Insegnamento di algoritmi avanzati

Algoritmi probabilistici

Roberto Posenato

ver. 0.5, 01/10/2009

Idea generale

L'importante è scegliere!

- Quando un algoritmo deve eseguire una scelta e fare la scelta ottima richiede un tempo proibitivo, allora una soluzione alternativa è fare una scelta casuale e determinare una soluzione non ottima.
- Se una scelta non ottima ha probabilità bassa e comporta un errore limitato nella soluzione finale, allora la ripetizione dell'esecuzione permette di limitare la probabilità di sbagliare e porta alla soluzione ottima in un tempo inferiore rispetto alla ricerca deterministica.

Idea generale

Definizione di algoritmo probabilistico

Definizione 1 (Algoritmo probabilistico)

Un algoritmo probabilistico (randomizzato) è un algoritmo la cui computazione dipende da una o più scelte casuali.

Esso può essere sempre riscritto nella forma:

Procedura `Probabilistico(`x`)`

1: $c =$ `random(`x`);` // genera una variabile casuale
2: $y =$ `costruisciSoluzione(`x, c`)` // usa c per costruire la soluzione
3: **return** (y);

Idea generale

Proprietà di un algoritmo probabilistico

1. Il medesimo algoritmo probabilistico può fornire risultati diversi per il medesimo input;
2. Un algoritmo probabilistico può quindi determinare un risultato errato. La probabilità di risultato errato deve essere ragionevolmente piccola ($<< \frac{1}{2}$) per qualsiasi istanza;
3. Maggiore è il numero di volte in cui viene eseguito un algoritmo probabilistico, maggiore è la *confidenza* della soluzione fornita;
4. Generalmente un algoritmo probabilistico ha una complessità migliore nel caso medio rispetto all'equivalente deterministico;

Idea generale

Confronto con gli algoritmi deterministici e non

- Un algoritmo deterministico esegue un numero finito di passi per trovare la soluzione del problema;
- Un algoritmo non deterministico utilizza una funzione `indovina` per costruire una potenziale soluzione e una funzione `verifica` per decidere se la soluzione "indovinata" è una soluzione positiva;

Un algoritmo probabilistico:

- non è un algoritmo deterministico perché prevede un grado di casualità nella generazione della soluzione.
- non è un algoritmo non deterministico perché la casualità viene usata per costruire una soluzione e non per fornire un risultato casuale.

Idea generale

Prima classificazione algoritmi probabilistici

La casualità può essere usata in tre modi:

Algoritmi numerici per fornire un risultato approssimato con una certa confidenza.
Esempio: *Con probabilità 90%, la risposta è* 60 ± 1.
Più volte si esegue l'algoritmo, maggiore diventa la precisione o la confidenza.

Algoritmi di Monte Carlo per fornire un risultato esatto con alta probabilità benché talvolta possano fornire una risposta errata.
Esempio: *Con probabilità 99%, la risposta è* 60.
Più si esegue l'algoritmo, maggiore diventa la probabilità che il risultato sia esatto.

Algoritmi Las Vegas per fornire un risultato esatto con certezza.
L'algoritmo può anche non rispondere.
Esempio: *La risposta è* 60.

Idea generale

Prima classificazione algoritmi probabilistici

Tipi di algoritmi probabilistici

Anche se banale, si assuma che si voglia costruire un algoritmo probabilistico che stampi le date di eventi storici.
Alla domanda “Quando Cristoforo Colombo scoprì l’America?”, le risposte che si possono ottenere dai tre tipi di algoritmi, eseguiti per 5 volte consecutive, sono del tipo:

Algoritmi numerici 1490 ± 10, 1485 ± 10, 1492 ± 10, 1491 ± 10, 1492 ± 10;

Algoritmi di Monte Carlo 1492, 1500, 1492, 1492, 1491;

Algoritmi Las Vegas 1492, no ouput, 1492, 1492, no output.

Idea generale

Generatore di numeri casuali

- Le scelte casuali devono avere una distribuzione uniforme $[0, 1)$: valori in intervalli diversi sono ottenibili da questi per traslazione.
- Generatori casuali reali non sono praticamente reperibili e non è realistico pensare che forniscano valori reali tra 0 e 1.
- Ci si accontenta quindi di usare i generatori di numeri pseudo-casuali.

Definizione 2 (Generatore di numeri pseudo-casuali)

Un generatore di numeri pseudo-casuali è una funzione deterministica in grado di generare una sequenza di valori $x_1, \ldots, x_n$ indistinguibile da un'altra sequenza di valori casuali $y_1, \ldots, y_n$ in tempo polinomiale.

Idea generale

Generatore di numeri casuali

Nota!

In pratica:

- questi generatori determinano sequenze di valori pseudo-casuali con una precisione fissata a partire da un certo *seme*.
- Medesimo seme in tempi diversi determina medesime sequenze.
- Tutte le sequenze sono necessariamente periodiche!
- Una scelta corretta del generatore permette di avere sequenze che, a fini dell'applicazione, possono considerarsi casuali.

Idea generale

Tempo atteso vs Tempo medio

Tempo medio di un algoritmo

È la media dei tempi di computazione su tutti gli input della medesima dimensione assumendo questi equiprobabili:

$$T_{medio}(n) = \frac{\sum_i T(istanza_i^n)}{|\{\text{istanze}^n\}|}.$$

Conosciamo già i limiti di tale concetto!

Tempo atteso di un algoritmo probabilistico

È definito per ciascuna istanza ed è la media dei tempi richiesti per risolvere l'istanza: $T_{atteso}(istanza) = E[T(istanza)]$

Visto che si sono output diversi, i tempi di calcolo possono essere diversi, da cui la definizione.

Il valore NON dipende dalla distribuzione degli input, ma dalle scelte casuali fatte dall'algoritmo.

Idea generale

Tempo atteso vs Tempo medio

Come descrivere le prestazioni di un algoritmo probabilistico?

- Per un algoritmo probabilistico è quindi opportuno parlare di *tempo atteso medio* o *tempo atteso nel caso peggiore*.
- Questo ha particolare significato nel caso peggiore: si determina il tempo atteso valutando l'istanza peggiore possibile e non il tempo di una singola esecuzione su un'istanza con la peggiore scelta probabilistica.

Esempio 1

Quicksort randomizzato ha un *tempo atteso nel caso peggiore* pari a $O(n \log n)$. L'istanza peggiore, che richiede tempo $O(n^2)$, se viene risolta dall'algoritmo deterministico, richiederà sempre tempo $O(n^2)$, se viene risolta dalla versione randomizzata potrà richiedere $O(n^2)$, ma in media richiederà $O(n \log n)$.

Algoritmi probabilistici numerici

Definizione

La casualità è stata inizialmente usata per determinare soluzioni numeriche approssimate.

Definizione 3 (Algoritmo probabilistico numerico)

Un algoritmo probabilistico è un algoritmo numerico quando determina una soluzione approssimata con un intervallo di confidenza del tipo "con probabilità p la soluzione corretta è $y \pm \varepsilon$".

Algoritmi probabilistici numerici

Esempio per la stima di π

Esiste un risultato curioso circa π:

Teorema 1 (Teorema di Leclerc)

Si consideri un ago e un pavimento di listelli di larghezza costante pari al doppio della lunghezza dell'ago e con fessure di larghezza nulla. La probabilità che l'ago cada di traverso su una fessura del pavimento è $\frac{1}{\pi}$.

Se si lanciano n aghi, il numero di aghi che sono di traverso sulle fessure sono $k = \frac{n}{\pi}$, ovvero la media di un processo binomiale di probabilità $\frac{1}{\pi}$.

Idea: algoritmo di Buffon

Con un adeguato numero di lanci n, si può stimare π contando il numero di aghi che sono di traverso e calcolare $\pi \approx \frac{n}{k}$.

Algoritmi probabilistici numerici

Esempio per la stima di π

Qual è la qualità delle soluzioni?

- Sia X_i la v.c. bernoulliana che descrive l'i-esimo ago: 1 se l'ago cade di traverso su una fessura, 0 altrimenti.
- Per il teorema, $Pr[X_i = 1] = \frac{1}{\pi}$ per ciascun i.
- Proprietà distribuzione bernoulliana: $E(X_i) = \frac{1}{\pi}$, $Var(X_i) = \frac{1}{\pi}(1 - \frac{1}{\pi})$
- Sia X la v.c. binomiale che descrive la media (e non la somma!) di n lanci: $X = \sum_{i=1}^{n} X_i/n$: $Pr[X = \frac{k}{n}] = \binom{n}{k} \left(\frac{1}{\pi}\right)^k \left(1 - \frac{1}{\pi}\right)^{n-k}$
- Proprietà distribuzione binomiale: $E(X) = \frac{1}{\pi}$ e $Var(X_i) = \frac{1}{n\pi}(1 - \frac{1}{\pi})$.

Algoritmi probabilistici numerici

Esempio per la stima di π

continua "Qual è la qualità delle soluzioni?"

richiamo Teorema del Limite Centrale

Siano $(X_n)_{n\geq 0}$ v. c. indipendenti e identicamente distribuite, tali che $E(X_n) = m < \infty$ e $var(X_n) = \sigma^2 < \infty$. La successione

$$Z_n^* = \frac{\sum_{i=1}^{n} X_i - nm}{\sqrt{\sigma^2 n}}$$

converge ad una v.c. gaussiana di media 0 e varianza 1.

Nota!

Già con $n > 30$ si ha una buona approssimazione!

Algoritmi probabilistici numerici

Esempio per la stima di π

continua "Qual è la qualità delle soluzioni?"

- Dalla tabella della distribuzione normale, si determina che
$$Pr\left[|X - E(X)| < 2,576\sqrt{var(X)}\right] \approx 99\%$$
- Se si sostituiscono i valori noti e si pone $\varepsilon > 2,576\sqrt{\frac{1}{n\pi}(1-\frac{1}{\pi})}$, si ottiene
$$Pr[|X - \frac{1}{\pi}| < \varepsilon] \geq 99\% \text{ quando } n > \frac{1,440}{\varepsilon^2}$$

Nota!

n è stato determinato usando π, valore che stiamo cercando!
Una stima corretta di n dovrebbe avvenire mediante l'uso della varianza campionaria e della distribuzione di Student, che NON usano il π.
Qui siamo interessati solo a dimostrare qual è la qualità delle soluzioni trovate e quindi si usa π direttamente.

Algoritmi probabilistici numerici

Esempio per la stima di π

continua "Qual è la qualità delle soluzioni?"

- Dalla stima di $\frac{1}{\pi}$ passiamo alla stima di π:

$$|X - \frac{1}{\pi}| < \varepsilon \Longrightarrow |\frac{1}{X} - \pi| < \frac{\varepsilon\pi^2}{1 - \varepsilon\pi}$$

- quando $\varepsilon < 0,00415$, $9.8\varepsilon < \frac{\varepsilon\pi^2}{1-\varepsilon\pi} < 10\varepsilon$;
- con lo stesso numero di lanci, la stima di π ha un decimale di precisione in meno rispetto alla stima di $1/\pi$;
- l'errore relativo rimane comunque lo stesso poichè π è circa 10 volte più grande di $1/\pi$;
- Riassumendo, per ottenere una stima di π con un errore assoluto $\varepsilon < 0,0415$ con confidenza del 99% occorrono $n \geq 144/\varepsilon^2$ lanci di aghi.

Algoritmi probabilistici numerici

Esempio per la stima di π

Esercizio 1 (Complessità)

- Si dimostra che il costo di determinare se un ago è di traverso su una fessura è costante.
- Si assuma che il costo di simulazione di lancio sia pure costante.
- Qual è il costo dell'algoritmo?
- Per rispondere, fissare prima qual è la dimensione dell'input (non solo i valori!!) e poi determinare il costo finale.

Algoritmi probabilistici numerici

Altri applicazioni

Esempi di problemi risolvibili da algoritmi probabilistici numerici

- Problema dell'integrazione numerica. In letteratura spesso l'algoritmo che risolve questo problema è erroneamente chiamato Algoritmo Monte Carlo.
- Problema del contatore: usare un intero di n bit per contare un numero di eventi $>> 2^n$.

Algoritmi di Monte Carlo

Definizione

Definizione 4 (Algoritmo probabilistico Monte Carlo)

Un algoritmo probabilistico è un algoritmo di Monte Carlo quando è in grado di calcolare la soluzione corretta con alta probabilità.

Nota!

- data una soluzione y, non è possibile dire se la soluzione è corretta. Si conosce solo la probabilità di correttezza della soluzione;
- è possibile convincersi della correttezza della soluzione eseguendo più volte l'algoritmo sullo stesso input.

Algoritmi di Monte Carlo

Quando il problema è di decisione

Solitamente un algoritmo di Monte Carlo è costruito in modo tale che:

- La risposta **sì** è sempre esatta;
- La risposta **no** è esatta con probabilità $p \geq \frac{1}{2}$;
- Quindi l'errore può essere un falso negativo con probabilità al più $1 - p$.

Algoritmi di Monte Carlo

Esempio per PRIMALITÀ

Si consideri il seguente problema

PROBLEMA PRIMALITÀ (PRIMALITY)

DESCRIZIONE: Un intero dispari n.

QUESITO: n è primo?

Nota!

- Fondamentale per costruire le chiavi dei più importanti sistemi crittografici;
- Nel 2002, M. Agrawal, N. Kayal e N. Saxena hanno fornito un algoritmo deterministico, AKS primality test, di complessità $O((\log n)^{6+\varepsilon})$.
- Se $n \approx 2^{512}$ il calcolo deterministico richiede $512^6 \approx 1,80 \cdot 10^{16}$ operazioni.
  Se ogni operazione costa 65ps, il tempo finale è pari a 325h c.

Algoritmi di Monte Carlo

Esempio per PRIMALITÀ

Esiste un antico risultato che permette una soluzione algoritmi alternativa:

Teorema 2 (Teorema piccolo di Fermat)

Sia n primo. Allora per ogni intero $1 \leq a \leq n-1$ (a coprimo con n) vale

$$a^{n-1} \equiv 1 \pmod{n}$$

Nota!

La contronominale dice che se due interi n e a sono tali che $1 \leq a \leq n-1$ e $a^{n-1} \not\equiv 1 \pmod{n}$, allora n non è primo.

Algoritmi di Monte Carlo

Esempio per PRIMALITÀ

La contronominale suggerisce un primo algoritmo probabilistico:

Funzione `Fermat(`n`)`

// $n > 2$

1: $a =$ `uniform`$(1, n-1)$; // valore casuale uniforme tra 1 e n-1
2: **if** `expMod`$(a, n-1, n) == 1$ **then return** ***vero***;
3: **else return** ***falso***;

Nota!

La funzione `expMod` determina la potenza $n-1$ di a modulo n usando *divide et impera*.
La sua complessità è $\Theta((\log a)^{\log 3} n^{\log 3})$.

Algoritmi di Monte Carlo

Esempio per PRIMALITÀ

Osservazioni

- Se `Fermat(`n`)` risponde **falso**, non commette errore.
- Non dà inoltre nessuna indicazione circa la composizione di n. Sembra che il problema di FATTORIZZAZIONE sia molto più difficile di PRIMALITÀ.
- Inciso: i sistemi crittografici RSA necessitano di risolvere PRIMALITÀ in modo efficiente, ma fondano la loro sicurezza sulla difficoltà del problema FATTORIZZAZIONE.

Algoritmi di Monte Carlo

Esempio per PRIMALITÀ

Osservazioni

- Che dire quando `Fermat(`n`)` risponde **vero**?
- Non è sufficiente per dire che n è primo! Se lo fosse... il teorema avrebbe un'altra forma!
- Per esempio: $4^{14} \equiv 1 \pmod{15}$ anche se 15 non è primo.
- Un intero $2 \leq a \leq n-2$ tale che $a^{n-1} \equiv 1 \pmod{n}$ per un numero composto n è detto testimone falso di primalità

Algoritmi di Monte Carlo

Esempio per PRIMALITÀ

Osservazioni

- Buona notizia: i falsi testimoni sono rari! Sono circa il 3.3% quando $n < 1000$
- Cattiva notizia: ci sono numeri non primi che ammettono un numero significativo di falsi testimoni.
- Per esempio: 516 ammette 318 valori falsi testimoni.
- Più in generale, per qualsiasi $\delta > 0$ ci sono infiniti numeri non primi per i quali il test di Fermat scopre la non primalità con probabilità $< \delta$.
- Quindi non è possibile abbassare arbitrariamente l'errore dell'algoritmo.

Algoritmi di Monte Carlo

Esempio per PRIMALITÀ

Altra idea avuta da Miller e Rabin: eliminare i falsi testimoni

- Si sfrutta la proprietà: se p è primo, allora $a^{\frac{p-1}{2}} \equiv \pm 1 \mod p$. Nota: $(-1 == n-1)$
- Sia $n > 4$ dispari e siano s e t due interi tali che $n-1 = 2^s t$ con t dispari: s deve essere maggiore di 0 perché $n-1$ è pari.
- Sia $B(n)$ un insieme così definito: $a \in B(n)$ se e solo se $2 \leq a \leq n-2$ e
  - $a^t \equiv 1 \pmod n$ o
  - $\exists i, 0 \leq i < s$ tale che $a^{2^i t} \equiv n-1 \pmod n$
- Difficile da definire, ma facile da calcolare!
- $B(n)$ cattura i testimoni della primalità di n eliminando parte dei falsi testimoni!

Algoritmi di Monte Carlo

Esempio per PRIMALITÀ

Altra idea avuta da Miller e Rabin: eliminare i falsi testimoni

Si dimostra che dato $n > 4$:

- se n è primo, allora $B(n) = \{a \mid 2 \leq a \leq n-2\}$
- se n non è primo, allora $|B(n)| \leq (n-9)/4$

Algoritmo conseguente

Si sceglie a tra $2 \leq a \leq n-2$ e si determina se $a \in B(n)$:

- se $a \notin B(n)$, si risponde **falso** senza errori.
- se $a \in B(n)$, si risponde **vero**.

Se n non è primo, si commette errore: probabilità $\leq$ ¼.

Nota!

Non è un algoritmo Monte Carlo classico per problemi di decisione: qui risposta negativa è certa, risposta positiva ha un margine di errore. Alcuni autori preferiscono riformulare l'algoritmo per il problema complementare: COMPOSTO

Algoritmi di Monte Carlo

Esempio per PRIMALITÀ

Funzione `MillerRabin`(n, k)

// $n > 4$ e dispari, k numero di test da eseguire

1: **for** $i = 1; i \leq k; i{+}{+}$ **do**
2: $a = \texttt{uniform}(2, n-2)$; // valore casuale uniforme
3: **if** $a \notin B(n)$ **then return** ***falso***;
4: **endfor**
5: **return** ***vero***;

Nota!

L'errore di fornire una risposta errata è 4^{-k}.
Già con $k = 10$ si riduce la probabilità di errore a meno di uno su un milione! Si dice `MillerRabin`(n, k) è un algoritmo Monte Carlo $(1 - 4^{-k})$-corretto per PRIMALITÀ.

Algoritmi di Monte Carlo

Esempio per PRIMALITÀ

Una nota sulla complessità

- Senza entrare nel dettaglio, ogni test $a \notin B(n)$ richiede 1 elevazione a potenza in modulo con esponente t e $s-1$ elevazioni a quadrato in modulo.
- Un'elevazione a potenza in modulo con esponente t costa $O(\log t)$ elevazioni al quadrato e moltiplicazioni in modulo.
- Contando elevazioni a quadrato come moltiplicazioni in modulo, il costo di $a \notin B(n)$ è $O(\log n)$ moltiplicazioni in modulo. Ogni moltiplicazione in modulo costa $O((\log n)^2)$.
- Se si vuole un errore $< \varepsilon$, allora $k \geq \lceil -\frac{\log \varepsilon}{2} \rceil$.
- La complessità è quindi $O((\log n)^3 \log \frac{1}{\varepsilon})$.
- Se $n \approx 2^{512}$ e $\varepsilon = 1/100$, si ha $512^3 4 \approx 5,36 \cdot 10^8$ operazioni. Se ogni operazione costa 65ps, il tempo finale è pari a 34ms c.

Algoritmi di Las Vegas

Definizione

Definizione 5 (Algoritmo probabilistico Las Vegas)

Un algoritmo probabilistico è un algoritmo di Las Vegas se, quando restituisce una soluzione, questa è corretta.

Nota!

Se durante una computazione non è possibile determinare una risposta esatta (a causa di scelte casuali, ad esempio), l'algoritmo deve terminare restituendo **nulla**.

Ci sono autori che indicano che l'algoritmo deve riniziare la computazione e continuare fino a quando la risposta esatta non è determinata.

Algoritmi di Las Vegas

Quando il problema è di decisione

Solitamente un algoritmo di Las Vegas è costruito come combinazione di due algoritmi di Monte Carlo:

- Alg_1 è un algoritmo di Monte Carlo per il problema di decisione Π con probabilità di errore p;
- Alg_2 è un algoritmo di Monte Carlo per il problema negato di Π (le risposte sono opposte a quelle di Π);
- ogni istanza di Π viene risolta sia da Alg_1 sia da Alg_2;
- la soluzione viene fornita confrontando le risposte ottenute:

	SI	NO
SI	non poss.	**NO**
NO	**SI**	dubbio

La prima riga indica le risposte dell'algoritmo Alg_1, la prima colonna quelle di Alg_2 mentre le celle interne indicano il comportamento dell'algoritmo. Le risposte in neretto sono certe.

Algoritmi di Las Vegas

Esempi di algoritmi

Problemi già visti e risolti con algoritmi Las Vegas

Gli algoritmi per i seguenti problemi sono particolari esempi di algoritmi Las Vegas:

- MEDIANA: usa un elemento di casualità per guidare la scelta dell'elemento v.
  Si dimostra che se la scelta non è ottimale, la successiva, in media, è.
- ORDINAMENTO: Quicksort usa lo stesso meccanismo usato per risolvere MEDIANA. La scelta casuale dell'elemento pivot permette di ridurre la differenza, in termini di confronti, tra buone e cattive istanze.

Algoritmi di Las Vegas

Applicazione tipica

La maggioranza degli algoritmi Las Vegas vengono impiegati nei casi in cui:

- Esiste un algoritmo deterministico che è efficiente nella stragrande maggioranza delle istanze ed estremamente inefficiente in alcune;
- È possibile riusare l'algoritmo deterministico introducendo la casualità in modo che il nuovo algoritmo risulti un po' meno efficiente nella stragrande maggioranza delle istanze e abbastanza efficiente in alcune;
- Le performance medie dei due algoritmi non sono diverse, ma la varianza è molto più piccola nell'algoritmo probabilistico.

Algoritmi di Las Vegas

Esempio per UNIVERSAL HASHING

Tabella hash

La struttura dati *tabella hash* permette di accedere a dati identificati da chiavi in un tempo medio costante.

Nota!

Il tempo di accesso è *costante* ($\Theta(\alpha + 1)$, dove α è indice di carico) se si ammette una distribuzione uniforme delle chiavi e si sceglie una buona funzione hash.

Notazione

- $U =$ universo degli indici possibili per le chiavi;
- $B = \{0, 1, \ldots, m-1\}$ insieme delle posizioni di una tabella hash;
- $h : U \to B$ è una funzione hash se è f. suriettiva;
- h è detta buona quando $Pr[h(x) = h(y)] = 1/m$ quando $x, y \in U \wedge x \neq y$.

Algoritmi di Las Vegas

Esempio per UNIVERSAL HASHING

Problema con le tabelle hash

- L'assunzione di distribuzione uniforme delle chiavi può essere eccessiva.
- In un compilatore, ad esempio, gli identificatori non hanno nomi con una struttura tale da pensare che una funzione `hash` possa determinare chiavi distribuite uniformemente ad ogni esecuzione!

Possibile miglioramento: Universal hashing

- Se si scegliesse a caso una buona funzione hash all'inizio di ogni esecuzione di programma, le prestazioni medie del programma migliorerebbero!
- Infatti: sequenze di chiavi che con una funzione hash portano al caso peggiore, in una successiva esecuzione possono portare ad un'esecuzione migliore!

Algoritmi di Las Vegas

Esempio per UNIVERSAL HASHING

Definizione 6 (Insieme funzioni hash universali)

Sia $\mathcal{H}$ un'insieme di funzioni hash tra U e $B = \{0, 1, \ldots, m-1\}$ tali che per ciascuna coppia $x, y \in U, x \neq y$,

$$|\{h \mid h \in \mathcal{H} \wedge h(x) = h(y)\}| = |\mathcal{H}|/m$$

Nota!

- $\mathcal{H}$ è formato da funzioni hash buone diverse tra loro in modo che al più $1/m$ di loro possono mappare nello stesso modo chiavi diverse per qualsiasi coppia di chiavi.
- Alcuni autori chiamano questo insieme Universal$_2$.

Algoritmi di Las Vegas

Esempio per UNIVERSAL HASHING

Esempio 2 (Classe doppio hash)

Siano $U = \{0, 1, 2, \ldots, a-1\}$, $B = \{0, 1, \ldots, m-1\}$ e $p \geq a$ un numero primo.

Una classe universale di funzioni è data:

$$H = \{h_{ij} | 1 \leq i < p \wedge 0 \leq j < p\}$$

dove $h_{ij}(x) = ((ix + j) \mod p) \mod m$.

Algoritmi di Las Vegas

Esempio per UNIVERSAL HASHING

Esempio 3 (Classe composizione hash)

Siano m un numero primo. Si decompone ogni chiave $x =< x_0, \ldots, x_r >$ in $r + 1 < m$ pezzi.
Sia $a =< a_0, \ldots, a_r >$ una sequenza di $r + 1$ elementi scelti in $\{0, 1, \ldots, m - 1\}$.
Una classe universale di funzioni è data dalla classe:

$$H = \left\{ h_a \mid h_a(x) = \sum_{i=0}^{r} a_i x_i \mod m \wedge a \in \{0, 1, \ldots, m - 1\} \right\}$$

Algoritmi probabilistici

Esercizi

Esercizio 2

Data una moneta non bilanciata in cui la probabilità che esca testa in un lancio è il valore p non noto e i lanci non si influiscono, si costruisca un semplice algoritmo che generi sequenze bilanciate di bit casuali usando tale moneta.

Algoritmi probabilistici

Esercizi

Esercizio 3

Si dimostra che è sufficiente lanciare 1 500 000 aghi per stimare π con una precisione di 0.01 al 95% di confidenza. Si dimostri che è possibile "migliorare" l'algoritmo lanciando aghi con lunghezza doppia in quanto questo permette di determinare $\frac{n}{2k}$ come stima di π. Quanti aghi di questo tipo è necessario lanciare per stimare π con la medesima qualità di cui sopra?

Esercizio 4

Si scriva un programma che simuli l'esperimento di Buffon avendo come input la precisione e la confidenza. Si ricorda che non è possibile usare il valore π nella stima di n.

Algoritmi probabilistici

Esercizi

Esercizio 5

Nei lucidi è stato presentato un algoritmo Monte Carlo per il test di primalità che risponde sempre correttamente quando l'input è un numero primo e correttamente con probabilità ¾ quando l'input è composto in un tempo $O((\log n)^3)$.
Si determini un algoritmo di Monte Carlo che risponda in modo sempre corretto quando l'input è un numero composto e in modo corretto con probabilità almeno ½ quando l'input è primo. L'algoritmo deve operare in tempo $O((\log n)^k)$ per una qualche costante k.
Questo esercizio richiede di cercare altri metodi non spiegati a lezione! È un esercizio di ricerca e studio individuale.

Algoritmi probabilistici

Esercizi

Esercizio 6

Si dimostri che il test di primalità può essere risolto da un algoritmo di Las Vegas in tempo $O((\log n)^k)$ per una qualche costante k.