ANNO ACCADEMICO 2024/2025

Algoritmi e Complessità

Teoria

Altair's Notes

UNIVERSITÀ DI TORINO

DIPARTIMENTO DI INFORMATICA

Premessa

Licenza

Formato utilizzato

Box di "Concetto sbagliato":

Concetto sbagliato 0.1: Testo del concetto sbagliato

Testo contente il concetto giusto.

Box di "Corollario":

Corollario 0.0.1 Nome del corollario

Testo del corollario. Per corollario si intende una definizione minore, legata a un'altra definizione.

Box di "Definizione":

Definizione 0.0.1: Nome delle definizione

Testo della definizione.

Box di "Domanda":

Domanda 0.1

Testo della domanda. Le domande sono spesso utilizzate per far riflettere sulle definizioni o sui concetti.

Box di "Esempio":

Esempio 0.0.1 (Nome dell'esempio)

Testo dell'esempio. Gli esempi sono tratti dalle slides del corso.

Box di "Note":

Note:-

Testo della nota. Le note sono spesso utilizzate per chiarire concetti o per dare informazioni aggiuntive.

Box di "Osservazioni":

Osservazioni 0.0.1

Testo delle osservazioni. Le osservazioni sono spesso utilizzate per chiarire concetti o per dare informazioni aggiuntive. A differenza delle note le osservazioni sono più specifiche.

1 Introduzione

1.1 Il Corso in Breve...

Questo corso ruota attorno alla parola "*riduzione*": si evita di reinventare l'acqua calda. La riduzione si collega al concetto di *intrattabilità*: non si conoscono algoritmi sufficientemente veloci per risolvere un determinato problema per ogni sua istanza (o non esistono ma non lo si sa dimostrare). Il corso è incentrato sulle tecniche per affrontare problemi intrattabili.

- Brute Force.
- Backtracking.
- Least Cost.
- ...

Note:-

La chiave di lettura per parlare di questi strumenti è proprio la riduzione.

Successivamente si parlerà di *correttezza*: dimostrare che un programma restituisca l'output desiderato per qualsiasi istanza di input.

1.1.1 Problemi Motivazionali

Problema delle Valutazioni: si devono valutare 8 parti di un esame con 50 domande totali per cui si può ottenere un massimo di 125 punti. Il problema consiste nel trovare il modo di massimizzare il voto dato, ma limitandolo a 100 punti.

Domanda 1.1

È facile verificare che l'insieme selezionato è una soluzione? (il voto assegnato è minore o uguale al massimo ammesso)

Domanda 1.2

È facile verificare che l'insieme selezionato è una risposta? (il voto assegnato premia al massimo l'esaminando)

Domanda 1.3

È facile scrivere un algoritmo che fornisce una soluzione?

Domanda 1.4

È facile scrivere un algoritmo che fornisce una risposta?

Problema del Bando: l'incubatore dell'azienda TDT bandisce l'assegnazione di 90 unità di denaro per finanziare 3 progetti, uno per ciascuna delle seguenti aree di intervallo:

- Area 0.
- Area 1.
- Area 2.

I progetti sono classificati in base a un indice di utilità:

- Criterio 1.
- Criterio 2.
- ...

Note:-

Se si decide di finanziare un progetto lo si finanzia per intero.

Domanda 1.5

È facile certificare un insieme soluzione? (il totale finanziato è minore o uguale 90)

Domanda 1.6

È facile certificare un insieme risposta? (il totale finanziato è minore o uguale a 90, con utilità massima)

Domanda 1.7

Cosa ha in comune con gli algoritmi sintetizzati per le valutazioni?

Problema: si vuole prestare denaro per un massimo di 220 unità. Il massimo rischio ammissibile è del 40% (Rischio $= \frac{\text{Richiesta}}{\text{Affidabilità}}$ per singolo finanziamento). Si vuole massimizzare il guadagno.

Domanda 1.8

È facile verificare che l'insieme selezionato è una soluzione? (il totale finanziamenti è minore o uguale a 220 e il rischio è minore o uguale al 40%)

Domanda 1.9

È facile verificare che l'insieme selezionato è una risposta? (il totale finanziamenti è minore o uguale a 220 e il rischio è minore o uguale al 40%, massimizzando il guadagno)

Domanda 1.10

Cosa ha in comune con gli algoritmi sintetizzati in precedenza?

Note:-

Questi problemi possono essere ridotti al knapsack.

1.1.2 Strategie di soluzione

- Calcolo classico: soluzione a problemi che coinvolgono funzioni, derivate e integrali.

Dizionario di riferimento (Bellman):

- *Risorsa economica*: denaro, persone, materiali, etc. disponibile in quantità x.
- *Attività*: modi diversi, da 1 a N, in cui possiamo usare la risorsa economica.
- *Quantità allocate*: $x_1, \ldots, x_N$, una per ogni attività.
- *Ricavo singola attività*: $g_i(x_i)$ relativo all'attività i.
- *Ricavo*: valore globale che dipende dal modo in cui ogni attività usa la porzione allocata della risorsa.

Esempio 1.1.1 (Valutazioni)

- *Risorsa economica*: 100 unità
- *Attività*: 50 correzioni concorrenti.
- *Quantità allocate*: inclusioni tra risposte per voto.
- *Ricavo singola attività*: voto effettivamente assegnabile da risposta.
- *Ricavo*: voto finale.

Note:-

Il calcolo classico non è sufficiente a risolvere i problemi motivazionali.

Domanda 1.11

La *programmazione lineare* (PL) e la *programmazione lineare intera* (PLI) sono alternative al calcolo classico?

La PL non è adatta: le soluzioni ai problemi Valutazioni, Bando e Rischio assumono la forma di tuple $x_1^*, \ldots, x_N^*$, in cui ogni $x_i^* \in \{0, 1\}$. In accordo col vocabolario iniziale, questo significa che ogni attività coincide con una scelta:

- Se x_i^*, l'attività consiste nel *non scegliere* di inserire il ricavo locale g_i in quello totale.
- Se x_i^*, l'attività consiste nel *scegliere* di inserire il ricavo locale g_i in quello totale.

Le tecniche algoritmiche offerte dalla *PL* sono sviluppate per ottenere valori delle variabili $x_1^*, \ldots, x_N^*$ di problemi esprimibili almeno come:

$$\max/\min_{x_1, x_2, \ldots, x_N} \quad g_1(x_1) + g_2(x_2) + \cdots + g_N(x_N) \tag{1.1}$$

$$\ldots$$

$$x_i \in \mathbb{R} \quad (i \in \{1, \ldots, N\}) \tag{1.2}$$

in cui le variabili posso assumere valori in $\mathbb{R}$. Quindi, non possiamo immaginare di risolvere alcuno dei problemi motivazionali con tecniche di *PL*.

Anche la PLI può essere inefficiente: la *PLI* si distingue dalla PL perché fornisce tecniche per risolvere problemi della forma:

$$\max/\min_{x_1,x_2,\dots,x_N} \quad g_1(x_1) + g_2(x_2) + \cdots + g_N(x_N) \tag{1.3}$$

$$\dots$$

$$x_i \in \{0,1\} \quad (i \in \{1,\dots,N\}) \tag{1.4}$$

in cui le variabili possono assumere valori in $\{0,1\}$.

Note:-

Imponendo variabili discrete, il comportamento della funzione da massimizzare (o minimizzare) diventa non derivabile. Quindi la ricerca dei valori ottimali deve avanzare per tentativi.

Le euristiche Greedy:

- Allocano una quantità di risorsa per la prossima attività tale da massimizzare il ricavo in quel momento.
- Non ammettono "ripensamenti": non è possibile ritrattare sulla quantità di risorse assegnata a un'attività una volta decisa.

Due euristiche Greedy per valutazioni:

1. Si sceglie in base al miglior valore assoluto del voto assegnato.
2. Si sceglie in base al miglior rapporto $\frac{\text{voto assegnato}}{\text{voto massimo assegnabile}}$.

Domanda	Voto Max	Voto Assegnato
1	2	2
2	5	4.6
3	4	3.9
4	2	2

Voto massimo assegnabile: 9

Figure 1.1: Esempio di valutazioni.

Osservazioni 1.1.1

- Se si applica la prima euristica si assegna 7.9 (2+2+3.9).
- Se si applica la seconda euristica si assegna 8.5 (4.6+3.9).

Però il massimo possibile è 8.6 (2+2+4.6).

Note:-
Nessuna delle due scelte è ottimale.

1.2 Brute-Force

Si vuole trovare un algoritmo per risolvere il problema delle valutazioni a *basso costo computazionale*.

Algoritmo 1.2.1 *(Brute-Force)*: L'algoritmo Brute-Force effettua una visita esaustiva dello spazio delle permutazioni.

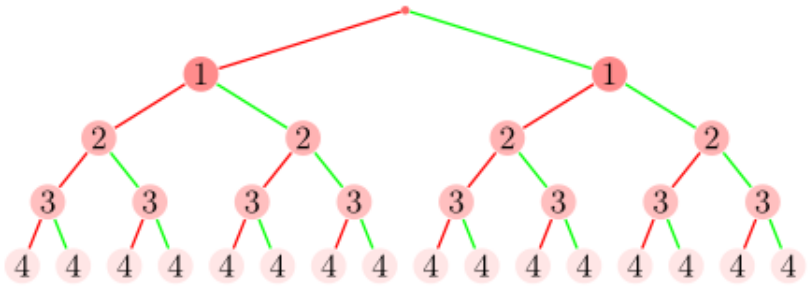

Figure 1.2: Applicazione di Brute-Force.

Note:-
Un ramo rosso indica falso (risposta non presa), un ramo verde indica true (risposta presa).

Osservazioni 1.2.1

- La strategia Brute-Force su permutazioni è completa: indivisua *almeno* una risposta che corrisponde alla soluzione migliore.
- Le euristiche Greedy non sono complete.
- Con la generazione di permutazioni si può, senza cambiare nulla, avere la soluzione per il problema dei bandi.

1.2.1 Permutazioni, Disposizioni e Sottoinsiemi

Domanda 1.12

Dato il segmento $[1, \dots, n]$, sappiamo esistere $n!$ permutazioni, dato un qualsiasi numero $0 \leqslant N < n!$, esiste un metodo per ricavare univocamente una permutazione a partire da N?

I passi per rispondere affermativamente:

- Trasformare N in notazione *factoradic* N_f.
- Usare N_f per generare una permutazione, riorganizzando gli elementi in $[1, \ldots, n]$.

$$N = \sum_{i=1}^{n-1} a_i i!$$

$$(0 \leqslant a_i \leqslant i)$$

$$\begin{aligned}
719 &= \sum_{i=1}^{6-1} a_i \cdot i! \\
&= 5 \cdot 5! + 4 \cdot 4! + 3 \cdot 3! + 2 \cdot 2! + 1 \cdot 1! + 0 \cdot 0! \\
&= (5 \cdot 5 + 4) \cdot 4! + 3 \cdot 3! + 2 \cdot 2! + 1 \cdot 1! + 0 \cdot 0! \\
&= ((5 \cdot 5 + 4) \cdot 4 + 3) \cdot 3! + 2 \cdot 2! + 1 \cdot 1! + 0 \cdot 0! \\
&= (((5 \cdot 5 + 4) \cdot 4 + 3) \cdot 3 + 2) \cdot 2! + 1 \cdot 1! + 0 \cdot 0! \\
&= ((((5 \cdot 5 + 4) \cdot 4 + 3) \cdot 3 + 2) \cdot 2 + 1) \cdot 1! + 0 \cdot 0! \\
&= ((((5 \cdot 5 + 4) \cdot 4 + 3) \cdot 3 + 2) \cdot 2 + 1) \cdot 1! \ .
\end{aligned}$$

Figure 1.3: Numero più alto rappresentabile con $n = 6$.

Esempio 1.2.1

Supponiamo di voler usare 463 per generare una permutazione dei 6 elementi in $l = \{A, B, C, D, E, F\}$. Abbiamo $(463)_{10} = (341010)!$. Ogni cifra di $(341010)!$ è l'indice, a partire da 0 dell'elemento da estrarre per produrre una permutazione da l:

- 3 estrae l'elemento di indice 3. Da $\{A, B, C, D, E, F\}$ a $\{A, B, C, E, F\}$.
- 4 estrae l'elemento di indice 4. Da $\{A, B, C, E, F\}$ a $\{A, B, C, E\}$.
- 1 estrae l'elemento di indice 1. Da $\{A, B, C, E\}$ a $\{A, C, E\}$.
- 0 estrae l'elemento di indice 0. Da $\{A, C, E\}$ a $\{C, E\}$.
- 1 estrae l'elemento di indice 1. Da $\{C, E\}$ a $\{C\}$.
- 0 estrae l'elemento indice 0. La lista $\{C\}$ si svuota.

La permutazione risultante è quindi: $\{\mathbf{D}, \mathbf{F}, \mathbf{B}, \mathbf{A}, \mathbf{E}, \mathbf{C}\}$.

Note:-

Però questo metodo è molto inefficiente rispetto ad altri (ha complessità $O(n^2)$, ma usando balanced tree si scende a $O(n * log(n))$).

2

Computazione Quantistica