti in input tre possibili valori A, B e C. Tali valori possono essere scelti in modo arbitrario dalla seguente tabella:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **C** |
| A1 | B1 | C1 |
| A2 | B2 | C2 |
| A3 | B3 |  |
| A4 |  |  |
| # Valori | | |
| 4 | 3 | 2 |

**Tabella 1 – Variabili e valori**

Il numero totale di possibili combinazioni delle variabili (A,B,C) è pari a ; in pratica, per poter essere certi di garantire aver provato almeno una volta tutte le combinazioni possibili dei valori delle variabili (A,B,C) occorre effettuare 24 casi di test. Tali combinazioni sono le seguenti:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1-4** | **5-8** | **9-12** | **13-16** | **17-20** | **21-24** |
| A1;B1;C1 | A1;B3;C1 | A2;B2;C1 | A3;B1;C1 | A3;B3;C1 | A4;B2;C1 |
| A1;B1;C2 | A1;B3;C2 | A2;B2;C2 | A3;B1;C2 | A3;B3;C2 | A4;B2;C2 |
| A1;B2;C1 | A2;B1;C1 | A2;B3;C1 | A3;B2;C1 | A4;B1;C1 | A4;B3;C1 |
| A1;B2;C2 | A2;B1;C2 | A2;B3;C2 | A3;B2;C2 | A4;B1;C2 | A4;B3;C2 |

**Tabella 2 – Combinazioni dei valori delle variabili A,B,C**

Ora, nel caso specifico,un tale numero di test può ancora essere abbordabile. Se tuttavia si considera il caso generale di N variabili *X1, X2, …Xk* , la prima che assume *n1* possibili valori, la seconda *n2* possibili valori, la k-ma che assume *nk* possibili valori, il numero totale di combinazioni è pari a:  che, anche per valori non elevati di *n1*, *n2* ,…, *nk* è un valore grande. Ad esempio se k=5 ed (*n1*=3; *n2*=4; *n3=2;* *n4=2;* *n*5=3) si ottiene un numero di combinazioni pari a  che è già un bel numero di test da eseguire se si vuole garantire la copertura completa di tutte le combinazioni.

Se poi – come capita sovente nella pratica – il numero di valori *ni* che possono assumere le variabili è elevato, si fa presto a raggiungere le centinaia di migliaia (o milioni) di combinazioni, cosa che rende improponibile eseguire dei test completi, su tutte le combinazioni.

Come possiamo fare per portare a termine un test comunque efficace quando il numero di variabili e di valori è così alto da rendere impossibile un test esaustivo di tutte le combinazioni? Quali tecniche di riduzione applicare?

# 1-wise testing.

Nei casi in cui le combinazioni siano elevate, è possibile per lo meno verificare che - almeno una volta - ciascun singolo valore delle variabili sia stato dato in input al programma da testare. In altri termini, se la variabile A può assumere i valori A1, A2, A3 occorre almeno eseguire un primo test in cui la variabile A=A1, un secondo test in cui A=A2 e un terzo test in cui la variabile A=A3; lo stesso dicasi per le altre variabili. Tale tipo di test fornisce una copertura cosiddetta **wise-1**, e ne vedremo tra breve il significato. In pratica, abbiamo la seguente tabella:

|  |  |  |  |
| --- | --- | --- | --- |
| **# TEST** | **A** | **B** | **C** |
| 1 | A1 | \* | \* |
| 2 | A2 | \* | \* |
| 3 | A3 | \* | \* |
| 4 | A4 | \* | \* |
| 5 | \* | B1 | \* |
| 6 | \* | B2 | \* |
| 7 | \* | B3 | \* |
| 8 | \* | \* | C1 |
| 9 | \* | \* | C2 |

**Tabella 3 – Insieme di test 1-wise massimo**

Una prima riduzione consiste nel fare assumere a una variabile tutti i valori consecutivi, inserendo per le altre variabili un valore qualunque (indicato con \* nella **Tabella 3**) e procedendo in questo modo per tutte le variabili e valori. In questo modo si riducono i casi da 24 a soli 9. Il numero di casi si può ancora ulteriormente ridurre “sfruttando” il fatto che al posto degli \* si può selezionare un valore della variabile che poi può essere escluso dai casi di test successivi.

In pratica, per il caso di test # 1 al posto di B=\* inseriamo B=B1, al posto di C=\* inseriamo C=C1 ed eliminiamo il caso di test # 5 e il caso di test #8, che sono entrambi coperti dal caso di test # 1;

Per il caso di test # 2, al posto di B=\* inseriamo B=B2 e al posto di C=\* inseriamo C=C2 ed eliminiamo i casi di test # 6 e # 9 che sono entrambi coperti dal caso di test # 2.

Per il caso di test # 3, al posto di B=\* inseriamo B=B3 e al posto di C=\* inseriamo un qualunque valore C1 o C2; poiché i valori della variabile C pari a C1 e C2 sono già in effetti stati coperti dai casi di test # 1 e #2, possiamo lasciare C=\* rimandando a dopo la scelta se inserire C=C1 o C=C2. Eliminiamo il test #7 poiché B=B3 è adesso coperto dal caso di test # 3.

Avendo adesso capito il meccanismo, resta solo il caso di test # 4 che copre A=A4, mentre possiamo lasciare B=\* e C=\* demandando a dopo la scelta di cosa selezionare effettivamente.

In pratica il simbolo \* rappresenta il “don’t care” ossia, non modifica la copertura del test; tutte le variabili sono, almeno una volta, utilizzate nel caso di test reale e quelle con “\*” hanno dei valori che possono essere coperti 2 volte.

La tabella finale dei test ridotti wise-1 è la seguente:

|  |  |  |  |
| --- | --- | --- | --- |
| **# TEST** | **A** | **B** | **C** |
| 1 | A1 | B1 | C1 |
| 2 | A2 | B2 | C2 |
| 3 | A3 | B3 | \* |
| 4 | A4 | \* | \* |

**Tabella 4 – Insieme di test 1-wise minimo**

La **Tabella 4** è ottenuta dalla **Tabella 3** spostando verso l’alto le colonne della variabile B e C sino a ricoprire i valori con \*; il valore \* rimane nelle righe non coperte dai valori stessi (riga 3 variabile C e riga 4 variabile B e C).

La parola inglese “*wise*” sta ad indicare avveduto, astuto, prudente; ad esempio, *all-wise* significa “saggio”; quindi wise-1 sta a indicare qualcosa come “astuto a livello 1”, cosa che fa pensare che wise-2 sia un livello più alto di astuzia wise-1. Tale affermazione non è del tutto errata, anche se in italiano, potremmo tradurlo più correttamente come “modo” (anche se la parola inglese per “modo” è tutt’altra: way, manner, mode,…).

Dire quindi che un test set, come quello riportato in **Tabella 4** fornisce una copertura wise-1, significa affermare che ogni singolo valore di ogni variabile è coperto almeno una volta.

In pratica per il caso wise-1 il si ottiene la regola seguente:

“Date N variabili *X1, X2, …Xk* , la prima che assume *n1* possibili valori, la seconda *n2* possibili valori, la k-ma che assume *nk* possibili valori, il numero massimo di test che garantiscono la copertura wise-1 è pari a , mentre il numero minimo di test è pari al valore massimo tra { *n1* , *n2* ,…, *nk* }.“

Nella pratica, quello che interessa è sempre il numero minimo di casi di test che servono per garantire la copertura prescelta (e questo per ovvie ragioni).

# 2-wise testing o pairwise testing

Se il test 1-wise garantisce la copertura di ogni singolo valore di ciascuna variabile, è facile intuire che un insieme di test (nel seguito Test Set) a copertura wise-2, garantisce che tutte le coppie di valori delle variabili siano coperte almeno una volta. Nel caso delle variabili riportate nella **Tabella 1**, tutte le coppie delle **variabili** sono le seguenti: {(A,B), (A,C), (B,C)}. Infatti, il calcolo combinatoriale insegna che il numero di combinazioni di N valori presi K a K (con N≥K) è pari a:



Nel nostro caso di tre variabili (N=3) prese a due a due (K=2), applicando la formula sopra riportata abbiamo ; le tre coppie che sono proprio {(A,B), (A,C), (B,C)}.

Volendo andare a calcolare tutte le possibili coppie dei **valori** **delle variabili,** occorre considerare quanto segue:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **COPPIA** | **# VALORI VARIABILI** | | | **TOTALE PARZIALE** |
| **A** | **B** | **C** |
| (A,B) | 4 | 3 |  |  |
| (A,C) | 4 |  | 2 |  |
| (B,C) |  | 3 | 2 |  |
| **TOTALE COMPLESSIVO** | | | |  |

**Tabella 5 – Conteggio delle coppie di valori delle variabili A, B e C**

Quindi, il totale di coppie di tutti i valori delle variabili A, B e C i cui valori sono riportati nella **Tabella 1** è pari a 26 e sono quelle riportate nella tabella che segue:

| **#** | **# COPPIE VALORI** | | |
| --- | --- | --- | --- |
| **A, B** | **A, C** | **B, C** |
| 1 | A1,B1 | A1,C1 | B1, C1 |
| 2 | A1,B2 | A1,C2 | B1, C2 |
| 3 | A1,B3 | A2,C1 | B2, C1 |
| 4 | A2,B1 | A2,C2 | B2, C2 |
| 5 | A2,B2 | A3,C1 | B3, C1 |
| 6 | A2,B3 | A3,C2 | B3, C2 |
| 7 | A3,B1 | A4,C1 |  |
| 8 | A3,B2 | A4,C2 |  |
| 9 | A3,B3 |  |  |
| 10 | A4,B1 |  |  |
| 11 | A4,B2 |  |  |
| 12 | A4,B3 |  |  |
| **# COPPIE** | 12 | 8 | 6 |
| **TOTALE** | 12+8+6=26 | | |

**Tabella 6 – Coppie di valori delle variabili A, B e C**

Perché si dovrebbero considerare un Test Set a copertura wise-2? Non basterebbe considerare un Test Set con copertura wise-1? Qui entriamo in una questione spinosa, in cui i pareri sono vari, concordanti e discordanti.

Riportiamo di seguito l’incipit del sito <http://www.pairwise.org/> :

*“Pairwise (a.k.a. all-pairs) testing is an effective test case generation technique that is based on the observation that most faults are caused by interactions of at most two factors. Pairwise-generated test suites cover all combinations of two therefore are much smaller than exhaustive ones yet still very effective in finding defects.”*

Citiamo anche l’opinione di James Bach e Patrick J. Schroeder relativa al metodo di testing Pair Wise (“Pairwise Testing: A Best Practice That Isn’t” from James Bach, Patrick J. Schroeder disponibile in forma completa al link <http://www.testingeducation.org/wtst5/PairwisePNSQC2004.pdf> ):

“What do we know about the defect removal efficiency of pairwise testing? Not a great deal. Jones states that in the U.S., on average, the defect removal efficiency of our software processes is 85% [26]. This means that the combinations of all fault detection techniques, including reviews, inspections, walkthroughs, and various forms of testing remove 85% of the faults in software before it is released.

In a study performed by Wallace and Kuhn [27], 15 years of failure data from recalled medical devices is analyzed. They conclude that 98% of the failures could have been detected in testing if all pairs of parameters had been tested (they didn’t execute pairwise testing, they analyzed failure data and speculate about the type of testing that would have detected the defects). In this case, it appears as if adding pairwise testing to the current medical device testing processes could improve its defect removal efficiency to a "best in class" status, as determined by Jones [26].

On the other hand, Smith, et al. [28] present their experience with pairwise testing of the Remote Agent Experiment (RAX) software used to control NASA spacecraft. Their analysis indicates that pairwise testing detected 88% of the faults classified as correctness and convergence faults, but only 50% of the interface and engine faults. In this study, pairwise testing apparently needs to be augmented with other types of testing to improve the defect remove al efficiency, especially in the project context of a NASA spacecraft. Detecting only 50% of the interface and engine faults is well below the 85% U.S. average and presumably intolerable under NASA standards. The lesson here seems to be that one cannot blindly apply pairwise testing and expect high defect removal efficiency. Defect removal efficiency depends not only on the testing technique, but also on the characteristics of the software under test. As Mandl [4] has shown us, analyzing the software under test is an important step in determining if pairwise testing is appropriate; it is also an important step in determining what addition al testing technique should be used in a specific testing situation.”

**[4]** R. Mandl, "Orthogonal Latin Squares: An Application of Experiment Design to Compiler Testing," Communication of the ACM, vol. 28, no. 10, pp. 1054-1058, 1985.

**[26]** Jones, Software Assessments, Benchmarks, and Best Practices. Boston, MA: Addison Wesley Longman, 2000.

**[27]** D. R. Wallace and D. R. Kuhn, "Failure Modes in Medical Device Software: An Analysis of 15 Years of Recall Data," Int'l Jour. of Reliability, Quality and Safety Engineering, vol. 8, no. 4, pp. 351-371, 2001.

**[28]** B. Smith, M. S. Feather, and N. Muscettola, "Challenges and Methods in Testing the Remote Agent Planner," in Proc. 5th Int'l Conf. on Artificial Intelligence Planning and Scheduling (AIPS 2000), 2000, pp. 254-263

In pratica, il metodo di test Pairwise o 2-wise garantisce che tutte le combinazioni delle coppie di valori delle variabili siano testate, cosa che “dovrebbe garantire” la massimizzazione delle anomalie riscontrate, con percentuali che variano dal 50% al 98% secondo gli studi condotti. In effetti, nessun test potrà mai garantire una definita percentuale di rimozione dei difetti (cosa che si può **solo** calcolare **a consuntivo** per lo **specifico progetto**); diciamo – per essere realisti – che il Pairwise raggiunge un buon compromesso tra il numero di test da eseguire e le anomalie evidenziate, quando il numero di variabili e relativi valori in gioco è talmente alto da non poter essere effettuato un test a copertura di tutte le combinazioni (cosiddetto *all-wise testing* o *N-wise testing*, dove N è il numero delle variabili in gioco).

Nel caso di Test Set a copertura wise-2 è molto semplice conoscere il numero massimo di test che forniscono la copertura di tutte le coppie di valori delle variabili. Tale valore è pari al numero di coppie dei valori delle variabili stesse. Nel nostro esempio delle tre variabili A, B e C della **Tabella 1** il numero è pari a 26 (quello calcolato in **Tabella 6**). Il problema vero, quello della determinazione del numero minimo di test che garantisce la copertura wise-2, è a tuttora insoluto, pur esistendo una varietà di metodi e di algoritmi che approssimano tale valore per un problema ad un numero arbitrario di variabili e valori. Un esempio di tool che usano questi algoritmi sono: Microsoft **P**airwise **I**ndependent **C**ombinatorial **T**esting tool (PICT), scaricabile da <http://download.microsoft.com/download/f/5/5/f55484df-8494-48fa-8dbd-8c6f76cc014b/pict33.msi>, oppure AllPairs di James Bach scaricabile da <http://www.satisfice.com/tools.shtml> oppure ancora altri strumenti dei quali potete prendere visione al link <http://www.pairwise.org/tools.asp> .

# n-wise testing con n>2

A questo punto, è semplice estendere il concetto di test pairwise o 2-wise al caso generico di **n-wise**, con n>2. Il generico Test Set garantisce una copertura n-wise se è in grado di coprire tutte le n-ple (3-ple se n=3, 4-ple se n=4 e così via). Come nel caso Pairwise è sempre possibile conoscere le dimensioni del Test Set massimo, pari al numero di n-ple dei valori delle variabili, ma non esiste un metodo per conoscere – nel caso generale – la dimensione del Test Set Minimo che garantisce la copertura n-wise. Utilizzando PICT è possibile estrarre un Test Set che approssima il più possibile il Test Set minimo.

E’ chiaro che, dato un insieme di N variabili, il massimo livello di wise cui si può “aspirare” è pari al numero di variabili. Quindi, se abbiamo 4 variabili, un test set 4-wise coincide con tutte le combinazioni possibili, mentre un Test Set 5-wise (o superiore) non ha alcun senso.

# Wise crunch tools

Le tecniche di test combinatoriale di cui abbiamo discusso nei paragrafi precedenti sono rivolte a risolvere un problema di base, di cui abbiamo già discusso e che riformuliamo come segue:

*PROBLEMA DEL TEST COMBINATORIALE DIRETTO: “Dato un sistema software che accetta N variabili, ciascuna delle quali può assumere differenti valori, trovare il Test Set con il minore numero di casi di test possibile, che mi garantisca (almeno) una copertura di tutte le combinazioni (2-ple) di tutti i valori delle variabili in gioco”*.

Per risolvere tale problema è stata sviluppata la tecnica Pairwise ed un congruo numero di tools a supporto. Una volta che tale test set (il più possibile ridotto) è stato generato, si eseguono i test cases e si rilevano le (eventuali) anomalie del software sotto test.

Esiste anche un secondo problema, forse meno “gettonato” rispetto al precedente, che il seguente:

*PROBLEMA DEL TEST COMBINATORIALE INVERSO: “Dato un Test Set per il quale non si conosce il metodo di generazione, calcolare quale percentuale di copertura garantisce il Test Set rispetto al livello n-wise, con n compreso tra 1 ed il numero di variabili del Test Set”*.

Il tipico esempio è quello in cui i test sono generati da strumenti automatici sui quali si ha un controllo scarso o quasi nullo, oppure quando i “casi di test” sono generati da dei flussi automatici che alimentano interfacce tra differenti sistemi (si pensi a un sistema che passa dei dati contabili da un sistema A a un sistema B); nelle fasi di test tali dati vengono – in genere - “estratti” da serie storiche sulle quali non si ha alcun controllo.

Per scenari di test in qualche modo riconducibili a un problema combinatoriale inverso, non si sono trovati degli strumenti a supporto e/o tali strumenti non sono facilmente reperibili.

Nel seguito descriveremo una serie di tools denominati “Wise crunch tools” (eseguibili in ambiente Windows, ma – se necessario – facilmente portabili su piattaforme Unix/Linux) personalmente sviluppati che permettono di estrarre test set minimali e calcolare la copertura di un generico test set, utilizzando algoritmi di calcolo sistematico di copertura a partire da tutte le nple relative ai valori delle variabili; tali algoritmi ricadono nella categoria di “*algoritmi a forza bruta*” e come tali possono essere usati (su un normale PC di lavoro) se il numero di variabili e/o valori non è troppo elevato.

## Procedura di installazione

Il pacchetto “Wise crunch tools” è composto dai seguenti file e cartelle:

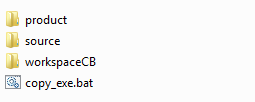


Figura 1 **– Elementi componenti i WiseCrunchTools – Struttura delle cartelle**

La cartella “***source***” contiene i sorgenti degli eseguibili in linguaggio C++ e non è d’interesse per l’utente finale dei tools.

La cartella “**workspaceCB**” contiene i file di workspace utilizzati dall’IDE OpenSource di sviluppo C/C++ denominata “Code::Blocks” (<http://www.codeblocks.org/>) utilizzata nella fase di sviluppo dei tools; anche questa cartella non è d’interesse per l’utente finale dei tools. Ovviamente, può essere utilizzato un qualunque ambiente di sviluppo C/C++ se si ha l’interesse di sviluppare/modificare i tools.

Il file “***copy\_exe.bat***” è una semplice utility di deploy dei file eseguibili, che copia gli eseguibili dalla cartella *source\<nome tool>\bin\Release* di ciascun tool nella cartella *\product\bin*. Anche in questo caso, tale file non è di interesse per l’utente finale dei tools.

Il contenuto della cartella “***product***” è il seguente:

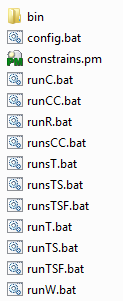


Figura 2 **– Elementi componenti i WiseCrunchTools – Struttura della cartella “product”**

Tale cartella contiene le utility principali **di primo livello** che saranno descritte nel dettaglio nei prossimi paragrafi.

Il contenuto della cartella “***bin***” è il seguente:

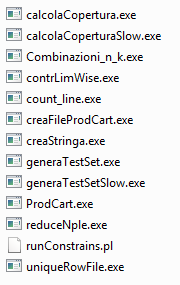


Figura 3 **– Elementi componenti i WiseCrunchTools – Struttura della cartella “bin”**

Tale cartella contiene le utility principali **di secondo livello** che saranno descritte nel dettaglio nei prossimi paragrafi.

La procedura di installazione si riduce ai seguenti passi.

1. Copiare il contenuto della cartella “***product***” sul file system della macchina. Supporremo nel seguito che tale cartella sia stata copiata sotto ***C:\WiseCrunchTools\product.***
2. Aggiungere alla variabile PATH di sistema e/o di utente i seguenti due percorsi:
   1. *C:\WiseCrunchTools\product*
   2. *C:\WiseCrunchTools\product\bin*

In Windows, è possibile compiere l’operazione da: *Pannello di Controllo →Sistema→Impostazioni Avanzate*, premere il pulsante “*Variabili d’ambiente*” e impostare quindi la variabile PATH di sistema o di utente, secondo i privilegi disponibili. Si faccia comunque riferimento al manuale di configurazione del vostro sistema operativo Windows per ogni evenienza.

Non ci sono vincoli particolari se si sceglie di installare i WiseCrunchTools in una cartella il cui nome contiene spazi, come ad esempio “*C:\My Documents\Wise Crunch Tools*”.

## Procedura di configurazione

La procedura di configurazione consiste nel modificare il file denominato “***config.bat***” presente nella cartella “*product*”. Il contenuto del file è il seguente (trascurando i commenti di intestazione) :

REM Nome del progetto in corso

set "PROJECT=###### <NOME PROGETTO> ######"

REM Directory di lavoro (deve terminare con \)

set "APP\_PATH=C:\Users\<utente>\Documents\<nome\_progetto>\"

REM File di input (deve essere presente nella directory di lavoro)

set "INPUT\_FILE\_NAME=input.txt"

REM Il separatore \*\*DEVE\*\* essere uguale al separatore utilizzato sul

REM file di input INPUT\_FILE\_NAME per separare i campi

set "SEP=;"

I campi da impostare sono:

* Campo ***PROJECT***: inserire il nome del progetto su cui andrà a lavorare. E’ una semplice stringa mnemonica che serve a ricordare sul quale progetto si sta operando e si può inserire in essa un qualunque valore.
* Campo ***APP\_PATH***: inserire una cartella valida del file system. Su tale cartella saranno salvati tutti i dati di base e comuni generati dai tools che si andranno a utilizzare. La cartella **deve obbligatoriamente terminare** con il carattere **“\”** (slash).
* Campo ***INPUT\_FILE\_NAME***: inserire il nome di un file formato testo valido, contenente le variabili e i relativi valori nel formato seguente:
  + <NOME VAR\_1 DI 1 CARATTERE>:VAL1\_1<SEP>VAL1\_2<SEP>VAL1\_3<SEP>…<SEP>VAL1\_N
  + <NOME VAR\_2 DI 1 CARATTERE>:VAL2\_1<SEP>VAL2\_2<SEP>VAL3<SEP>…<SEP>VAL2\_K
  + ….
  + <NOME VAR\_P DI 1 CARATTERE>:VALP\_1<SEP>VALP\_2<SEP>VALP<SEP>…<SEP>VALP\_T
* Campo ***SEP***: il separatore che ***deve obbligatoriamente coincidere*** con quello usato nel file d’input impostato nella variabile INPUT\_FILE\_NAME. **Il separatore non può essere uguale a “:” (due punti) oppure “\*” (asterisco) che sono caratteri riservati.**

Il nome della variabile nel file d’input **deve obbligatoriamente essere di un solo carattere**; la cosa migliore è utilizzare le lettere dell’alfabeto (maiuscole o minuscole e i numeri da 0 a 9).

Un esempio di file di input valido è il seguente:

**A:**A1;A2;A3;A4

**B:**B1;B2,B3

**C:**C1;C2;C3

I nomi delle variabili (A, B e C) sono separati dalla lista dei valori dal carattere **“:”** (due punti). Il separatore dei valori è **“;”** (punto e virgola) e deve coincidere con la variabile ***SEP*** impostata sul file “***config.bat***”.

## Panoramica generale dei WiseCrunchTools

I tools facenti parte del prodotto denominato WiseCrunchTools tendono a fornire un supporto alla soluzione di entrambi i problemi del test combinatoriale in precedenza enunciati.

I WiseCrunchTools **non intendono mettersi in competizione** con i tools già esistenti e tendenti a risolvere il ***problema diretto*** del test combinatoriale, quali ad esempio Microsoft PICT, AllPair di J. Bach o altri svariati tools commerciali e non presenti in commercio; i tools già presenti da lungo tempo sul mercato implementano algoritmi sicuramente più efficaci dei WiseCrunchTools e *vanno quindi preferiti* per quanto riguarda – ripeto – la soluzione del problema diretto.

Per quanto riguarda invece il ***problema inverso*** del test combinatoriale non esistono, a mia conoscenza, dei tool sul mercato; i WiseCrunchTools cercano quindi di fornire una prima soluzione al problema inverso, sicuramente migliorabile nel tempo, quando si saranno meglio comprese le logiche e le leggi che soggiacciono al test combinatorio e agli insiemi di Test minimali e massimali a esso correlati.

Ricordiamo che:

1. Risolvere il problema diretto significa determinare il test set più piccolo possibile con un livello di copertura WISE concordato (in genere WISE=2) a partire dall’insieme delle variabili e dei valori.
2. Risolvere il problema inverso significa determinare il livello di copertura di un test set rispetto a un dato livello WISE di riferimento (anche qui in genere WISE=2).

I tool si dividono in due categorie:

1. *Tools di primo livello*: scripts batch DOS che forniscono una immediata risposta ad una serie di scenari standard che si presentano tipicamente nei progetti di test che richiedono tecniche combinatoriali.
2. *Tools di secondo livello*: eseguibili (linguaggio di sviluppo C++/Perl) che possedendo una maggiore versatilità permettono di rispondere a dei problemi forse meno comuni ma che si presentano/possono presentare nei progetti di test che richiedono tecniche combinatoriali.

Gli script di primo livello sono stati pensati come dei “wrapper” costruiti intorno agli eseguibili di secondo livello, allo scopo di “semplificare la vita” all’utente finale e proponendo una serie di semplici comandi che permettono velocemente di ottenere una serie di “informazioni standard”.La tabella che segue mappa le due categorie di tools, dettagliando anche il tipo d’informazioni di risposta.

L’eseguibile di secondo livello denominato *uniqueRowFile.exe* non è richiamato dagli script di primo livello e va considerato un’utility particolare che non è utilizzata direttamente per la soluzione del problema diretto o inverso, ma che può rendersi utile in casi specifici avanzati, rimanendo quindi **solo** come utility di secondo livello.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Primo Livello | Secondo Livello | | | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| runW |  |  | X | X | X | X | X |  |  | X |  |  |  |
| runCC | X |  |  | X | X |  |  |  |  |  |  |  |  |
| runsCC |  | X |  | X | X |  |  |  |  |  |  |  |  |
| runT |  |  |  | X | X |  |  | X |  |  |  |  |  |
| runsT |  |  |  | X | X |  |  |  | X |  |  |  |  |
| runTS |  |  |  | X | X |  |  | X |  |  |  |  |  |
| runsTS |  |  |  | X | X |  |  |  | X |  |  |  |  |
| runTSF | X |  |  | X | X |  |  | X |  |  |  |  |  |
| runsTSF | X |  |  | X | X |  |  |  | X |  |  |  |  |
| runC |  |  |  |  |  |  |  |  |  |  |  | X |  |
| runR |  |  |  | X | X |  |  |  |  |  | X |  |  |

**Tabella 7 – Mappatura tools di primo livelli vs. secondo livello (wrapping map)**

A seguire l’elenco degli eseguibili di secondo livello:

1. calcolaCopertura.exe
2. calcolaCoperturaSlow.exe
3. Combinazioni\_n\_k.exe
4. contrLimWise.exe
5. count\_line.exe
6. creaFileProdCart.exe
7. creaStringa.exe
8. generaTestSet.exe
9. generaTestSetSlow.exe
10. ProdCart.exe
11. reduceNple.exe
12. runConstrains.pl
13. uniqueRowFile.exe

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tool | Livello | Scenario | | | |  |  | Categoria problema |
|  |  | 1 | 2 | 3 | 4 | 5 | 6 |
| runW | 1 | X |  |  |  |  |  | Inverso |
| runCC | 1 |  | X |  |  |  |  | Inverso |
| runsCC | 1 |  | X |  |  |  |  | Inverso |
| runT | 1 |  |  | X |  |  |  | Diretto |
| runsT | 1 |  |  | X |  |  |  | Diretto |
| runTS | 1 |  |  |  | X |  |  | Diretto |
| runsTS | 1 |  |  |  | X |  |  | Diretto |
| runTSF | 1 |  |  |  |  | X |  | Diretto |
| runsTSF | 1 |  |  |  |  | X |  | Diretto |
| runC | 1 |  |  |  |  |  | X | Diretto/Inverso |
| runR | 1 |  |  | X |  |  |  | Inverso |
| calcolaCopertura | 2 |  | X |  |  |  |  | Inverso |
| calcolaCoperturaSlow | 2 |  | X |  |  |  |  | Inverso |
| Combinazioni\_n\_k | 2 |  |  |  |  |  |  | / |
| contrLimWise | 2 |  |  |  |  |  |  | / |
| count\_line | 2 |  |  |  |  |  |  | / |
| creaFileProdCart | 2 |  |  |  |  |  |  | / |
| creaStringa | 2 |  |  |  |  |  |  | / |
| generaTestSet | 2 |  |  | X | X | X |  | Diretto |
| generaTestSetSlow | 2 |  |  | X | X | X |  | Diretto |
| ProdCart | 2 | X |  |  |  |  |  | Inverso |
| reduceNple | 2 |  |  | X |  |  |  | Diretto |
| runConstrains | 2 |  |  |  |  |  | X | Diretto/Inverso |
| uniqueRowFile | 2 |  |  |  |  |  |  | / |

**Tabella 8 – Mappatura tools con scenari standard**

A seguire l’elenco degli **scenari standard** presi in considerazione:

1. Calcolo insieme massimo (tutte le n-ple).
2. Calcolo della copertura di un Test Set rispetto a un livello wise prescelto.
3. Creazione di un Test Set minimale con copertura = un livello wise prescelto a partire dal Test Set di tutte le combinazioni.
4. Creazione di un Test Set minimale con copertura = un livello wise prescelto a partire da un test set qualunque.
5. Creazione di un Test Set minimale con copertura = un livello wise prescelto a partire da un test set qualunque ed escludendo un insieme di n-ple note.
6. Applicare delle condizioni di vincolo ai Test Set o alle n-ple.

La **Tabella 8** mostra come i tools messi a disposizione forniscano degli strumenti utili ad affrontare entrambe le tipologie di problemi; vogliamo comunque sottolineare che, essendo tali tools basati su algoritmi cosiddetti di “*brute force*”, se l’interesse principale è quello di avere velocemente un Test Set minimale a copertura wise predefinita da utilizzare per i test, è consigliabile usare gli strumenti ormai da lungo tempo presenti sul mercato.

## Convenzioni lessicali e tipografiche

Nel seguito, il numero di variabili del file d’input coincidente con il massimo valore possibile della copertura wise, verrà indicato con WISE\_MAX.

Faremo molte volte riferimento ad un Test Set Minimo, considerandolo come insieme contenente il minimo numero di test set possibile. Questa è un’imprecisione poiché non esiste a oggi alcun algoritmo che garantisce che il Test Set estratto sia il minimo in assoluto. Il termine corretto sarebbe quindi quasi-Minimo; quindi il termine “minimo” dovrà essere inteso in quest’accezione.

Tutti i tools descritti nel seguito hanno un help a consolle cui si accede passando l’opzione **/?** come consueto per i comandi DOS. Tale opzione, essendo comune a tutti i tools, non viene riportata nella descrizione delle specifiche del singolo strumento.

I parametri compresi tra parentesi graffe **{…}** in **grassetto** sono OBBLIGATORI, mentre quelli compresi tra parentesi tonde *(…)* in *italico* sono *FACOLTATIVI*.

## Tool di primo livello – script batch DOS

Di seguito sono descritti i singoli tools di primo livello, più semplici da usare, che tendono a supportare i “scenari standard” precedentemente descritti.

### Come convenzione, il nome di tutti gli script batch segue lo schema:

* Prefisso **run** comune a tutti
* Lettera **s** solo per gli script cosiddetti “slow”, che ottimizzano l’utilizzo della memoria a scapito del tempo di elaborazione.
* Numero di lettere **XX…X** maiuscole pari al numero di parametri che lo script richiede

Ad esempio *runCC* è uno script che richiede due parametri d’input, mentre *runW* ne richiede uno solo. Lo script *runsCC* richiede due parametri in input ed è in genere più lento di *runCC*, ma richiede meno memoria di sistema.

### Tool runW

E’ il primo tool che è obbligatorio usare per poter poi utilizzare gli altri tools. Crea tutte le n-ple corrispondenti al valore del **W**ise passato in input (runW = run **W**ise)

#### Input:

1. **{WISE}**: Numero intero compreso tra 1 ed il WISE\_MAX

#### Output:

1. File contenente tutte le n-ple corrispondenti al {WISE} in input, denominato ***out\_p\_{WISE}.txt*** nella cartella APP\_PATH.
2. File con tutte le possibili combinazioni delle variabili di input, denominati **out\_c\_{WISE}.txt** contenuto nella cartella APP\_PATH.

#### Esempio:

runW 2

Crea tutte le coppie di valori delle variabili del file di input.

### Tool runCC e runsCC

Esegue il **C**alcolo della **C**opertura rispetto al WISE passato in input del Test Set passato anch’esso in input (runCC = run **C**alcola **C**opertura).

#### Input:

1. **{FILE TESTSET}:** Il Test Set per il quale si vuole calcolare la copertura
2. **{WISE NUMBER}:** Il valore del WISE rispetto al quale la copertura deve essere calcolata
3. *(-n)* Stampa le n-ple trovate nel file {FILE TESTSET}
4. *(-d)* Stampa i dettagli, consistenti nella coppia #riga Test Set, #riga N-pla
5. *(-f)* Stampa la mappa di frequenza: #riga N-pla, #occorrenze n-pla trovate
6. *(-r)* Stampa la statistica (% di copertura trovata)
7. *(-z)* Stampa il record del Test Set con il numero di n-ple associate

#### Output:

1. Le informazioni come descritto dalle opzioni *(-n) , (-d), (-f), (-r)* e *(-z)*. Si noti che se non si passa almeno un’opzione non viene stampato nulla. L’output è stampato a video e può essere reindirizzato su file con le consuete operazioni di “pipe” del DOS (> oppure >>).

#### Prerequisiti

1. Occorre che sia stato in precedenza eseguito il comando *runW {WISE NUMBER}*. In caso contrario viene stampato un avviso con la richiesta di eseguire tale comando.

Il tool ***runCC*** è più veloce ma richiede una memoria di sistema maggiore. Nel caso in cui la memoria di sistema non sia sufficiente, il programma s’interrompe con un overflow di memoria gestito. Il tool ***runsCC*** (si noti la “s” in run**s** che sta per “slow”) richiede meno memoria, ma impiega più tempo, lavorando principalmente su file.

#### Esempio:

runCC test\_set.txt 2 –r -f

Stampa la statistica e la mappa di frequenza per il test set contenuto nel file “test\_set.txt”.

### Tools runT e runsT

Estrae il Test Set minimale con copertura garantita pari al **W**ise passato in input (runT = run **T**est) a partire dal file delle n-ple relative al WISE\_MAX (tutte le combinazioni).

#### Input:

1. **{WISE}**: Numero intero compreso tra 1 ed il WISE\_MAX

#### Output:

1. File contenente il test set minimale con copertura garantita al {WISE} passato in input, denominato ***test\_set\_generato\_{WISE\_MAX}\_{WISE}.txt*** nella cartella di esecuzione.
2. File contenente il test set minimale con copertura garantita al {WISE} passato in input, denominato **clean\_*test\_set\_generato\_{WISE\_MAX}\_{WISE}.txt*** nella cartella di esecuzione.

La differenza tra i due file è che in ***test\_set\_generato\_{WISE\_MAX}\_{WISE}.txt*** viene riportato il numero di test relativo al file delle n-ple {WISE\_MAX}, contenente tutte le combinazioni possibili, mentre in **clean\_*test\_set\_generato\_{WISE\_MAX}\_{WISE}.txt*** tale valore non viene riportato.

#### Prerequisiti

1. Occorre che sia stato in precedenza eseguito il comando *runW {WISE\_MAX}*. In caso contrario è stampato un avviso con la richiesta di eseguire tale comando.

#### Esempio:

runT 2

Crea il test set a copertura wise-2 a partire dal file contenente tutte le combinazioni possibili.

Notare che, se {WISE}≡{WISE MAX} allora è semplicemente copiato il file out\_p\_{WISE\_MAX} nella cartella di esecuzione creando ***solo*** il file **clean\_*test\_set\_generato\_{WISE\_MAX}\_{WISE\_MAX}.txt.*** In questo caso viene anche stampato un errore DOS “*File Not Found*” o “*File non trovato*” causato dal fatto che il file ***test\_set\_generato\_{WISE\_MAX}\_{WISE\_MAX}.txt*** non viene prodotto; la cosa è comunque voluta e come tale corretta.

Il tool ***runT*** è più veloce ma richiede una memoria di sistema maggiore. Nel caso in cui la memoria di sistema non sia sufficiente, il programma s’interrompe con un overflow di memoria gestito. Il tool ***runsT*** (si noti la “s” in run**s** che sta per “slow”) richiede meno memoria, ma impiega più tempo, lavorando principalmente su file.

### Tools runTS e runsTS

Estrae il Test Set minimale con copertura garantita pari al **W**ise passato in input a partire dal file delle n-ple passato in input (runTS = run **T**est **S**et).

#### Input:

1. **{FILE TESTSET BASE}** Test Set di partenza da cui estrarre il Test Set di output
2. **{WISE}**: Numero intero compreso tra 1 ed il WISE\_MAX

#### Output:

1. File contenente il test set minimale con copertura garantita al {WISE} passato in input o alla stessa copertura del file {FILE TESTSET BASE} se minore di {WISE}, denominato ***test\_set\_generato\_{WISE}.txt*** nella cartella di esecuzione.
2. File contenente il test set minimale con copertura garantita al {WISE} passato in input o alla stessa copertura del file {FILE TESTSET BASE} se minore di {WISE}, denominato **clean\_*test\_set\_generato\_{WISE}.txt*** nella cartella di esecuzione.

La differenza tra i due file è che in ***test\_set\_generato\_{WISE}.txt*** viene riportato il numero di test relativo al file delle n-ple {FILE TESTSET BASE} passato in input, mentre in **clean\_*test\_set\_generato\_{WISE}.txt*** tale valore non viene riportato.

#### Prerequisiti

1. Occorre che sia stato in precedenza eseguito il comando *runW {WISE}*. In caso contrario è stampato un avviso con la richiesta di eseguire tale comando.

#### Esempio:

runTS base\_testset.txt 2

Crea il test set a copertura wise-2 (o copertura pari a quella di base\_testset.txt se minore di 2) a partire dal file base\_testset.txt passato in input.

Notare che se il file {FILE TESTSET BASE} passato in input non garantisce una copertura al livello {WISE} passato in input, i file di output prodotti coincideranno con il file di input ed avranno – ovviamente – la stesa percentuale di copertura del file {FILE TESTSET BASE} di input.

Il tool ***runTS*** è più veloce ma richiede una memoria di sistema maggiore. Nel caso in cui la memoria di sistema non sia sufficiente, il programma s’interrompe con un overflow di memoria gestito. Il tool ***runsTS*** (si noti la “s” in run**s** che sta per “slow”) richiede meno memoria, ma impiega più tempo, lavorando principalmente su file.

### Tools runTSF e runsTSF

Estrae il Test Set minimale con copertura garantita pari al **W**ise passato in input a partire dal file delle n-ple passato in input escludendo le n-ple già coperte dal file contenente il Test Set Parziale passato anch’esso in input (runTSF = run **T**est **S**et **F**orbidden).

#### Input:

1. **{FILE TESTSET BASE}** Test Set di partenza da cui estrarre il Test Set di output
2. **{FILE TESTSET PARZIALE}** Test Set Parziale con casi di test validi, da escludere dall’output da generare.
3. **{WISE}**: Numero intero compreso tra 1 ed il WISE\_MAX

#### Output:

1. File contenente il test set minimale con copertura garantita al {WISE} passato in input o alla stessa copertura del file {FILE TESTSET BASE} se minore di {WISE}, denominato **delta\_*test\_set\_generato\_{WISE}.txt*** nella cartella di esecuzione.
2. File contenente il test set minimale con copertura garantita al {WISE} passato in input o alla stessa copertura del file {FILE TESTSET BASE} se minore di {WISE}, denominato **delta**\_**clean\_*test\_set\_generato\_{WISE}.txt*** nella cartella di esecuzione.

La differenza tra i due file è che in ***test\_set\_generato\_{WISE}.txt*** viene riportato il numero di test relativo al file delle n-ple {FILE TESTSET BASE} passato in input, mentre in **clean\_*test\_set\_generato\_{WISE}.txt*** tale valore non viene riportato. Si noti che il file generato è solo il **delta** dei casi di test rispetto a quelli contenuti in {FILE TESTSET PARZIALE}; quindi il file generato non può garantire una copertura al WISE passato in input. Tale copertura è garantita solo dal file “somma” di delta\_\*\_*test\_set\_generato\_{WISE}.txt*con {FILE TESTSET PARZIALE}**.**

#### Prerequisiti

1. Occorre che sia stato in precedenza eseguito il comando *runW {WISE}*. In caso contrario è stampato un avviso con la richiesta di eseguire tale comando.

#### Esempio:

runTSF base\_testset.txt parziale.txt 2

Crea il test set a copertura wise-2 (o copertura pari a quella di base\_testset.txt se minore di 2) a partire dal file base\_testset.txt passato in input, senza considerare le n-ple già comprese nel file del Test Set parziale.txt.

Notare che se il file {FILE TESTSET BASE} passato in input non garantisce una copertura al livello {WISE} passato in input, i file di output prodotti coincideranno con il file di input ed avranno – ovviamente – la stesa percentuale di copertura del file {FILE TESTSET BASE} di input.

Il tool ***runTSF*** è più veloce ma richiede una memoria di sistema maggiore. Nel caso in cui la memoria di sistema non sia sufficiente, il programma s’interrompe con un overflow di memoria gestito. Il tool ***runsTSF*** (si noti la “s” in run**s** che sta per “slow”) richiede meno memoria, ma impiega più tempo, lavorando principalmente su file.

### Tool runR

Estrae il Test Set **non minimale** ma comunque ridotto rispetto al Test Set Massimo con copertura garantita pari al Wise passato in input. (runR = run **R**educe).

#### Input:

1. **{WISE}**: Numero intero compreso tra 1 ed il WISE\_MAX

#### Output:

1. Test Set contenente un insieme ridotto ma non minimale con copertura pari al WISE passato in input. I valori con \* rappresentano i *“dont care”*; al posto di “\*” è possibile prendere un qualunque valore della variabile, senza con questo inficiare il livello di copertura al WISE. L’output è stampato a video e può essere reindirizzato su file con le consuete operazioni di “pipe” del DOS (> oppure >>)

Il numero di test cases del test set generato dipende dall’ordinamento del file out\_p\_{WISE}.txt che viene passato all’eseguibile richiamato dal file batch *runR.bat*. Sicuramente esiste un ordinamento del file per cui il test set di output contiene un numero minimo di Test Cases con copertura garantita a WISE, ma trovare tale ordinamento non è fattibile da un punto di vista computazionale, poiché troppo oneroso.

#### Prerequisiti

1. Occorre che sia stato in precedenza eseguito il comando *runW {WISE}*. In caso contrario è stampato un avviso con la richiesta di eseguire tale comando.

#### Esempio:

runR 2

Crea il test set a copertura wise-2 non ridotto al minimo, ma con un numero di Test Cases inferiore rispetto al Test Set Massimo.

### Tool runC

Applica le constrains al file di tipo test set o di tipo n-ple passato in input.

#### Input:

1. **{FILE }:** Il Test Set o file delle n-ple al quale si vogliono applicare i vincoli programmati nel file *constrains.pm* (linguaggio Perl)

#### Output:

1. Sullo standard output vengono stampate le nple o i casi di test validi, mentre sullo standard error sono stampate le nple o i casi di test non validi L’output è stampato a video e sullo standar error (in genere coincidente con lo standard output) e può essere reindirizzato su file con le consuete operazioni di “pipe” del DOS (> oppure >>).

#### Prerequisiti

1. Nessuno; Il tool non è vincolato agli altri tools direttamente. Può essere usato in quei casi in cui il processo di generazione di un test set minimale mediante i tools *runT, runTS o runTSF* (e relativi equivalenti slow) richiederebbe troppo tempo/risorse.

#### Esempio:

runC test\_set.txt 1>validi.txt 2>non\_validi.txt

Applica i vincoli programmati nel file *constrains.pm* (linguaggio Perl) e stampa i record validi nel file “*validi.txt*” (record che ***rispettano*** i vincoli) e i record non validi nel file “*non\_validi.txt*” (record che ***non rispettano*** i vincoli). Si noti l’utilizzo dei “pipe” (>) che reindirizzano lo standard output (*1>valid.txt*) e lo standard error (*2>non\_validi.txt*).

Il file “*constrains.pm”* contenente i filtri è un modulo scritto in linguaggio Perl, all’interno del quale è possibile scrivere delle condizioni che permettono di accettare o scartare i record del file passato in input. La variabile da prendere in considerazione è il vettore **$r** che accede ai valori delle variabili del record letto. In pratica, per ogni riga letta, viene eseguita la seguente associazione:

$r->[0]: valore corrente sulla riga letta della prima variabile (A)

$r->[1]: valore corrente sulla riga letta della seconda variabile (B)

…

$r->[K]: valore corrente sulla riga letta della (K-1) variabile (J)

Ogni controllo deve restituire un valore che se vale:

* 1: la riga considerata **non e’ valida**
* 0: la riga considerata **è valida**

Ad esempio

if($r->[0] eq "A1" && $r->[1] eq "B3"){

return 1;

};

Ciascuna riga del file d’input è letta e la condizione è verificata: se il valore della prima variabile (A) è “A1” e il valore della seconda variabile (B ) è “B3” la riga è scartata. Nella presente release del prodotto non è stato implementato un meccanismo per variare i valori delle variabili della riga del file; in altri termini, una condizione del tipo:

if($r->[0] eq "A1" && $r->[1] eq "B3"){

$r->[1]="B2"

return 0;

};

**non farebbe variare** il valore della seconda variabile da “B3” a “B2”; quindi, ritornando 0, la riga che dovrebbe essere esclusa, sarebbe invece – erroneamente – presa per valida.

In pratica, il filtro è solo in validazione o meno dei valori presenti, ma non ne modifica il contenuto.

## Tool di secondo livello – eseguibili

Di seguito sono descritti i singoli tools di secondo livello, meno semplici da usare ma più versatili, che oltre a supportare gli scenari “standard” in precedenza descritti, possono essere utili all’utente esperto per scenari più generali e complessi.

### Eseguibili calcolacopertura.exe e calcolaCoperturaSlow.exe

Esegue il **C**alcolo della **C**opertura rispetto al WISE passato in input del Test Set anch’esso dato in input.

#### Input:

1. –**w: {WISE FILE NAME}** Nome del file con tutte le {WISE}-ple , il file out\_p\_{WISE}.txt
2. **–t: {FILE TESTSET}:** Il Test Set per il quale si vuole calcolare la copertura
3. **–o: {WISE NUMBER}:** Il valore del WISE usato per il calcolo della copertura
4. *(-s)* Separatore dei campi del file
5. *(-n)* Stampa le n-ple trovate nel file {FILE TESTSET}
6. *(-d)* Stampa i dettagli, consistenti nella coppia #riga Test Set, #riga N-pla
7. *(-f)* Stampa la mappa di frequenza: #riga N-pla, #occorrenze n-pla trovate
8. *(-r)* Stampa la statistica (% di copertura trovata)
9. *(-z)* Stampa il record del Test Set con il numero di n-ple associate

#### Output:

1. Le informazioni come descritto dalle opzioni *(-n) , (-d), (-f), (-r)* e *(-z)*. Si noti che se non si passa almeno un’opzione non viene stampato nulla. L’output è stampato a video e può essere reindirizzato su file con le consuete operazioni di “pipe” del DOS (> oppure >>).

Il programma ***calcolaCopertura*** più veloce ma richiede una memoria di sistema maggiore. Nel caso in cui la memoria di sistema non sia sufficiente, il programma s’interrompe con un overflow di memoria gestito. Il programma ***calcolaCoperturaSlow*** richiede meno memoria, ma impiega più tempo, lavorando principalmente su file.

#### Esempio:

calcolaCopertura –w: out\_p\_2.txt –t: test\_set.txt –o: 2 –r -f

Stampa la statistica e la mappa di frequenza per il test set contenuto nel file “test\_set.txt” rispetto al livello WISE=2.

### Eseguibile Combinazioni\_n\_k

Estrae tutte le combinazioni K a K di una stringa di lunghezza N passata in input.

#### Input:

1. –**s: {STRINGA}** Stringa in input con caratteri distinti
2. **–k: {k}:** Valore della lunghezza degli spezzoni della stringa (combinazioni K a K)

#### Output:

1. Tutte le combinazioni della stringa K a K L’output è stampato a video e può essere reindirizzato su file con le consuete operazioni di “pipe” del DOS (> oppure >>).

#### Esempio:

Combinazioni\_n\_k –s: ABCDE –k: 2

Restituisce la lista:

A B

A C

A D

B C

B D

C D

che sono le combinazioni della stringa ABCDE prese a 2 a 2. Si noti che se la stringa contiene caratteri duplicati (ad esempio ABCBCD, che contiene il carattere “B” 2 volte), il programma restituisce un errore.

### Eseguibili generaTestSet.exe e generaTestSetSlow.exe

Crea il test set con copertura pari al WISE passato in input o pari a quella del Test Set passato anch’esso in input, se minore di WISE. Il file generato esclude le n-ple contenute nel file di input {FROM FILE}, se presente.

#### Input:

1. –**w: {WISE FILE NAME}** Nome del file con tutte le {WISE}-ple , il file *out\_p\_{WISE}.txt*
2. **–t: {FILE TESTSET}:** Il Test Set per il quale si vuole calcolare la copertura.
3. **–o: {WISE NUMBER}:** Il valore del WISE usato per il calcolo della copertura.
4. *(-s)* Separatore dei campi del file.
5. *-f:(FROM FILE)*  Nome del file contenente la lista delle n-ple da non tener conto nella generazione del file di output.

#### Output:

1. Test set con la copertura richiesta come da parametri in input. L’output è stampato a video e può essere reindirizzato su file con le consuete operazioni di “pipe” del DOS (> oppure >>).

Il programma ***generaTestSet*** più veloce ma richiede una memoria di sistema maggiore. Nel caso in cui la memoria di sistema non sia sufficiente, il programma s’interrompe con un overflow di memoria gestito. Il programma ***generaTestSetSlow*** richiede meno memoria, ma impiega più tempo, lavorando principalmente su file.

#### Esempio:

generaTestSet –w: out\_p\_2.txt –t: test\_set.txt –o: 2 –f: nple.txt

Genera un Test Set dal file test\_set.txt con copertura pari a 2 o alla copertura del file test\_set.txt se minor di 2, considerando le n-ple contenute nel file out\_p\_2.txt dal quale esclude le n-ple contenute nel file nple.txt. Quest’ultimo file è una lista della forma:

12

14

15

26

23

…

dove i numeri che compaiono sono i numeri di riga del file out\_p\_2.txt contenente tutte le n-ple a copertura wise=2; le n-ple contenute nel file nple.txt verranno escluse durante la generazione del test set di output. Il file con le n-ple da escludere può essere generato con l’eseguibile *calcolaCopertura.exe* (o *calcolaCoperturaSlow*.exe) con la sola opzione (-n) applicato al test set dal quale di vogliono “estrarre” le n-ple coperte.

### Eseguibile ProdCart.exe

Genera tutte le possibili combinazioni .dei valori delle variabili definite nel file d’input

#### Input:

1. –**i: {INPUT FILE NAME}** Nome del file di input in cui ciascuna riga contiene tutti i valori di una variabile separate da (SEP)
2. *–s: (SEP)***:** Separatore dei valori delle variabili. Il default è “;”.
3. *(-c)* Stampa dei commenti in fase di generazione dell’output.

#### Output:

1. File contenente tutte le combinazioni possibili delle variabili di input. L’output è stampato a video e può essere reindirizzato su file con le consuete operazioni di “pipe” del DOS (> oppure >>).

Il programma ***ProdCart*** è generalmente molto più veloce dei precedenti, ma nel caso in cui la memoria di sistema non sia sufficiente (ossia, quando il numero di combinazioni sia molto elevato), esce con un messaggio di overflow di memoria gestito.

#### Esempio:

ProdCart –i: input.txt

Genera un file con tutte le combinazioni possibili delle variabili. Se, ad esempio, il file input.txt è il seguente:

A1;A2;A3;A4

B1;B2;B3

C1;C2

L’output comprendente tutte le combinazioni sarà:

A1;B1;C1

A1;B1;C2

A1;B2;C1

A1;B2;C2

A1;B3;C1

A1;B3;C2

A2;B1;C1

A2;B1;C2

A2;B2;C1

A2;B2;C2

A2;B3;C1

A2;B3;C2

A3;B1;C1

A3;B1;C2

A3;B2;C1

A3;B2;C2

A3;B3;C1

A3;B3;C2

A4;B1;C1

A4;B1;C2

A4;B2;C1

A4;B2;C2

A4;B3;C1

A4;B3;C2

ossia, tutte le combinazioni dei valori di tutte le tre variabili A,B e C, una per ciascuna riga del file di input. Si noti che le righe del file di input di *ProdCart* **non sono** etichettate come per il caso del file di input del tool batch *runW* (si veda il paragrafo “Procedura di configurazione”). In quel caso il file d’input avrebbe dovuto riportare in testa a ciascuna riga il nome della variabile (composto di una unica lettera) seguito da “:”, ossia:

**A:**A1;A2;A3;A4

**B:**B1;B2;B3

**C:**C1;C2

Tale formato di file **non è corretto** per l’eseguibile *ProdCart*.

### Eseguibile reduceNple.exe

Compatta quanto più possibile le n-ple contenute nel file di input, sostituendo ai valori “\*” dei valori specifici delle variabili e creando così un Test Set a partire dal file delle n-ple che, pur non essendo il Test Set minimo, è ridotto rispetto al Test Set Massimo (coincidente con tutte le n-ple). Il numero di record prodotti dipende, in modo non noto, dall’ordinamento del file delle n-ple in input. Sicuramente esiste un ordinamento del file per cui il test set di output contiene un numero minimo di Test Cases con copertura garantita a WISE, ma trovare tale ordinamento non è fattibile da un punto di vista computazionale, poiché troppo oneroso.

#### Input:

1. **–n: {NPLE FILE}**: File con le n-ple da ridurre
2. *–s: (SEP):*Separatore dei valori delle variabili. Il default è “;”.

#### Output:

1. Test Set contenente un insieme ridotto ma non minimale con copertura pari al WISE delle n-ple passato in input. I rimanenti valori con \* rappresentano i *“dont care”*; al posto di “\*” è possibile prendere un qualunque valore della variabile, senza con questo inficiare il livello di copertura al WISE. L’output è stampato a video e può essere reindirizzato su file con le consuete operazioni di “pipe” del DOS (> oppure >>).

#### Esempio:

reduceNple –n: out\_p\_1.txt

Supponendo che il file out\_p\_1.txt sia il seguente e che garantisca una copertura 1 dei valori delle variabili A, B e C:

A1;\*;\*

A2;\*;\*

A3;\*;\*

A4;\*;\*

\*;B1;\*

\*;B2;\*

\*;B3;\*

\*;\*;C1

\*;\*;C2

Il test set ridotto è il seguente:

A1;B1;C1

A2;B2;C2

A3;B3;\*

A4;\*;\*

Che, in questo caso l’output prodotto coincide anche con il test minimo, anche se nel caso generale con wise > 1 non è vero.

### Eseguibili di utility

Gli eseguibili descritti molto brevemente di seguito non forniscono supporto diretto alla generazione e/o alla gestione dei test set, ma sono utilizzati prevalentemente dai tools batch DOS per eseguire operazioni secondarie che è impossibile o – quantomeno – molto complesso eseguire direttamente in DOS. Tali utilities possono anche essere di una qualche utilità operativa, anche se non sono da considerarsi “tout cours” dei tool di test. Li documentiamo di seguito brevemente per completezza, ricordando che lanciando semplicemente il comando senza passare parametri, viene stampato (per le utility più significative) un help riguardante la modalità di utilizzo.

* ***contrLimWise.exe***: effettua il controllo dei limiti minimo e massimo dei WISE all’interno dei tools batch. Se lanciato, restituisce **“ko”** e – preso a se stante – non è di alcuna utilità pratica e non restituisce un help relativo alla modalità di utilizzo
* **count\_line.exe**: effettua il conteggio delle linee del file passato in input e lo stampa a video; se il file input.txt ha 10 linee, allora count\_line input.txt restituisce 10. Se lanciato senza parametri in input restituisce un help relativo alla modalità di utilizzo.
* **creaFileProdCart.exe**: crea il file nel formato accettabile da ProdCart.exe, con le sole variabili sulle quale devono essere calcolate le combinazioni e rimpiazzando gli altri valori con “\*”. Preso a se stante, non è di alcuna utilità pratica e lanciato senza parametri restituisce un help relativo alla modalità di utilizzo.
* **creaStringa.exe**: crea una stringa contenente i nomi (singoli caratteri) delle variabili del file di input descritto nel paragrafo “Procedura di configurazione”. Preso a se stante, non è di alcuna utilità pratica e lanciato senza parametri restituisce un help relativo alla modalità di utilizzo.
* **runConstrains.pl**: non è un eseguibile propriamente detto, ma uno script Perl utilizzato dal tool *runC* per applicare i vincoli ai Test Set o ai file di N-ple. *Da non usare direttamente, a meno che non si sia ben consci di quello che si sta facendo*.
* **uniqueRowFile.exe**: non considera le linee duplicate del file passato in input e stampa a video quelle univoche. Lanciato senza parametri, restituisce un help sulla modalità di utilizzo e può essere di utilità pratica se si hanno dei file con righe duplicate da “ripulire”. Questo eseguibile non è utilizzato all’interno degli script di primo livello e va considerato un’utility particolare che non è utilizzata direttamente per la soluzione del problema diretto o inverso, ma che può rendersi utile in casi specifici, rimanendo così **solo** come utility di secondo livello.

# Evoluzione del documento

| **Versione** | **Descrizione** |
| --- | --- |
| **01** | Prima versione rilasciata il 19 febbraio 2014 |