

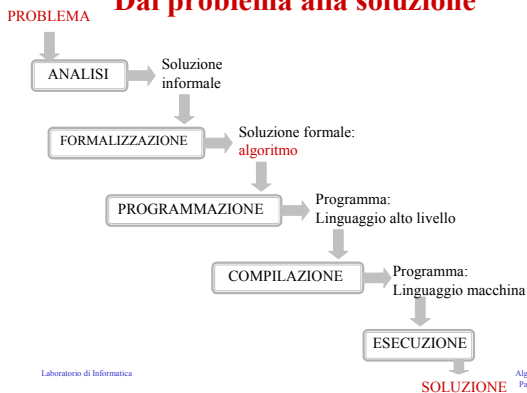
Laboratorio di Informatica

Dal problema al programma

Calcolatore

- Informatica: Scienza della rappresentazione e dell'**elaborazione** automatica dell'informazione
- Calcolatore: Sistema elettronico **programmabile** al fine di svolgere diverse funzioni
 - Nota: la caratteristica fondamentale che lo contraddistingue da altri sistemi elettronici è la programmabilità
- Calcolatore: sistema in grado di eseguire un processo computazionale (algoritmo) su dei dati in base a delle regole specificate attraverso un programma

Dal problema alla soluzione



Analisi

- L'analisi del problema è il primo passo; deve fornire:
 - un **nome** e una **breve descrizione** di che cosa si vuol fare;
 - Una **specificazione funzionale** indica
 - quali sono i **dati iniziali**, cioè quelli da elaborare
 - detti anche **ingressi**
 - che **risultato** si vuole, **in funzione degli ingressi**
 - detto anche **uscita**

Analisi: esempio

- **Nome: Radici**
- **Descrizione:** vogliamo trovare le soluzioni reali di un'equazione di secondo grado
- **Specificazione funzionale:**
 - **Argomenti o ingressi:**
 - **a, b, c:** numeri reali, coefficienti dell'equazione
 - **Risultati o uscite:**
 - "nessuna radice"
 - " $x_1 = x_2 = r$ " se l'equazione ax^2+bx+c ha radici coincidenti = **r**
 - " $x_1 = r_1, x_2 = r_2$ " se l'equazione ax^2+bx+c ha radici distinte = **r1, r2**

Algoritmi, una panoramica

Storia

La parola algoritmo deriva da nome di un matematico persiano:

Abu ja'far Mohammed ibn Musa al-Khowarizmi
Baghdad 825 d.C.

Definizione sintetica

- Una sequenza di passi, definiti con precisione, che portano alla realizzazione di un compito

Esempio

- Giri a destra al primo incrocio, poi a sinistra al secondo semaforo che incontra; dopo trecento metri troverà un incrocio: prosegua diritto fino a che non giunge al confine comunale, poi giri subito a sinistra: quella è la via cercata.
- Vada per circa due chilometri mantenendo questa direzione (la frase è accompagnata da un'indicazione con la mano); troverà uno stradone, percorra lo stradone, giri a destra, a quel punto è arrivato.

Definizione

1. un insieme finito e ordinato di passi
 - a. elementari (**Comprensibilità**)
 - b. non ambigui (**Precisione**)
2. che determinano un procedimento atto a risolvere in un tempo finito (**Eseguibilità**)
3. un problema o una classe di problemi (**Generalità**)
4. fornendo la soluzione corretta (**Correttezza**)
5. per ogni istanza iniziale ammissibile (**Gestione delle Eccezioni**)

Efficienza

- Un algoritmo è efficiente se raggiunge la soluzione del problema nel minor tempo possibile ed utilizzando il minor numero di risorse

Complessità: cenni

- Dato un problema la cui conoscenza è totale, per cui esiste una soluzione teorica/concettuale, esiste sempre un algoritmo che trova la soluzione ottima al problema?
 - Da un punto di vista informatico/algoritmico la risposta è no.

Complessità

- Che cosa determina la risolubilità di un problema da un punto di vista informatico?
 - Tempo (numero di passi) necessario a computare la soluzione, ovverosia il tempo di calcolo (complessità in tempo)
 - Spazio necessario alla memorizzazione dei dati intermedi per il ritrovamento della soluzione (complessità in spazio).

La classe P

- In termini informatici, dire che un problema è risolubile significa dire che il problema appartiene alla classe P, ovverosia alla classe dei problemi per i quali esiste almeno un algoritmo il cui tempo di calcolo è al più polinomiale nella dimensione dell'ingresso.

Un problema di classe P: ricerca

- Data una lista di coppie non ordinate (nome, numero di telefono), trovare il numero di telefono di Mario Rossi (assunto che sia in elenco).
 - La dimensione dell'ingresso è 10000 (ci sono 10000 persone in elenco)
 - Il tempo necessario per trovare la soluzione è al massimo 10000, perché per trovare il numero di Mario Rossi devo scorrere al massimo 10000 elementi della lista.
- Quindi se ci sono n persone in elenco, per risolvere il problema, il programma fa al massimo n confronti.

$$\leq n$$

Un altro problema di classe P

- Sempre nelle ipotesi precedenti, supponiamo di voler capire se ci sono due persone con lo stesso numero di telefono di Mario Rossi.
 - Devo prima cercare Mario Rossi e memorizzarne il numero -> 10000 operazioni
 - Devo poi cercare un'altra persona con lo stesso numero di Mario Rossi -> altre 10000 operazioni.
- Il tempo massimo totale è $10.000+10.000=20.000$
- In questo caso ho $\leq 2n$, il numero di operazioni è doppio del caso precedente ma è sempre lineare.

Complessità

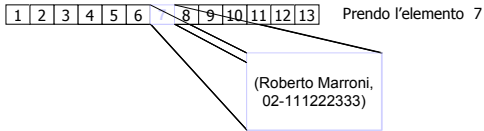
- Che cosa contraddistingue la bontà di un algoritmo rispetto a un altro?
 - Velocità di esecuzione (complessità in tempo)
 - Occupazione di memoria (complessità in spazio)
- Che cosa è possibile fare per rendere più efficiente un algoritmo?
 - Ridurre il numero di operazioni dell'algoritmo al minimo.
 - Non aggiungere il calcolo di informazione inutile o ridondante.

Problema di ricerca ordinato

- Consideriamo il problema sollevato in precedenza:
 - Cambierebbe qualcosa in termini di tempo di calcolo se l'elenco dei nomi fosse ordinato lessicograficamente?
 - No, ma se si cambia anche l'algoritmo si potrebbe avere un incremento di prestazioni sostanziale.

Ricerca binaria

- Cerchiamo in modo efficiente il numero di telefono di Mario in un elenco ordinato lessicograficamente di 13 coppie (nome, numero). L'idea è la seguente:



Roberto Marroni viene prima di Mario Rossi in elenco?

Mario è al numero 10!

Divide et Impera

- Per rintracciare Mario Rossi, abbiamo utilizzato la tecnica di ricerca dicotomica.
- Tale tecnica appartiene alla più generale classe di algoritmi denominata "Divide et Impera".
- Tale classe è caratterizzata dal suddividere il problema primario in diversi sotto problemi per cui trovare la soluzione è più facile.
- Latini docent
- $\leq \log_2 n < n$
- $N=1024 \rightarrow$ Ricerca sequenziale al più 1024 passi
Ricerca binaria al più 10 passi

E se l'elenco non fosse ordinato?

- E' quindi necessario ordinare l'elenco prima di effettuare una ricerca dicotomica.
- Ma quanto costa ordinare un elenco?
 - Ci sono molti algoritmi che ordinano una lista di elementi
 - Quick-sort: $\leq (n \log_2 n)$

Quindi...

- Se devo eseguire 100 ricerche, su un elenco di 1024 (2^{10}) nomi
 - Ordinamento: $1024 * \log(1024) = 10240$
 - Ricerca dicotomica: $\log(1024) = 10$
 - Per 100 ricerche avremo: $10240 + 100 * 10 = 11.240$
 - Ricerca sequenziale: $1024 * 100 = 102.400$
 - che vuol dire che mi conviene ordinare la lista.
- N.B.: Ordino la lista **una volta sola!**

I problemi difficili

- Da un punto di vista prettamente informatico, quando un problema è difficile?
 - Quando l'algoritmo che lo risolve in modo ottimo richiede troppo tempo (o troppo spazio) per fornire la soluzione.

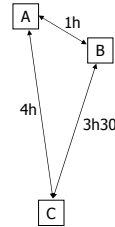
La classe NP

- Un problema è "difficile" quando appartiene alla classe NP, ovverosia alla classe di problemi per i quali non esiste un algoritmo risolvibile in tempo polinomiale.

Un problema NP, il commesso viaggiatore

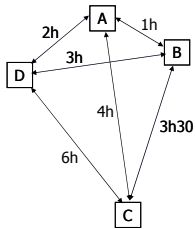
- Un commesso viaggiatore deve raggiungere N città: scegliere l'ordine delle città da visitare in modo che si minimizzi il tempo per visitarle tutte. Per ogni coppia di città (A,B) si conosce il tempo necessario a raggiungere B partendo da A o viceversa.

Il commesso viaggiatore



- Un caso ovvio, partendo da A:
 - Da A andremo a B
 - Da B a C
 - Tempo necessario: 4h30
- Fissato il punto di partenza A si possono fare solo due "viaggi": (A, B, C) oppure (A, C, B)

Il commesso viaggiatore

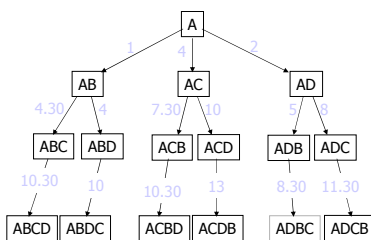
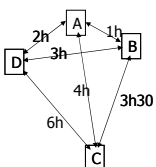


- La soluzione è già meno ovvia. Perché? Quanti sono i possibili viaggi, assunto che si parte da A?
 - (A,B,C,D) = 10h30
 - (A,C,B,D) = 10h30
 - (A,C,D,B) = 13h
 - (A,D,C,B) = 11h30
 - (A,B,D,C) = 10h
 - (A,D,B,C) = 8h30

Il commesso viaggiatore

- Aggiungendo una città, si è passati da 2 possibili soluzioni a 6 possibili soluzioni fra le quali ricercare la soluzione ottima.
- Se il commesso dovesse passare per 30 città? Ci sarebbero 30! soluzioni, ovverosia
2652528598121910000000000000000000

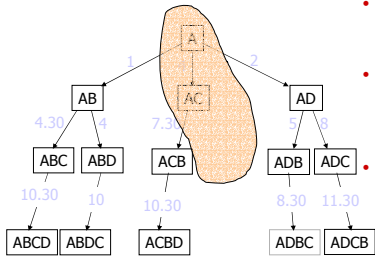
Un algoritmo per il commesso viaggiatore : "srotoliamo".



Devo vedere tutte le soluzioni per trovare l'ottima?

- No, non è necessario. Si possono omettere le soluzioni palesemente sbagliate. Ma che cosa vuol dire che una soluzione è palesemente sbagliata?
- Scorrendo l'albero dall'alto verso il basso, da sinistra verso destra la prima soluzione che si incontra è [ABCD,10.30], la seconda è [ABDC,10]

Bounding

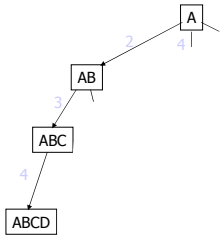


- Si noti che il cammino ACD costa 10.
- Ha senso cercare una soluzione peggiore di [ADBC, 10]?
- Ovviamente no, quindi possiamo evitare di andare oltre.

Il bounding dipende dall'albero?

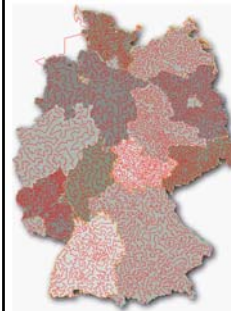
- Sì. Se l'albero fosse costruito in un modo diverso (ad esempio con una soluzione molto buona sui primi rami), potremmo eliminare molte soluzioni sub-ottime.
- Il problema è costruire l'albero in maniera efficiente, ma non ci sono strategie furbe da seguire.
- E' comunque una strategia utilizzata quando il numero di città è tra 40 e 60.

Infatti, se ad esempio l'albero fosse così...



- La soluzione ottima è [ABCD, 4]
- Posso eliminare praticamente tutto l'albero.

Un problema reale: d15112



- 15112 città in Germania.
- Tempo stimato di calcolo: 22,6 anni con 110 processori.
- Risolto nel 2001

Riassumendo

- Per i problemi appartenenti alla classe NP
 - Non si riesce a trovare soluzioni ottime in tempi accettabili
 - O ci si accontenta di soluzioni approssimate
 - O si usano algoritmi probabilistici

Un problema di ottimizzazione: il problema dello zaino

- Ho uno zaino di capacità C e N oggetti o_1, o_2, \dots, o_N , di volume s_1, s_2, \dots, s_N e valore v_1, v_2, \dots, v_N
- Devo riempire il mio zaino in modo da massimizzare il valore totale degli oggetti che mi porto via.

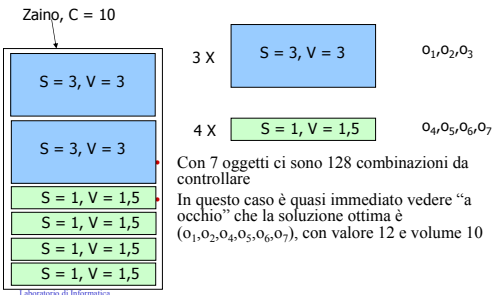
La soluzione ottima

- Per ottenere sempre la soluzione ottima, ossia quella che massimizza il valore dello zaino pieno, l'unico modo è cercarla tra tutte, un po' come nel primo algoritmo per il rintracciamento di Mario Rossi.
- Se, però, aggiungere un numero all'elenco comportava al massimo un confronto in più, qui aggiungere un oggetto significa raddoppiare i confronti.

La soluzione ottima

- Devo infatti controllare, per ogni oggetto, la possibilità che esso vada inserito nello zaino e che non venga inserito: in numeri, con 30 oggetti, devo controllare più di 1000000000 di combinazioni, con 100 oggetti servono 1267650600228229401496703205376 confronti, il che rende il problema dello zaino un problema la cui soluzione ottima è impossibile da trovare.

La soluzione ottima, un caso trattabile a occhio



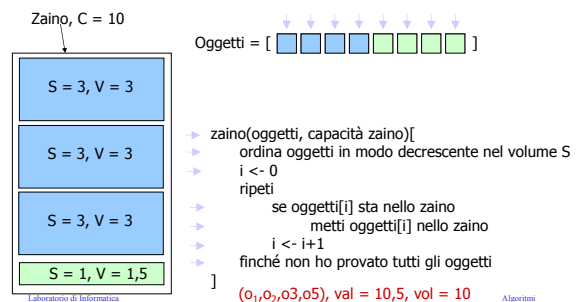
E se ci sono più oggetti?

- Come si risolve questo problema? Si è visto che cercare la soluzione ottima è impossibile, e quindi ci si accontenterà di trovare soluzioni "sub-ottime" o approssimate
- Vedremo un algoritmo appartenente a una famiglia nota in letteratura come famiglia "greedy", che significa golosone, ghiotto

Greedy

- L'idea è tanto semplice quanto miope: scegliere di mettere nello zaino sempre l'oggetto di maggior valore, finché lo zaino non è pieno. E' un algoritmo che costruisce la soluzione finale aggiungendo di volta in volta la migliore soluzione locale.
 - PRO: è molto veloce.
 - CONTRO: le soluzioni trovate possono essere ben lontane da quella ottima.

L'algoritmo greedy



Un miglioramento greedy

- Se ordinassimo la lista degli oggetti, anziché in funzione del solo valore, in funzione del valore “specifico” V/S di ogni oggetto?

Oggetti = [3 3 3 3 1 1 1 1] (come nell'es prec)

Si otterrebbe

Oggetti = [1,5 1,5 1,5 1,5 1 1 1 1]

E la soluzione greedy sarebbe $(o_5, o_6, o_7, o_8, o_1, o_2)$, val = 12, vol = 10

UGUALE A QUELLA OTTIMA!

E' stato comunque un caso...

- ...aver trovato la soluzione ottima. Anche questo accorgimento ha molti punti deboli, esattamente come il primo algoritmo visto.
- Quale algoritmo si usa, quindi?

Algoritmo di Sahni

- Gli n oggetti sono ordinati in modo decrescente nel valore specifico.
- Scelgo un $k < n$
- Per ogni $i \leq k$ prendo tutte le combinazioni di i oggetti su n
- Inserisco ogni combinazione di oggetti nello zaino (rispettando i vincoli)
- Se c'è altro spazio completo la soluzione con un algoritmo greedy.
- Tra tutte le soluzioni generate, scelgo la migliore.

Sahni: prestazioni

- Quanti modi di scegliere k oggetti su n ci sono?

- $\binom{n}{k}$

- Quanto costa ordinare n elementi?

- $n \log n$

- Il costo totale è quindi:

$$n \log n \sum_{i=1}^k \binom{n}{i} \leq n \log n \sum_{i=1}^k 2^i \leq n \log n * 2^{k+1}$$

Sahni: correttezza

- E' possibile dimostrare che, con tale algoritmo, l'errore relativo è minore di $1/k$, ossia

$$\frac{\text{ottima} - \text{approssimata}}{\text{ottima}} \leq \frac{1}{k}$$

Algoritmi probabilistici

- E' una categoria di algoritmi che non sempre risolvono un problema, ovvero che possono sbagliare con una probabilità p
- La strategia consiste nel limitare p

La primalità di un numero

- Dal piccolo teorema di Fermat si deduce che, se X non è primo almeno $\frac{3}{4}$ dei numeri tra 1 e $X-1$ “testimoniano la non primalità di X ”

La primalità di X (algoritmo di Rabin)

- Estraggo quindi un numero random Y tra 1 e $X-1$.
- Se Y testimonia la non primalità di X allora X non è primo.
- Altrimenti X è primo

Concentriamoci sulla correttezza

- E' importante notare come possa non essere sufficiente accontentarsi di una risposta potenzialmente errata (e p , in questo esempio, è anche discretamente alta).
- Come diminuire p ?

Eseguendo l'algoritmo più volte

- Infatti, rieseguendo l'algoritmo, o ottengo che X NON è primo perché estraggo un numero che testimonia la sua non primalità (nel qual caso la prima esecuzione aveva sbagliato), oppure il rischio di sbagliare è $\frac{1}{4}$
- Ma qual è la probabilità di sbagliare 2 volte? E' $p \times p = \frac{1}{4} \times \frac{1}{4} = \frac{1}{16}$
- Se eseguo l'algoritmo per 10 volte, $p=0,00000095$ e posso stare tranquillo sul fatto che X sia primo