Correttezza

Correttezza Parziale: Legato ad una procedura, ovvero l'algoritmo termina solo con determinati ingressi
Correttezza Totale: Legato ad un algoritmo, ovvero l'algoritmo termina con ogni possibile ingresso

Specifiche di un algoritmo

Pre-condizioni: ipotesi sull'ingresso(come deve essere fatto l'*input*)
Post-condizioni:proprietà dell'uscita(criterio che stablisce come deve essere fatto l'output).
Esempio divisione intera:
Div(a,b)
Pre: $a \geq 0, b > 0$ numeri interi
Post: q e r tali che $a = b \cdot q + r$ e $0 \leq r < b$

Induzione Completa/Semplice

Semplice: Si induce incrementando/diminuendo un parametro di 1 in 1
$P(m) => P(m+1)$
Esempio: per la **Torre di Hanoi**, se ho n dischi richiamo l'algoritmo con $n-$ dischi.
dunque il parametro n viene diminuito di 1 ad ogni iterazione

Completa: Si induce incrementando/diminuendo un parametro di un valore diverso da 1.
Esempio: per la **Divisione Ricorsiva**, la soluzione con input (a, b) si trova richiamando l'algoritmo con input $(a-b, b)$, dunque il parametro a viene diminuito di b ad ogni iterazione

Induzione Completa

Caso Base: $\exists k \geq 0 . P(0) \wedge P(1) \wedge ... \wedge P(K)$
Ovvero, dato un $k \geq 0$, $P(n)$ vale fino a $n = k$

Passo induttivo: $\forall m \geq k.\, P(0) \wedge P(1) \wedge \ldots \wedge P(m) => P(m+1)$
Ovvero, per ogni $m \geq k$, vale $P(m)$
Conclusione: caso base e passo induttivo implicano $\forall n \geq 0. P(n)$

Invariante di ciclo

Invariante: Proposizione che esprime una proprietà delle variabili che persiste in ogni punto dell'algoritmo.

- **inizializzazione:** la proposizione vale subito prima del ciclo
- **mantenimento:** se la proposizione vale prima di entrare nel ciclo, vale anche dopo che il corpo del ciclo è stato eseguito.

L'invariante deve essere **utile** a verificare la correttezza dell'algoritmo.

Nel caso della **divisione iterativa**:

$\textsc{Div-It}(a, b)$

$\quad\quad \triangleright$ Pre: $a \geq 0, b > 0$

$\quad\quad \triangleright$ Post: ritorna q, r tali che $a = b\,q + r \wedge 0 \leq r < b$

$r \leftarrow a$

$q \leftarrow 0$

while $r \geq b$ **do**

$\quad r \leftarrow r - b$

$\quad q \leftarrow q + 1$

end while

return q, r

una *invariante di ciclo* potrebbe essere $a = bq + r$ con $0 \leq r$

Questa proposizione vale prima del ciclo(**inizializzazione**):
$q = 0$ dunque bq vale 0 mentre $r = a$ dunque $bq + r => 0 + a$

Vale anche durante e dopo il ciclo(**mantenimento**):

$$a = bq + r = b(q+1) + (r-b) = bq' + r' \wedge 0 \leq r' = r - b$$

Alla fine del ciclo sappiamo che:

$a = bq + r \wedge 0 \leq r$

ma visto che siamo usciti dal ciclo sappiamo anche che $0 \leq r < b$, dal momento che quella è la condizione di terminazione del ciclo.

Possiamo dunque constatare che l'algoritmo è **corretto** dal momento che soddisfa le post condizioni(**criterio dell'output**)

$\triangleright$ Post: ritorna q, r tali che $a = b\,q + r \wedge 0 \leq r < b$