Algoritmo

Un algoritmo è una strategia che serve per risolvere un problema ed è costituito da una sequenza finita di operazioni (dette anche istruzioni) e rispetti le seguenti caratteristiche

- ✓ sia eseguibile
- ✓ sia priva di ambiguità
- ✓ arrivi ad una conclusione in un tempo finito

Complessità algoritmo

La valutazione della complessità computazionale deve soddisfare due requisiti:

1. Non deve dipendere da una particolare macchina o da un particolare compilatore: il tempo di esecuzione di ogni istruzione in un qualunque linguaggio non è uguale per tutti i calcolatori ma varia (es: dipendenza della frequenza di lavoro della cpu)
2. Deve essere espressa in funzione dei dati in ingresso: un algoritmo elabora dei dati in ingresso per fornire altri dati che rappresentano la soluzione del problema, il tempo di esecuzione dell'algoritmo varia al variare dei dati in ingresso.
   - Tempo di esecuzione → relativo sia alla dimensione dei dati in ingresso che al loro valore.
   - Dati di ingresso → La complessità di un algoritmo dipende sempre dal numero dei dati che devono essere elaborati dall'algoritmo (Dimensione dei dati in ingresso=n).
   - Valori assunti dai dati in ingresso → In molti casi la complessità computazionale di un algoritmo dipende anche da questi.

Al fine di tener conto dell'influenza dei valori assunti dai dati in ingresso sulla complessità computazionale, in genere vengono considerati 3 possibili scenari: un tempo di calcolo massimo, minimo e uno medio:

Massimo: T(n) viene fissato considerando i valori dei dati in ingresso che comportano il massimo tempo di esecuzione.

Minimo: T(n) viene fissato considerando i valori dei dati in ingresso che comportano il minimo tempo di esecuzione.

Medio: il tempo medio di esecuzione su dati di dimensione n, ottenuto considerando tutti i possibili valori dei dati in ingresso, per ciascuno di essi determinando la complessità computazionale e, infine, eseguendo la media dei valori di complessità computazionale ottenuti.

Complessità asintotica

Notazione O → fornisce una delimitazione superiore al tempo di esecuzione di un algoritmo, cioè fornisce una valutazione approssimata per eccesso.

Notazione ohm → fornisce una delimitazione inferiore al costo di esecuzione di un algoritmo.

Classi di complessità → grazie all'analisi della complessità asintotica possiamo categorizzare gli algoritmi nelle seguenti classi di complessità:

- ✓ costante O(1)
- ✓ logaritmica O (logn)
- ✓ lineare O (n)
- ✓ nlog O (nlogn)
- ✓ quadratica O (n2)
- ✓ cubica O (n3)
- ✓ esponenziale O (an) a > 1

Ordine di Efficienza: O (1) < O (logn) < O (n) < O (nlogn) < O (n2) < O (n3) < O (an)

Delimitazioni superiori e inferiori per gli algoritmi

● Un algoritmo ha complessità O(f(n)) se è Tworst(n) = O(f(n)).
In tal caso f(n) è infatti una delimitazione superiore del tempo di calcolo per qualsiasi input: si ha la garanzia che al crescere di n il tempo di calcolo non cresce di più di f(n), qualunque sia l'input.

● Un algoritmo ha complessità Ω(f(n)) se è Tbest(n) = Ω(f(n)).

In tal caso f(n) è infatti una delimitazione inferiore del tempo di calcolo per qualsiasi input: si ha purtroppo la certezza che al crescere di n il tempo di calcolo cresce almeno come f(n), qualunque sia l'input.

● Un algoritmo ha complessità Θ(f(n)) se ha complessità O(f(n)) e Ω(f(n)).

In tal caso f(n) è infatti una delimitazione sia inferiore che superiore del tempo di calcolo per qualsiasi input.

Algebra degli "O" grandi

● Sia g1(n) la complessità del primo blocco e g2(n) la complessità del secondo blocco. La complessità globale è: O(g1(n) + g2(n)) = O(max{g1(n), g2(n)})
La complessità di un blocco costituito da più blocchi in sequenza è quella del blocco di complessità maggiore

● Sia g1(n) la complessità del blocco esterno e g2(n) la complessità di quello interno. La complessità globale è: O(g1(n) * g2(n)) = O(g1(n)) * O(g2(n))

La complessità di un blocco costituito da più blocchi annidati è data dal prodotto delle complessità dei blocchi componenti.

Istruzione dominante

Si chiama operazione dominante o caratteristica di un algoritmo l'operazione che viene eseguita con maggiore frequenza; nel caso di operazioni eseguite con la stessa frequenza si considera quella più onerosa in termini di tempo impiegato per la sua esecuzione.

Calcolo della complessità

- si considera il numero di dati tendente a infinito, ovvero si calcola la complessità asintotica di un determinato algoritmo
- contempla di norma il peggior caso possibile

Complessità computazionale

“La complessità computazionale si occupa della valutazione del costo degli algoritmi in termini di risorse di calcolo, quali il tempo di elaborazione e la quantità di memoria utilizzata. Nel contesto della complessità computazionale vengono anche studiati i costi intrinseci alla soluzione dei problemi, con l‘obiettivo di comprendere le prestazioni massime raggiungibili da un algoritmo applicato.

Si possono suddividere le classi in due gruppi:

- algoritmi con crescita polinomiale in n→ risolvibile, trattabile
- algoritmi con crescita esponenziale in n→ non risolvibili, intrattabili

P Vs NP

uno dei sette problemi del millennio per cui l’istituto matematico Clay ha messo in palio un milione di dollari per la soluzione; per comprenderlo è necessario conoscere il concetto di complessità computazionale, daremo delle semplici definizioni per gli altri concetti necessari a capire di cosa tratta il problema P vs NP.

I problemi P

I problemi P sono quelli per cui è possibile scrivere un algoritmo, che dato un qualsiasi input ammissibile ne calcoli la soluzione, con complessità computazionale al massimo polinomiale.

Sono i problemi trattabili ovvero quelli per i quali si conoscono degli algoritmi risolutivi per i quali le operazioni richieste anche al crescere della dimensione dell’ingresso non diventano eccessive anche per un calcolatore, sono quindi quelli che possono essere risolti in tempi “umani”.

Esempi di problemi P:

Cercare un elemento all’interno di un array di n posizioni è un problema di tipo P l’algoritmo di ricerca sequenziale lo risolve con complessità computazionale O(*n*).

Ordinare un array di n posizioni è un problema di tipo P, sono conosciuti algoritmi che risolvono questo problema con complessità computazionale O(*n*log(n)*).

I problemi NP

I problemi NP sono quelli per cui dato un ingresso l’algoritmo che verifica se una singola ipotesi di soluzione è corretta in tempo polinomiale.

Attenzione

Attenzione a non farsi trarre in inganno dalla sigla NP che sta per Polinomiale Non-deterministico e non per non polinomiale.

Esempi di problemi NP

- problema del commesso viaggiatore:
  Qual è il percorso più breve che partendo e tornando allo stesso punto tocca determinati punti di una città?

- soddisfacibilità di un'espressione booleana
  data un'espressione booleana funzione di alcune variabili booleane, esiste almeno una combinazione di valori delle variabili che rende vera la funzione?
- fattorizzazione di un numero in fattori primi
  Dato un numero intero fattorizzarlo in fattori primi.

P vs NP – Che relazione c'è tra i problemi P e i quelli NP?

Tra i problemi di tipo NP sono compresi anche tutti i problemi P, perché se è possibile scrivere un algoritmo con una complessità computazionale al più polinomiale che ne calcoli le soluzioni questo stesso algoritmo per verificare una soluzione in tempo polinomiale basta calcolare la soluzione dati gli ingressi e confrontarla con quella data (il confronto tra due soluzioni è un problema di tipo P).

Per quanto detto qui sopra è evidente che quello dei problemi P sia un sottoinsieme dei problemi NP, ovvero che l'insieme dei problemi P sia incluso in quello dei problemi NP.

Il problema del millennio P vs NP riguarda proprio questa inclusione ovvero è se P sia incluso in senso stretto in NP o se P e NP coincidano.

Spieghiamoci meglio se P fosse incluso in senso stretto in NP vuol dire che esisterebbero dei problemi dell'insieme NP per i quali non si potrà mai trovare un algoritmo risolutivo con complessità computazionale al più polinomiale.

Se invece P coincidesse con NP vorrebbe dire che tutti i problemi per cui si può verificare la correttezza di una singola soluzione istruzione con un algoritmo con complessità computazionale al più polinomiale avrebbero un algoritmo che li risolve con complessità computazionale al più polinomiale.

Come procedere con la verifica?

Per provare che P è diverso da NP basterebbe provare che esiste un problema NP per cui non è possibile creare un algoritmo che lo risolva in tempo polinomiale, attenzione bisognerebbe dimostrare che non sia possibile creare un algoritmo con complessità polinomiale e non solo che non sia ancora stato provato.

Per provare il contrario basterebbe invece trovare un algoritmo con complessità computazionale al più polinomiale per un problema che fa parte dell'insieme dei problemi NP-Completi, risoltone uno tutti i problemi NP sarebbero risolvibili con complessità computazionale al più polinomiale perché trasformabili nel problema scelto con una trasformazione che richiede un tempo al più polinomiale.

Ma quindi?

Ad oggi il problema P vs NP è un problema aperto, nessuno è riuscito a dimostrare né che i due insiemi coincidono né che sono diversi tra loro, molti ritengono che P e NP siano insiemi diversi, ma questa è solo un'ipotesi che non è mai stata dimostrata scientificamente.

Ricordo che qualora risolveste la questione P vs NP avreste diritto a un premio da un milione di dollari!