Requisiti per la cifratura

Gregorio D'Agostino

8 Aprile 2021

Primalità

- In molte applicazioni crittografiche (abbiamo visto RSA) bisogna trovare numeri primi molto grandi. Quindi dobbiamo disporre di metodi per verificare la primalità di un numero.

Primalità

- In molte applicazioni crittografiche (abbiamo visto RSA) bisogna trovare numeri primi molto grandi. Quindi dobbiamo disporre di metodi per verificare la primalità di un numero.
- I numeri primi devono essere presi a caso altrimenti l'attaccante si concentra sul nostro metodo per trovare i primi e lo spazio delle chiavi diviene molto meno esteso. In altre parole la cifratura diviene più debole.

Primalità

- In molte applicazioni crittografiche (abbiamo visto RSA) bisogna trovare numeri primi molto grandi. Quindi dobbiamo disporre di metodi per verificare la primalità di un numero.
- I numeri primi devono essere presi a caso altrimenti l'attaccante si concentra sul nostro metodo per trovare i primi e lo spazio delle chiavi diviene molto meno esteso. In altre parole la cifratura diviene più debole.
- Il tempo per trovare i numeri primi necessari deve essere "ragionevole".
  Come detto molte volte il tempo stimato per la decrittazione deve essere maggiore del periodo in cui si intende mantenere la confidenzialità valutate le risorse di calcolo dell'attaccante.
  Analogamente il tempo per definire le chiavi della cifratura deve essere sufficientemente breve anche se le chiavi possono essere utilizzate reiteratamente.

Metodo dell'ispezione

- Algoritmo: Dato un numero n per verificare se è primo, lo si divide per tutti i numeri minori della sua radice quadrata approssimata per difetto.

$$\forall a < \sqrt{n} : \ mod(n, a) \neq 0 \Rightarrow n \ primo.$$

Metodo dell'ispezione

- Algoritmo: Dato un numero n per verificare se è primo, lo si divide per tutti i numeri minori della sua radice quadrata approssimata per difetto.

$$\forall a < \sqrt{n} : \ mod(n, a) \neq 0 \Rightarrow n \ primo.$$

- Numero di operazioni (resti) necessarie per il test: $\sim \sqrt{n}$

Metodo dell'ispezione

- Algoritmo: Dato un numero n per verificare se è primo, lo si divide per tutti i numeri minori della sua radice quadrata approssimata per difetto.

$$\forall a < \sqrt{n} : \ mod(n, a) \neq 0 \Rightarrow n \ primo.$$

- Numero di operazioni (resti) necessarie per il test: $\sim \sqrt{n}$
- Memoria necessaria per il test: 3 interi di precisione pari alla lunghezza di n.

Miglioramenti del metodo della ispezione

- Se si considerano solo i numeri pari (ed il 2) si dimezza il numero di divisioni (resti) e si aggiungono $\sqrt{n}$ moltiplicazioni.
  Si verificano solo i numeri della forma $2k + 1$. Il tempo di calcolo si dimezza ($\sim \sqrt{n}(1 - 1/2) = 1/2\sqrt{n}$).
  Memoria necessaria per il test: 3 interi di precisione pari alla lunghezza di n.

Miglioramenti del metodo della ispezione

- Se si considerano solo i numeri pari (ed il 2) si dimezza il numero di divisioni (resti) e si aggiungono $\sqrt{n}$ moltiplicazioni. Si verificano solo i numeri della forma $2k+1$. Il tempo di calcolo si dimezza ($\sim \sqrt{n}(1-1/2) = 1/2\sqrt{n}$).
  Memoria necessaria per il test: 3 interi di precisione pari alla lunghezza di n.
- Se si escludono anche i multipli di 3 il tempo di calcolo diviene circa un terzo ($\sqrt{n}(1-1/2)(1-1/3) = 1/3\sqrt{n}$). Si verificano solo i numeri della forma $6k \pm 1$

Miglioramenti del metodo della ispezione

- Se si considerano solo i numeri pari (ed il 2) si dimezza il numero di divisioni (resti) e si aggiungono $\sqrt{n}$ moltiplicazioni. Si verificano solo i numeri della forma $2k+1$. Il tempo di calcolo si dimezza ($\sim \sqrt{n}(1-1/2) = 1/2\sqrt{n}$).
  Memoria necessaria per il test: 3 interi di precisione pari alla lunghezza di n.
- Se si escludono anche i multipli di 3 il tempo di calcolo diviene circa un terzo ($\sqrt{n}(1-1/2)(1-1/3) = 1/3\sqrt{n}$). Si verificano solo i numeri della forma $6k \pm 1$
- Se si escludono anche i multipli di 5 il tempo di calcolo diviene circa un quarto ($\sqrt{n}(1-1/2)(1-1/3)(1-1/5) = 4/15\sqrt{n}$).
  Memoria necessaria per il test: 6 interi di precisione pari alla lunghezza di n.

Miglioramenti del metodo della ispezione

- Se si considerano solo i numeri pari (ed il 2) si dimezza il numero di divisioni (resti) e si aggiungono $\sqrt{n}$ moltiplicazioni. Si verificano solo i numeri della forma $2k+1$. Il tempo di calcolo si dimezza ($\sim \sqrt{n}(1-1/2) = 1/2\sqrt{n}$).
  Memoria necessaria per il test: 3 interi di precisione pari alla lunghezza di n.
- Se si escludono anche i multipli di 3 il tempo di calcolo diviene circa un terzo ($\sqrt{n}(1-1/2)(1-1/3) = 1/3\sqrt{n}$). Si verificano solo i numeri della forma $6k \pm 1$
- Se si escludono anche i multipli di 5 il tempo di calcolo diviene circa un quarto ($\sqrt{n}(1-1/2)(1-1/3)(1-1/5) = 4/15\sqrt{n}$).
  Memoria necessaria per il test: 6 interi di precisione pari alla lunghezza di n.

Miglioramenti del metodo della ispezione

- Se si considerano solo i numeri pari (ed il 2) si dimezza il numero di divisioni (resti) e si aggiungono $\sqrt{n}$ moltiplicazioni. Si verificano solo i numeri della forma $2k+1$. Il tempo di calcolo si dimezza ($\sim \sqrt{n}(1-1/2) = 1/2\sqrt{n}$).
  Memoria necessaria per il test: 3 interi di precisione pari alla lunghezza di n.
- Se si escludono anche i multipli di 3 il tempo di calcolo diviene circa un terzo ($\sqrt{n}(1-1/2)(1-1/3) = 1/3\sqrt{n}$). Si verificano solo i numeri della forma $6k \pm 1$
- Se si escludono anche i multipli di 5 il tempo di calcolo diviene circa un quarto ($\sqrt{n}(1-1/2)(1-1/3)(1-1/5) = 4/15\sqrt{n}$).
  Memoria necessaria per il test: 6 interi di precisione pari alla lunghezza di n.
- L'idea si può estendere eliminando i multipli dei primi r primi. Il tempo scende e la memoria sale.

Miglioramenti del metodo della ispezione

- Se si considerano solo i numeri pari (ed il 2) si dimezza il numero di divisioni (resti) e si aggiungono $\sqrt{n}$ moltiplicazioni. Si verificano solo i numeri della forma $2k+1$. Il tempo di calcolo si dimezza ($\sim \sqrt{n}(1-1/2) = 1/2\sqrt{n}$).
  Memoria necessaria per il test: 3 interi di precisione pari alla lunghezza di n.
- Se si escludono anche i multipli di 3 il tempo di calcolo diviene circa un terzo ($\sqrt{n}(1-1/2)(1-1/3) = 1/3\sqrt{n}$). Si verificano solo i numeri della forma $6k \pm 1$
- Se si escludono anche i multipli di 5 il tempo di calcolo diviene circa un quarto ($\sqrt{n}(1-1/2)(1-1/3)(1-1/5) = 4/15\sqrt{n}$).
  Memoria necessaria per il test: 6 interi di precisione pari alla lunghezza di n.
- L'idea si può estendere eliminando i multipli dei primi r primi. Il tempo scende e la memoria sale.
- Esercizio: scrivere codice Octave che attui il metodo dell'ispezione.

Il crivello di Eratostene

- Il crivello di Eratostene è un algoritmo che consente di trovare tutti i primi minori della radice quadrata di un numero assegnato n e verificarne la primalità.

Il crivello di Eratostene

- Il crivello di Eratostene è un algoritmo che consente di trovare tutti i primi minori della radice quadrata di un numero assegnato n e verificarne la primalità.
- Il metodo consiste nel cancellare da una lista (contenente i numeri fino a $\sqrt{n}$) tutti i multipli dei numeri primi già scoperti iniziando dal 2. Ad ogni iterazione il primo numero non cancellato è primo. Per verificarne la primalità basta dividere n solo per i sopravvissuti.

Il crivello di Eratostene

- I crivello di Eratostene è un algoritmo che consente di trovare tutti i primi minori della radice quadrata di un numero assegnato n e verificarne la primalità.
- Il metodo consiste nel cancellare da una lista (contenente i numeri fino a $\sqrt{n}$) tutti i multipli dei numeri primi già scoperti iniziando dal 2. Ad ogni iterazione il primo numero non cancellato è primo. Per verificarne la primalità basta dividere n solo per i sopravvissuti.
- Quando l'algoritmo è utilizzato solo per verificare la primalità del numero n, si interrompe non appena uno dei primi trovati divide n. L'algoritmo in questo caso è l'estensione naturale dei precedenti.

Il crivello di Eratostene

- I crivello di Eratostene è un algoritmo che consente di trovare tutti i primi minori della radice quadrata di un numero assegnato n e verificarne la primalità.
- Il metodo consiste nel cancellare da una lista (contenente i numeri fino a $\sqrt{n}$) tutti i multipli dei numeri primi già scoperti iniziando dal 2. Ad ogni iterazione il primo numero non cancellato è primo. Per verificarne la primalità basta dividere n solo per i sopravvissuti.
- Quando l'algoritmo è utilizzato solo per verificare la primalità del numero n, si interrompe non appena uno dei primi trovati divide n. L'algoritmo in questo caso è l'estensione naturale dei precedenti.
- Per trovare il numeri primi minori di $\sqrt{n}$ la memoria richiesta è pari a $\sqrt{n}$, ma il tempo si riduce: le divisioni (i resti) sono assenti e si compiono circa $\sqrt{n}$ moltiplicazioni.

Il crivello di Eratostene

- I crivello di Eratostene è un algoritmo che consente di trovare tutti i primi minori della radice quadrata di un numero assegnato n e verificarne la primalità.
- Il metodo consiste nel cancellare da una lista (contenente i numeri fino a $\sqrt{n}$) tutti i multipli dei numeri primi già scoperti iniziando dal 2. Ad ogni iterazione il primo numero non cancellato è primo. Per verificarne la primalità basta dividere n solo per i sopravvissuti.
- Quando l'algoritmo è utilizzato solo per verificare la primalità del numero n, si interrompe non appena uno dei primi trovati divide n. L'algoritmo in questo caso è l'estensione naturale dei precedenti.
- Per trovare il numeri primi minori di $\sqrt{n}$ la memoria richiesta è pari a $\sqrt{n}$, ma il tempo si riduce: le divisioni (i resti) sono assenti e si compiono circa $\sqrt{n}$ moltiplicazioni.
- Per verificare la primalità di n si compiono al più $\pi(\sqrt{n})$ resti. Con il simbolo $\pi(k)$ si indica il numero dei numeri primi minori di k.

Esercizi sul crivello di Eratostene

- Scrivere un codice di Octave che realizzi il crivello di Eratostene.

Esercizi sul crivello di Eratostene

- Scrivere un codice di Octave che realizzi il crivello di Eratostene.
- Calcolare i primi 100 primi.

Esercizi sul crivello di Eratostene

- Scrivere un codice di Octave che realizzi il crivello di Eratostene.
- Calcolare i primi 100 primi.
- Gli esercizi con octave non sono vincolanti per il corso da 6 crediti, sono consigliati per chi volesse approfondire. Sul sito Gordion si trovano molte delle soluzioni.

Quanto sono densi i numeri primi

- Abbiamo già dimostrato che sono infiniti. Quindi:

$$\lim_{n\to\infty} \pi(n) = \infty.$$

Quanto sono densi i numeri primi

- Abbiamo già dimostrato che sono infiniti. Quindi:

$$\lim_{n\to\infty} \pi(n) = \infty.$$

- La funzione $\pi(n)$ è monotona crescente. Con quale velocità va all'infinito?

Quanto sono densi i numeri primi

- Abbiamo già dimostrato che sono infiniti. Quindi:

$$\lim_{n\to\infty} \pi(n) = \infty.$$

- La funzione $\pi(n)$ è monotona crescente. Con quale velocità va all'infinito?
- Il teorema dei numeri primi (attribuito ad Hadamard e a de la Vallée-Poussin) dimostra che si comporta come $\pi(n) \sim n/log(n)$, cioè:

$$\lim_{n\to\infty} \frac{\pi(n)}{n/log(n)} = 1.$$

Quanto sono densi i numeri primi

- Abbiamo già dimostrato che sono infiniti. Quindi:

$$\lim_{n\to\infty} \pi(n) = \infty.$$

- La funzione $\pi(n)$ è monotona crescente. Con quale velocità va all'infinito?
- Il teorema dei numeri primi (attribuito ad Hadamard e a de la Vallée-Poussin) dimostra che si comporta come $\pi(n) \sim n/log(n)$, cioè:

$$\lim_{n\to\infty} \frac{\pi(n)}{n/log(n)} = 1.$$

- Se chiamiamo p_n l'n-esimo numero primo, una formulazione equivalente è:

$$p_n \sim n \cdot log(n).$$

Quanto sono densi i numeri primi - cont

- Gauss ha congetturato che $\pi(n)$ si comporti come la funzione Li:

$$Li(n) - Li(2) \stackrel{def}{=} \int_2^n \frac{dx}{log(x)}.$$

Quanto sono densi i numeri primi - cont

- Gauss ha congetturato che $\pi(n)$ si comporti come la funzione Li:
$$Li(n) - Li(2) \stackrel{def}{=} \int_2^n \frac{dx}{log(x)}.$$
- In questo caso la derivata (cioè la densità) sarebbe proprio $1/log(n)$. Ma questa ipotesi non è stata mai dimostrata, nè confutata.

Quanto sono densi i numeri primi - cont

- Gauss ha congetturato che $\pi(n)$ si comporti come la funzione Li:
$$Li(n) - Li(2) \stackrel{def}{=} \int_2^n \frac{dx}{log(x)}.$$
- In questo caso la derivata (cioè la densità) sarebbe proprio $1/log(n)$. Ma questa ipotesi non è stata mai dimostrata, nè confutata.
- Nell'intervallo dei numeri considerati per gli attuali livelli di sicurezza in crittografia, l'ipotesi di Gauss è ampiamente verificata.
Ai fini della stima dei tempi di ricerca dei numeri primi la stima di Hadamar $\rho \sim \frac{1)}{log(n)}$ è sufficiente.

Conseguenze per la criptografia

- La densità dei numeri primi va con l'inverso del logaritmo di n. Quindi più è grande n e più è difficile trovare dei numeri primi scegliendo a caso.

$$\rho(n) \stackrel{def}{=} \frac{\pi(n)}{n} \sim \frac{1}{log(n)}.$$

Conseguenze per la criptografia

- La densità dei numeri primi va con l'inverso del logaritmo di n. Quindi più è grande n e più è difficile trovare dei numeri primi scegliendo a caso.

$$\rho(n) \stackrel{def}{=} \frac{\pi(n)}{n} \sim \frac{1}{log(n)}.$$

- Questo significa che se cerchiamo chiavi più complesse per la cifratura RSA dobbiamo spendere un tempo più lungo o disporre di una piattaforma di calcolo più potente.

Conseguenze per la criptografia

- La densità dei numeri primi va con l'inverso del logaritmo di n. Quindi più è grande n e più è difficile trovare dei numeri primi scegliendo a caso.

$$\rho(n) \stackrel{def}{=} \frac{\pi(n)}{n} \sim \frac{1}{log(n)}.$$

- Questo significa che se cerchiamo chiavi più complesse per la cifratura RSA dobbiamo spendere un tempo più lungo o disporre di una piattaforma di calcolo più potente.
- Come già detto l'alternativa di utilizzare metodi non uniformi per scegliere i numeri primi da utilizzare come chiavi riduce lo spazio delle chiavi e rende la cifratura meno robusta.

Metodi di decrittazione

- La principale classificazione riguarda il determinismo e la strategia per individuare la chiave.

Metodi di decrittazione

- La principale classificazione riguarda il determinismo e la strategia per individuare la chiave.
- Nel caso di RSA, le strategie principali sono basate sulla decomposizione del modulo $n = p \cdot q$ oppure sulla ricerca diretta della chiave privata α.

Metodi di decrittazione

- La principale classificazione riguarda il determinismo e la strategia per individuare la chiave.
- Nel caso di RSA, le strategie principali sono basate sulla decomposizione del modulo $n = p \cdot q$ oppure sulla ricerca diretta della chiave privata α.
- Alcuni algoritmi sono deterministici e ipotizzano una scelta casuale dei valori di p e q, mentre altri ipotizzano metodi diversi per la scelta dei due primi.

Il Metodo di Fermat

- Si basa sul principio che se un numero non è primo, è esprimibile in forma di prodotto di due numeri a e b (non necessariamente primi):

$$n = a \cdot b.$$

Il Metodo di Fermat

- Si basa sul principio che se un numero non è primo, è esprimibile in forma di prodotto di due numeri a e b (non necessariamente primi):

$$n = a \cdot b.$$

- Trovare a e b equivale a trovare le loro semisomma (s) e semidifferenza (d):

$$\begin{cases} s & \stackrel{def}{=} & \frac{a+b}{2}, \\ d & \stackrel{def}{=} & \frac{a-b}{2}; \end{cases}$$

che si inverte in:

$$\begin{cases} a & = & s+d, \\ b & = & s-d. \end{cases}$$

Il Metodo di Fermat

- Si basa sul principio che se un numero non è primo, è esprimibile in forma di prodotto di due numeri a e b (non necessariamente primi):

$$n = a \cdot b.$$

- Trovare a e b equivale a trovare le loro semisomma (s) e semidifferenza (d):

$$\begin{cases} s & \stackrel{def}{=} & \frac{a+b}{2}, \\ d & \stackrel{def}{=} & \frac{a-b}{2}; \end{cases}$$

che si inverte in:

$$\begin{cases} a & = & s+d, \\ b & = & s-d. \end{cases}$$

- La differenza dei quadrati di s e d eguaglia n:

$$s^2 - d^2 = 1/4[(a+b)^2 - (a-b)^2] = 1/4[4ab] = ab = n.$$

Algoritmo di Fermat

- L'agoritmo di Fermat, parte dal numero k_0: il più piccolo numero il cui quadrato supera n:

$$\begin{cases} k_0^2 & > & n, \\ (k_0 - 1)^2 & < & n; \end{cases}$$

e calcola la differenza del quadrato rispetto ad n $(k^2 - n)$.

Algoritmo di Fermat

- L'agoritmo di Fermat, parte dal numero k_0: il più piccolo numero il cui quadrato supera n:

$$\begin{cases} k_0^2 & > & n, \\ (k_0 - 1)^2 & < & n; \end{cases}$$

  e calcola la differenza del quadrato rispetto ad n $(k^2 - n)$.
- Ad ogni passo si incrementa di uno il numero k $(k \to k + 1)$ e la procedura si arresta quando $k^2 - n$ è un quadrato perfetto e quindi possiamo identificare k con s e $k^2 - n$ con d^2.
- $p = k + \sqrt{k^2 - n}$, $q = k - \sqrt{k^2 - n}$

Algoritmo di Fermat

- L'agoritmo di Fermat, parte dal numero k_0: il più piccolo numero il cui quadrato supera n:

$$\begin{cases} k_0^2 & > & n, \\ (k_0 - 1)^2 & < & n; \end{cases}$$

  e calcola la differenza del quadrato rispetto ad n $(k^2 - n)$.
- Ad ogni passo si incrementa di uno il numero k $(k \to k + 1)$ e la procedura si arresta quando $k^2 - n$ è un quadrato perfetto e quindi possiamo identificare k con s e $k^2 - n$ con d^2.
- $p = k + \sqrt{k^2 - n}$, $q = k - \sqrt{k^2 - n}$
- Esercizio: Scrivere un codice Octave che realizza l'algoritmo di Fermat.

Algoritmo di Fermat

- L'agoritmo di Fermat, parte dal numero k_0: il più piccolo numero il cui quadrato supera n:

$$\begin{cases} k_0^2 & > & n, \\ (k_0 - 1)^2 & < & n; \end{cases}$$

  e calcola la differenza del quadrato rispetto ad n $(k^2 - n)$.
- Ad ogni passo si incrementa di uno il numero k $(k \to k + 1)$ e la procedura si arresta quando $k^2 - n$ è un quadrato perfetto e quindi possiamo identificare k con s e $k^2 - n$ con d^2.
- $p = k + \sqrt{k^2 - n}$, $q = k - \sqrt{k^2 - n}$
- Esercizio: Scrivere un codice Octave che realizza l'algoritmo di Fermat.
- L'algoritmo richiede pochissima memoria (5 interi) e un numero di passi che dipende dalla differenza d.

Conseguenze dell'esistenza dell'algoritmo di Fermat sulla cifratura RSA

- Se nell'algoritmo di RSA scegliamo due fattori primi (p e q) vicini tra loro, d sarà piccola e quindi l'algoritmo di Fermat fattorizzerà n e si arresterà in pochi passi.

Conseguenze dell'esistenza dell'algoritmo di Fermat sulla cifratura RSA

- Se nell'algoritmo di RSA scegliamo due fattori primi (p e q) vicini tra loro, d sarà piccola e quindi l'algoritmo di Fermat fattorizzerà n e si arresterà in pochi passi.
- Se invece scegliamo due numeri primi troppo diversi tra loro allora uno dei due sarà piccolo ed il metodo dell'ispezione diretta si arresterà in pochi passi.

Conseguenze dell'esistenza dell'algoritmo di Fermat sulla cifratura RSA

- Se nell'algoritmo di RSA scegliamo due fattori primi (p e q) vicini tra loro, d sarà piccola e quindi l'algoritmo di Fermat fattorizzerà n e si arresterà in pochi passi.
- Se invece scegliamo due numeri primi troppo diversi tra loro allora uno dei due sarà piccolo ed il metodo dell'ispezione diretta si arresterà in pochi passi.
- Il cifratore deve prevenire entrambi i possibili attacchi. Ne consegue che la differenza e la somma dei due numeri primi (p e q) devono essere dello stesso ordine di grandezza. La cardinalità di questo spazio definirà la complessità delle chiavi. In pratica basta limitare la scelta di p (e quindi q) ad esempio all'intervallo $(1/2\sqrt{n}, 3/4\sqrt{n})$.

Variazioni sulla cifratura RSA

- Una delle idee più naturali per complicare RSA consiste nell'usare una fattorizzazione a tre o più primi.

$$n = p_1 p_2 p_3.$$

Variazioni sulla cifratura RSA

- Una delle idee più naturali per complicare RSA consiste nell'usare una fattorizzazione a tre o più primi.

$$n = p_1 p_2 p_3.$$

- In realtà, la situazione peggiorerebbe per il cifratore, perché il minore dei fattori sarebbe necessariamente più piccolo e quindi basterebbe applicare il crivello di Eratostene a tutti i numeri minori della radice cubica di n. Il metodo dell'ispezione risulterebbe notevolmente semplificato.

Variazioni sulla cifratura RSA

- Una delle idee più naturali per complicare RSA consiste nell'usare una fattorizzazione a tre o più primi.

$$n = p_1 p_2 p_3.$$

- In realtà, la situazione peggiorerebbe per il cifratore, perché il minore dei fattori sarebbe necessariamente più piccolo e quindi basterebbe applicare il crivello di Eratostene a tutti i numeri minori della radice cubica di n. Il metodo dell'ispezione risulterebbe notevolmente semplificato.
- Inoltre anche la minima differenza tra due fattori sarebbe più piccola, accelerando quindi convergenza del metodo di Fermat. Anche in questo caso basterebbe studiare i k minori di $k_0 + n^{1/3}$.

Metodi indeterminati

- I metodi precedenti sono algoritmici e consentono di determinare la primalità del numero n con certezza e consentono anche di determinare almeno una fattorizzazione.

Metodi indeterminati

- I metodi precedenti sono algoritmici e consentono di determinare la primalità del numero n con certezza e consentono anche di determinare almeno una fattorizzazione.
- Esistono dei criteri che non risolvono con certezza il problema della primalità, ma che consentono in alcuni casi di escluderla.

Metodi indeterminati

- I metodi precedenti sono algoritmici e consentono di determinare la primalità del numero n con certezza e consentono anche di determinare almeno una fattorizzazione.
- Esistono dei criteri che non risolvono con certezza il problema della primalità, ma che consentono in alcuni casi di escluderla.
- Tali metodi sono basati su condizioni necessarie di primalità, ma non sufficienti.

Metodi indeterminati

- I metodi precedenti sono algoritmici e consentono di determinare la primalità del numero n con certezza e consentono anche di determinare almeno una fattorizzazione.
- Esistono dei criteri che non risolvono con certezza il problema della primalità, ma che consentono in alcuni casi di escluderla.
- Tali metodi sono basati su condizioni necessarie di primalità, ma non sufficienti.
- I numeri che soddisfano alcuni criteri necessari di primalità ma non suffienti di chiamano pseudo-primi

Metodi indeterminati

- I metodi precedenti sono algoritmici e consentono di determinare la primalità del numero n con certezza e consentono anche di determinare almeno una fattorizzazione.
- Esistono dei criteri che non risolvono con certezza il problema della primalità, ma che consentono in alcuni casi di escluderla.
- Tali metodi sono basati su condizioni necessarie di primalità, ma non sufficienti.
- I numeri che soddisfano alcuni criteri necessari di primalità ma non suffienti di chiamano pseudo-primi
- Il metodo (già visto) dalla primalità cinese è uno di questi: se $2^n \neq 1 \ (mod \ n)$ possiamo escludere che n sia primo, ma se la relazione è soddisfatta non possiamo esprimerci.

Metodi indeterminati

- I metodi precedenti sono algoritmici e consentono di determinare la primalità del numero n con certezza e consentono anche di determinare almeno una fattorizzazione.
- Esistono dei criteri che non risolvono con certezza il problema della primalità, ma che consentono in alcuni casi di escluderla.
- Tali metodi sono basati su condizioni necessarie di primalità, ma non sufficienti.
- I numeri che soddisfano alcuni criteri necessari di primalità ma non suffienti di chiamano pseudo-primi
- Il metodo (già visto) dalla primalità cinese è uno di questi: se $2^n \neq 1 \ (mod \ n)$ possiamo escludere che n sia primo, ma se la relazione è soddisfatta non possiamo esprimerci.
- Esercizio: scrivere un codice Octave che esegua il test di primalità cinese. [Basta partire da powers.m]

Criterio di Eulero

- Anche questo criterio può solo escludere la primalità, ma non garantirla.

Criterio di Eulero

- Anche questo criterio può solo escludere la primalità, ma non garantirla.
- Si basa sul piccolo teorema di Fermat.

Criterio di Eulero

- Anche questo criterio può solo escludere la primalità, ma non garantirla.
- Si basa sul piccolo teorema di Fermat.
- In realtà, se per tutti gli a (non solo il 2) $a^{n-1} = 1 \ (mod \ n)$ si può dire con certezza che n è primo [Bastano i numeri fino ad (n+1)/2].

Criterio di Eulero

- Anche questo criterio può solo escludere la primalità, ma non garantirla.
- Si basa sul piccolo teorema di Fermat.
- In realtà, se per tutti gli a (non solo il 2) $a^{n-1} = 1 \ (mod \ n)$ si può dire con certezza che n è primo [Bastano i numeri fino ad (n+1)/2].
- Nella realtà questo sarebbe molto più oneroso computazionalmente del metodo di Eratostene (e perfino dell'ispezione diretta) e quindi si limita il test ad un insieme finito di a. In tal modo il test garantisce una pseudoprimalità (può solo escludere la primalità se siamo fortunati). In pratica diviene solo più accurato del test cinese.

Criterio di Eulero

- Anche questo criterio può solo escludere la primalità, ma non garantirla.
- Si basa sul piccolo teorema di Fermat.
- In realtà, se per tutti gli a (non solo il 2) $a^{n-1} = 1 \ (mod \ n)$ si può dire con certezza che n è primo [Bastano i numeri fino ad (n+1)/2].
- Nella realtà questo sarebbe molto più oneroso computazionalmente del metodo di Eratostene (e perfino dell'ispezione diretta) e quindi si limita il test ad un insieme finito di a. In tal modo il test garantisce una pseudoprimalità (può solo escludere la primalità se siamo fortunati). In pratica diviene solo più accurato del test cinese.
- Per un' insieme selezionato di numeri $a \in S$, si verifica:

$$a^{\frac{n-1}{2}} = \pm 1 \ (mod \ n)$$

Criteri basati sul piccolo teorema di Fermat

- Si possono scegliere a caso dei numeri ed eseguire il test di Fermat su di essi. Chiameremo questo criterio "Ispezione casuale alla Fermat di dimensione m" (m numero di valori testati).

$$a^{n-1} \equiv 1 \ (\ mod\ n\ ).$$

Criteri basati sul piccolo teorema di Fermat

- Si possono scegliere a caso dei numeri ed eseguire il test di Fermat su di essi. Chiameremo questo criterio "Ispezione casuale alla Fermat di dimensione m" (m numero di valori testati).

$$a^{n-1} \equiv 1 \ (\ mod\ n\ ).$$

- I numeri di Carmichael resistono a qualunque test di Fermat (esclusi i divisori dello zero cioè di n). Per definizione un numero di Carmichael soddisfa le seguenti eguaglianze:

$$n : \forall a < n : a^n \equiv a \ (\ mod\ n\ ).$$

Criteri basati sul piccolo teorema di Fermat

- Si possono scegliere a caso dei numeri ed eseguire il test di Fermat su di essi. Chiameremo questo criterio "Ispezione casuale alla Fermat di dimensione m" (m numero di valori testati).

$$a^{n-1} \equiv 1 \ (\ mod\ n\ ).$$

- I numeri di Carmichael resistono a qualunque test di Fermat (esclusi i divisori dello zero cioè di n). Per definizione un numero di Carmichael soddisfa le seguenti eguaglianze:

$$n : \forall a < n : a^n \equiv a \ (\ mod\ n\ ).$$

- Il più piccolo numero di Carmichael è 561=3x11x17.

Criteri basati sul piccolo teorema di Fermat

- Si possono scegliere a caso dei numeri ed eseguire il test di Fermat su di essi. Chiameremo questo criterio "Ispezione casuale alla Fermat di dimensione m" (m numero di valori testati).

$$a^{n-1} \equiv 1 \ (\ mod\ n\ ).$$

- I numeri di Carmichael resistono a qualunque test di Fermat (esclusi i divisori dello zero cioè di n). Per definizione un numero di Carmichael soddisfa le seguenti eguaglianze:

$$n : \forall a < n : a^n \equiv a \ (\ mod\ n\ ).$$

- Il più piccolo numero di Carmichael è 561=3x11x17.
- Esercizio verificare con Octave che 341 è uno pseudo-primo cinese ma non è di Carmichael. Quanto fa 11^{31} e 31^{11} ? Che succede se a è un divisore di 341?

Criteri basati sulle radici quadrate dell'unità

- Esistono sempre due radici banali dell'unità:

$$\begin{cases} 1^2 & \equiv 1 \pmod{n}, \\ (n-1)^2 = n^2 - 2n + 1 & \equiv 1 \pmod{n}; \end{cases}$$

Criteri basati sulle radici quadrate dell'unità

- Esistono sempre due radici banali dell'unità:

$$\begin{cases} 1^2 & \equiv 1 \pmod{n}, \\ (n-1)^2 = n^2 - 2n + 1 & \equiv 1 \pmod{n}; \end{cases}$$

- $x^2 = 1$ equivale a $x^2 - 1 = 0$. Se n è primo, $\mathbb{Z}_n$ è un campo; $(x-1)(x+1) = 0$ implica che uno dei due fattori sia nullo e dunque le due soluzioni banali sono le uniche.

Criteri basati sulle radici quadrate dell'unità

- Esistono sempre due radici banali dell'unità:

$$\begin{cases} 1^2 & \equiv & 1 & (mod & n), \\ (n-1)^2 = n^2 - 2n + 1 & \equiv & 1 & (mod & n); \end{cases}$$

- $x^2 = 1$ equivale a $x^2 - 1 = 0$. Se n è primo, $\mathbb{Z}_n$ è un campo; $(x-1)(x+1) = 0$ implica che uno dei due fattori sia nullo e dunque le due soluzioni banali sono le uniche.
- Se n non è primo, possono esistere altre soluzioni. In particolare se $n = pq$ (con p e q primi). $x^2 = 1$ equivale (th. dei resti cinese) al sistema:

$$\begin{cases} x^2 & \equiv & 1 & (mod & p), \\ x^2 & \equiv & 1 & (mod & q); \end{cases}$$

che a sua volta equivale a

$$\begin{cases} x & \equiv & 1, p-1 & (mod & p), \\ x & \equiv & 1, q-1 & (mod & q); \end{cases}$$

Le radici quadrate non banali dell'unità

- Quindi ci sono quattro soluzioni distinte di cui due sono quelle banali:

$$\begin{cases} x \equiv 1 \pmod{p}, \\ x \equiv 1 \pmod{q}; \end{cases} \quad \begin{cases} x \equiv p-1 \pmod{p}, \\ x \equiv q-1 \pmod{q}; \end{cases}$$

e corrispindono alle soluzioni $x = 1$ e $x = n - 1 = pq - 1$. Ma ve ne sono altre due sono non banali:

$$\begin{cases} x \equiv 1 \pmod{p}, \\ x \equiv q-1 \pmod{q}; \end{cases} \quad \begin{cases} x \equiv p-1 \pmod{p}, \\ x \equiv 1 \pmod{q}. \end{cases}$$

Le radici quadrate non banali dell'unità

- Quindi ci sono quattro soluzioni distinte di cui due sono quelle banali:

$$\begin{cases} x \equiv 1 \pmod{p}, \\ x \equiv 1 \pmod{q}; \end{cases} \quad \begin{cases} x \equiv p-1 \pmod{p}, \\ x \equiv q-1 \pmod{q}; \end{cases}$$

e corrispindono alle soluzioni $x = 1$ e $x = n - 1 = pq - 1$. Ma ve ne sono altre due sono non banali:

$$\begin{cases} x \equiv 1 \pmod{p}, \\ x \equiv q-1 \pmod{q}; \end{cases} \quad \begin{cases} x \equiv p-1 \pmod{p}, \\ x \equiv 1 \pmod{q}. \end{cases}$$

- Conoscendo una delle due radici non banali:

$$x_1^2 \equiv 1 \; con \; 1 < x < n - 1.$$

Le radici quadrate non banali dell'unità

- Quindi ci sono quattro soluzioni distinte di cui due sono quelle banali:

$$\begin{cases} x \equiv 1 \pmod{p}, \\ x \equiv 1 \pmod{q}; \end{cases} \quad \begin{cases} x \equiv p-1 \pmod{p}, \\ x \equiv q-1 \pmod{q}; \end{cases}$$

e corrispindono alle soluzioni $x = 1$ e $x = n - 1 = pq - 1$. Ma ve ne sono altre due sono non banali:

$$\begin{cases} x \equiv 1 \pmod{p}, \\ x \equiv q-1 \pmod{q}; \end{cases} \quad \begin{cases} x \equiv p-1 \pmod{p}, \\ x \equiv 1 \pmod{q}. \end{cases}$$

- Conoscendo una delle due radici non banali:

$$x_1^2 \equiv 1 \; con \; 1 < x < n - 1.$$

- si ricava facilmente anche l'altra ($x_2 = n - x_1$):

$$x_2^2 = (n - x_1)^2 = n^2 - 2nx_1 + x_1^2 \equiv 1.$$

Decomposizione tramite una radice quadrata dell'unità

- Basta conoscere una radice quadrata non banale dell'unità per decomporre n:

$$x_1^2 \equiv 1 \Leftrightarrow (x_1 - 1)(x_1 + 1) \equiv 0 \ (mod\ n).$$

Decomposizione tramite una radice quadrata dell'unità

- Basta conoscere una radice quadrata non banale dell'unità per decomporre n:

$$x_1^2 \equiv 1 \Leftrightarrow (x_1 - 1)(x_1 + 1) \equiv 0 \ (mod\ n).$$

- che in $\mathbb{Z}$ diviene:

$$(x_1 - 1)(x_1 + 1) = kn.$$

Decomposizione tramite una radice quadrata dell'unità

- Basta conoscere una radice quadrata non banale dell'unità per decomporre n:

$$x_1^2 \equiv 1 \Leftrightarrow (x_1 - 1)(x_1 + 1) \equiv 0 \ (mod\ n).$$

- che in $\mathbb{Z}$ diviene:

$$(x_1 - 1)(x_1 + 1) = kn.$$

- Per il teorema dell' unicità della decomposizione, uno dei fattori $x_1 \pm 1$ è multiplo di p e l'altro di q.

Decomposizione tramite una radice quadrata dell'unità

- Basta conoscere una radice quadrata non banale dell'unità per decomporre n:

$$x_1^2 \equiv 1 \Leftrightarrow (x_1 - 1)(x_1 + 1) \equiv 0 \ (mod\ n).$$

- che in $\mathbb{Z}$ diviene:

$$(x_1 - 1)(x_1 + 1) = kn.$$

- Per il teorema dell' unicità della decomposizione, uno dei fattori $x_1 \pm 1$ è multiplo di p e l'altro di q.
- Quindi per calcolare p (o q) basta calcolare il massimo comune divisore tra $x_1 - 1$ ed n.

$$\begin{cases} q &= ((x_1 - 1), n), \\ p &= ((x_1 + 1), n). \end{cases}$$

Criteri basati sulle radici quadrate dell'unità - cont

- Da quanto visto, è possibile definire un criterio di pseudoprimalità con i seguenti passi:

Criteri basati sulle radici quadrate dell'unità - cont

- Da quanto visto, è possibile definire un criterio di pseudoprimalità con i seguenti passi:
- Si scelgono a caso m numeri maggiori di n/2 e si elevano al quadrato (analogamente al metodo di Fermat).

Criteri basati sulle radici quadrate dell'unità - cont

- Da quanto visto, è possibile definire un criterio di pseudoprimalità con i seguenti passi:
- Si scelgono a caso m numeri maggiori di n/2 e si elevano al quadrato (analogamente al metodo di Fermat).
- Se nessuno dei numeri è congruo all'unità il numero è pseudoprimo.

Criteri basati sulle radici quadrate dell'unità - cont

- Da quanto visto, è possibile definire un criterio di pseudoprimalità con i seguenti passi:
- Si scelgono a caso m numeri maggiori di n/2 e si elevano al quadrato (analogamente al metodo di Fermat).
- Se nessuno dei numeri è congruo all'unità il numero è pseudoprimo.
- Se scopriamo una radice dell'unità possiamo decomporre n e quindi decifrare RSA. [Il Quantum Computing fornisce algoritmi in grado di risolvere questo problema calcolando i quadrati di tutti i valori contemporaneamente]

Criteri basati sulle radici quadrate dell'unità - cont

- Da quanto visto, è possibile definire un criterio di pseudoprimalità con i seguenti passi:
- Si scelgono a caso m numeri maggiori di n/2 e si elevano al quadrato (analogamente al metodo di Fermat).
- Se nessuno dei numeri è congruo all'unità il numero è pseudoprimo.
- Se scopriamo una radice dell'unità possiamo decomporre n e quindi decifrare RSA. [Il Quantum Computing fornisce algoritmi in grado di risolvere questo problema calcolando i quadrati di tutti i valori contemporaneamente]
- Possiamo anche combinare i due metodi. Si eleva k al quadrato, poi se è congruo all'unità o se differisce da n per un quadrato perfetto, il numero n è scomposto.

Soluzione esercizio Spia

- f(12)=6, f(10)=5, f(8)=4, f(6)=3, f(4)=7.

Soluzione esercizio Spia

- f(12)=6, f(10)=5, f(8)=4, f(6)=3, f(4)=7.
- La soluzione più semplice è contare il numero di lettere necessario a scrivere il numero gridato dalla sentinella.

Soluzione esercizio Spia

- f(12)=6, f(10)=5, f(8)=4, f(6)=3, f(4)=7.
- La soluzione più semplice è contare il numero di lettere necessario a scrivere il numero gridato dalla sentinella.
- dodici ha 6 lettere; dieci ne ha 5, otto ne ha quattro, sei ne ha tre; ma quattro ne ha 7.

Soluzione esercizio Spia

- f(12)=6, f(10)=5, f(8)=4, f(6)=3, f(4)=7.
- La soluzione più semplice è contare il numero di lettere necessario a scrivere il numero gridato dalla sentinella.
- dodici ha 6 lettere; dieci ne ha 5, otto ne ha quattro, sei ne ha tre; ma quattro ne ha 7.
- Ovviamente esistono infiniti algoritmi che risolvono il problema, anche con algoritmi polinomiali (teorema di Lagrange). Ma questo è sufficientemente semplice da essere ricordato e calcolato rapidamente.

Soluzione esercizio Spia

- f(12)=6, f(10)=5, f(8)=4, f(6)=3, f(4)=7.
- La soluzione più semplice è contare il numero di lettere necessario a scrivere il numero gridato dalla sentinella.
- dodici ha 6 lettere; dieci ne ha 5, otto ne ha quattro, sei ne ha tre; ma quattro ne ha 7.
- Ovviamente esistono infiniti algoritmi che risolvono il problema, anche con algoritmi polinomiali (teorema di Lagrange). Ma questo è sufficientemente semplice da essere ricordato e calcolato rapidamente.
- L'esempio della spia mostra anche come un algoritmo apparentemente complicato (contare il numreo di lettere che compongono il numero) possa condurre, in casi particolari, a soluzioni semplici.

Nuovi esercizi

- Scrivere un codice Octave che esegua i crivello di Eratostene.

Nuovi esercizi

- Scrivere un codice Octave che esegua i crivello di Eratostene.
- Scrivere codice Octave che calcola una potenza di un numero (modulo n) nel minor numero di passi possibile (Quick-power).

Nuovi esercizi

- Scrivere un codice Octave che esegua i crivello di Eratostene.
- Scrivere codice Octave che calcola una potenza di un numero (modulo n) nel minor numero di passi possibile (Quick-power).
- Scrivere un codice che verifichi la condizione di pseudoprimalità di Fermat con i primi $n^{1/4}$ numeri primi.

Calcolo delle potenze

- In molte occasioni è necessario calcolare le potenze di un elemento A di un gruppo G ad una potenza n. Il gruppo può non essere numerico e nemmeno abeliano. Vedremo il prodotto a sinistra, ma il ragionamento è analogo per il prodotto a destra.

Calcolo delle potenze

- In molte occasioni è necessario calcolare le potenze di un elemento A di un gruppo G ad una potenza n. Il gruppo può non essere numerico e nemmeno abeliano. Vedremo il prodotto a sinistra, ma il ragionamento è analogo per il prodotto a destra.
- L'algoritmo più semplice consiste nello iterare il prodotto (a sinistra) per A:

$$A^{k+1} = A \cdot A^k.$$

Calcolo delle potenze

- In molte occasioni è necessario calcolare le potenze di un elemento A di un gruppo G ad una potenza n. Il gruppo può non essere numerico e nemmeno abeliano. Vedremo il prodotto a sinistra, ma il ragionamento è analogo per il prodotto a destra.
- L'algoritmo più semplice consiste nello iterare il prodotto (a sinistra) per A:
$$A^{k+1} = A \cdot A^k.$$
- Questo algoritmo richiede $n-1$ prodotti e una allocazione di memoria pari a 3 volte quella dell'elemento A. Se A è un numero intere servono 3 interi. Se A è una matrice di interi 5x5 serviranno 3 x(5x5)=75 interi.

Potenze di Operatori

- Ad ogni elemento A è associato un operatore lineare T_A:

$$T_A(B) \stackrel{def}{=} A \cdot B.$$

Potenze di Operatori

- Ad ogni elemento A è associato un operatore lineare T_A:

$$T_A(B) \stackrel{def}{=} A \cdot B.$$

- Per il calcolo dell'azione dell'operatore iterato n volte occorrono n passi:

$$T_A^{(n)}(B) \stackrel{def}{=} \overbrace{T_A(\ldots T_A(B)\ldots)}^{n \text{ volte}} = T_A(T_A^{(n-1)}(B)) = A \cdot T_A^{(n-1)}(B).$$

Calcolo Rapido delle potenze.

- Dobbiamo calcolare $B = A^k$:

Calcolo Rapido delle potenze.

- Dobbiamo calcolare $B = A^k$:
- Scriviamo k in formato binario (naturale sui computer). Poniamo $m = [\log_2(k)]$ (nextpow già studiato).

$$k = \sum_{i=0,m} b_i 2^i :$$

Calcolo Rapido delle potenze.

- Dobbiamo calcolare $B = A^k$:
- Scriviamo k in formato binario (naturale sui computer). Poniamo $m = [\log_2(k)]$ (nextpow già studiato).

$$k = \sum_{i=0,m} b_i 2^i :$$

- Si può riscrivere:

$$k = b_0 + 2(b_1 + 2(b_2 + \cdots 2b_m)))));$$

- Esponenziando

$$A^k = A^{b_0+2(b_1+2(b_2+\cdots 2b_m)))};$$

Calcolo Rapido delle potenze.

- Dobbiamo calcolare $B = A^k$:
- Scriviamo k in formato binario (naturale sui computer). Poniamo $m = [\log_2(k)]$ (nextpow già studiato).

$$k = \sum_{i=0,m} b_i 2^i :$$

- Si può riscrivere:

$$k = b_0 + 2(b_1 + 2(b_2 + \cdots 2b_m)))));$$

- Esponenziando

$$A^k = A^{b_0+2(b_1+2(b_2+\cdots 2b_m)))};$$

- Che equivale a:

$$A^k = A^{b_0} * (A^2)^{(b_1)} * (A^4)^{(b_2)} \cdots (A^{2^m})^{b_m}.$$

Calcolo Rapido delle potenze: algoritmo

- Vediamo uno schema per l'algoritmo per il calcolo di $B = T_A^{(k)} A_0 = A^k A_0$ (ad esempio $B = A^k$)

Calcolo Rapido delle potenze: algoritmo

- Vediamo uno schema per l'algoritmo per il calcolo di $B = T_A^{(k)} A_0 = A^k A_0$ (ad esempio $B = A^k$)
- Inizializzazione: B=I (in generale $B = A_0$);

Calcolo Rapido delle potenze: algoritmo

- Vediamo uno schema per l'algoritmo per il calcolo di $B = T_A^{(k)} A_0 = A^k A_0$ (ad esempio $B = A^k$)
- Inizializzazione: B=I (in generale $B = A_0$);
- Ciclo: y=mod(k,2) (legge primo bit a dx di k)

Calcolo Rapido delle potenze: algoritmo

- Vediamo uno schema per l'algoritmo per il calcolo di $B = T_A^{(k)} A_0 = A^k A_0$ (ad esempio $B = A^k$)
- Inizializzazione: B=I (in generale $B = A_0$);
- Ciclo: y=mod(k,2) (legge primo bit a dx di k)
- if (y != 0) B=A*B (in generale $B = T_A(B)$);

Calcolo Rapido delle potenze: algoritmo

- Vediamo uno schema per l'algoritmo per il calcolo di $B = T_A^{(k)} A_0 = A^k A_0$ (ad esempio $B = A^k$)
- Inizializzazione: B=I (in generale $B = A_0$);
- Ciclo: y=mod(k,2) (legge primo bit a dx di k)
- if (y != 0) B=A*B (in generale $B = T_A(B)$);
- A=A*A

Calcolo Rapido delle potenze: algoritmo

- Vediamo uno schema per l'algoritmo per il calcolo di $B = T_A^{(k)} A_0 = A^k A_0$ (ad esempio $B = A^k$)
- Inizializzazione: B=I (in generale $B = A_0$);
- Ciclo: y=mod(k,2) (legge primo bit a dx di k)
- if (y != 0) B=A*B (in generale $B = T_A(B)$);
- A=A*A
- k=(k-y)/2; (shift a dx di 1 passo).

Calcolo Rapido delle potenze: algoritmo

- Vediamo uno schema per l'algoritmo per il calcolo di $B = T_A^{(k)} A_0 = A^k A_0$ (ad esempio $B = A^k$)
- Inizializzazione: B=I (in generale $B = A_0$);
- Ciclo: y=mod(k,2) (legge primo bit a dx di k)
- if (y != 0) B=A*B (in generale $B = T_A(B)$);
- A=A*A
- k=(k-y)/2; (shift a dx di 1 passo).
- Continua il ciclo e si arresta per k=0.

Calcolo Rapido delle potenze: algoritmo

- Vediamo uno schema per l'algoritmo per il calcolo di $B = T_A^{(k)} A_0 = A^k A_0$ (ad esempio $B = A^k$)
- Inizializzazione: B=I (in generale $B = A_0$);
- Ciclo: y=mod(k,2) (legge primo bit a dx di k)
- if (y != 0) B=A*B (in generale $B = T_A(B)$);
- A=A*A
- k=(k-y)/2; (shift a dx di 1 passo).
- Continua il ciclo e si arresta per k=0.
- Bastano m passi, cioè $\log_2 k$ iterazioni e allocazione di 4 interi (o due interi e due oggetti di tipo A).

Calcolo Rapido delle potenze: algoritmo

- Vediamo uno schema per l'algoritmo per il calcolo di $B = T_A^{(k)} A_0 = A^k A_0$ (ad esempio $B = A^k$)
- Inizializzazione: B=I (in generale $B = A_0$);
- Ciclo: y=mod(k,2) (legge primo bit a dx di k)
- if (y != 0) B=A*B (in generale $B = T_A(B)$);
- A=A*A
- k=(k-y)/2; (shift a dx di 1 passo).
- Continua il ciclo e si arresta per k=0.
- Bastano m passi, cioè $\log_2 k$ iterazioni e allocazione di 4 interi (o due interi e due oggetti di tipo A).
- Esercizio: Scrivere codice Octave per il "quickpower".

Gruppi Ciclici

- Un gruppo G si dice ciclico quando esiste un elemento g tale che ogni elemento del gruppo sia una potenza di g:

$$\forall a \in G \exists k : a = g^k.$$

Gruppi Ciclici

- Un gruppo G si dice ciclico quando esiste un elemento g tale che ogni elemento del gruppo sia una potenza di g:

$$\forall a \in G \exists k : a = g^k.$$

- L'elemento g è detto generatore del gruppo.

Gruppi Ciclici

- Un gruppo G si dice ciclico quando esiste un elemento g tale che ogni elemento del gruppo sia una potenza di g:

$$\forall a \in G \exists k : a = g^k.$$

- L'elemento g è detto generatore del gruppo.
- I gruppi ciclici sono commutativi

$$\forall a, b \in G : a \cdot b = g^{k_a} \cdot g^{k_b} = g^{k_a + k_b} = g^{k_b} \cdot g^{k_a} = b \cdot a.$$

Gruppi Ciclici

- Un gruppo G si dice ciclico quando esiste un elemento g tale che ogni elemento del gruppo sia una potenza di g:

$$\forall a \in G \exists k : a = g^k.$$

- L'elemento g è detto generatore del gruppo.
- I gruppi ciclici sono commutativi

$$\forall a, b \in G : a \cdot b = g^{k_a} \cdot g^{k_b} = g^{k_a + k_b} = g^{k_b} \cdot g^{k_a} = b \cdot a.$$

- Ogni gruppo ciclico finito di cardinalità N è isomorfo a $(\mathbb{Z}_{N-1}, +)$ (si noti rispetto alla somma, NON al prodotto). l'isomorfismo è dato dalla relazione f biunivoca:

$$\forall a = g^{k_a} : f(a) = k_a.$$

Isomorfismo tra un gruppo ciclico G ($|G| = N$) e $(\mathbb{Z}_N, +)$

- L'esistenza di un generatore g consente di definire una trasformazione canonica che realizza l'isomorfismo. Si costruisce esplicitamente:

$$\forall a : f(a) = k_a : a = g^{k_a}.$$

Isomorfismo tra un gruppo ciclico G ($|G| = N$) e $(\mathbb{Z}_N, +)$

- L'esistenza di un generatore g consente di definire una trasformazione canonica che realizza l'isomorfismo. Si costruisce esplicitamente:

$$\forall a : f(a) = k_a : a = g^{k_a}.$$

- In linguaggio naturale: l'immagine di ogni elemento è l'esponente che bisogna dare al generatore g per ottenere l'argomento delle funzione. Per tale ragione la funzione f viene detta "logaritmo".

$$f(a) \stackrel{def}{=} \log_g(a).$$

Isomorfismo tra un gruppo ciclico G ($|G| = N$) e $(\mathbb{Z}_N, +)$

- L'esistenza di un generatore g consente di definire una trasformazione canonica che realizza l'isomorfismo. Si costruisce esplicitamente:

$$\forall a : f(a) = k_a : a = g^{k_a}.$$

- In linguaggio naturale: l'immagine di ogni elemento è l'esponente che bisogna dare al generatore g per ottenere l'argomento delle funzione. Per tale ragione la funzione f viene detta "logaritmo".

$$f(a) \stackrel{def}{=} \log_g(a).$$

- La funzione f è iniettiva:

$$\forall a \neq b : f(a) \neq f(b) \Leftrightarrow g^{k_a} \neq g^{k_b} \rightarrow k_a \neq k_b;$$

Isomorfismo tra un gruppo ciclico G ($|G| = N$) e $(\mathbb{Z}_N, +)$

- La funzione f è invertibile. La funzione inversa $f^{-1}()$ si definisce semplicemente per esponenziazione:

$$f^{-1}(k) \stackrel{def}{=} g^k.$$

$$f^{-1}(f(a)) = f^{-1}(k_a) = g^{k_a} = a.$$

Isomorfismo tra un gruppo ciclico G ($|G| = N$) e $(\mathbb{Z}_N, +)$

- La funzione f è invertibile. La funzione inversa $f^{-1}()$ si definisce semplicemente per esponenziazione:

$$f^{-1}(k) \stackrel{def}{=} g^k.$$

$$f^{-1}(f(a)) = f^{-1}(k_a) = g^{k_a} = a.$$

- Un metodo molto semplice per costruire gruppi ciclici in $\mathbb{Z}_n^*$ consistere nello scegliere un suo elemento a e calcolarne le potenze. Essendo lo spazio $\mathbb{Z}_n^*$ finito necessariamente una potenza fornirà il valore unitario. Il terorema di Eulero ci assicura che si verificherà al pi'u alla potenza $\phi(n)$-esima, ma potrebbe verificarsi prima.

Quando un gruppo moltiplicativo $(\mathbb{Z}_n^*, x)$ è ciclico? cioè quando ammette un generatore?

- Il problema fondamentale dei gruppi moltiplicativi $(\mathbb{Z}_n^*, x)$ è capire se esiste un generatore. Questo consente di trovare notevoli proprietà per lo spazio ed in particolare per i periodi dei suoi elementi.

Quando un gruppo moltiplicativo $(\mathbb{Z}_n^*, x)$ è ciclico? cioè quando ammette un generatore?

- Il problema fondamentale dei gruppi moltiplicativi $(\mathbb{Z}_n^*, x)$ è capire se esiste un generatore. Questo consente di trovare notevoli proprietà per lo spazio ed in particolare per i periodi dei suoi elementi.
- Dimostreremo che tutti e soli gli spazi che ammettono un generatore sono $(\mathbb{Z}_p^*, x)$ (con p primo), $(\mathbb{Z}_{p^k}^*, x)$ (con p primo dispari), $(\mathbb{Z}_{2p^k}^*, x)$, $\mathbb{Z}_2^*$ e $\mathbb{Z}_4^*$

Quando un gruppo moltiplicativo $(\mathbb{Z}_n^*, x)$ è ciclico? cioè quando ammette un generatore?

- Il problema fondamentale dei gruppi moltiplicativi $(\mathbb{Z}_n^*, x)$ è capire se esiste un generatore. Questo consente di trovare notevoli proprietà per lo spazio ed in particolare per i periodi dei suoi elementi.
- Dimostreremo che tutti e soli gli spazi che ammettono un generatore sono $(\mathbb{Z}_p^*, x)$ (con p primo), $(\mathbb{Z}_{p^k}^*, x)$ (con p primo dispari), $(\mathbb{Z}_{2p^k}^*, x)$, $\mathbb{Z}_2^*$ e $\mathbb{Z}_4^*$
- In altri termini, $(\mathbb{Z}_n^*, x)$ è un gruppo ciclico se e solo se n è della forma su dette:

$$n = 2^{\alpha} p^{h};$$

con $\alpha = \{0, 1\}$ e p primo dispari oppure $p = 2$ e $h = 1$.

Periodi degli elementi dei Gruppi Ciclici

- Dato un gruppo ciclico G di cardinalità $N = |G|$, generato da un generatore g:

$$G \stackrel{def}{=} \left\{1, g, g^2, \ldots, g^{N-1}\right\} \stackrel{def}{=} \{g_0, g_1, g_2, \ldots, g_{N-1}\}.$$

  Valgono diverse proprietà utili:

Periodi degli elementi dei Gruppi Ciclici

- Dato un gruppo ciclico G di cardinalità $N = |G|$, generato da un generatore g:

$$G \stackrel{def}{=} \left\{1, g, g^2, \ldots, g^{N-1}\right\} \stackrel{def}{=} \left\{g_0, g_1, g_2, \ldots, g_{N-1}\right\}.$$

Valgono diverse proprietà utili:

- Ogni generatore ha periodo $\tau = N$. Infatti tutti gli elementi devono essere diversi tra loro:

$$g^k \neq g^m;$$

altrimenti se $m = k + \tau$: $g^m = g^{k+\tau} = 1$.

Periodi degli elementi dei Gruppi Ciclici

- Dato un gruppo ciclico G di cardinalità $N = |G|$, generato da un generatore g:

$$G \stackrel{def}{=} \left\{1, g, g^2, \ldots, g^{N-1}\right\} \stackrel{def}{=} \{g_0, g_1, g_2, \ldots, g_{N-1}\}.$$

Valgono diverse proprietà utili:

- Ogni generatore ha periodo $\tau = N$. Infatti tutti gli elementi devono essere diversi tra loro:

$$g^k \neq g^m;$$

altrimenti se $m = k + \tau$: $g^m = g^{k+\tau} = 1$.

- Quindi $g^k = 1$ implica $k = l \cdot N$ [k multiplo di N].

Periodi degli elementi dei Gruppi Ciclici

- Dato un gruppo ciclico G di cardinalità $N = |G|$, generato da un generatore g:

$$G \stackrel{def}{=} \left\{1, g, g^2, \ldots, g^{N-1}\right\} \stackrel{def}{=} \{g_0, g_1, g_2, \ldots, g_{N-1}\}.$$

Valgono diverse proprietà utili:

- Ogni generatore ha periodo $\tau = N$. Infatti tutti gli elementi devono essere diversi tra loro:

$$g^k \neq g^m;$$

altrimenti se $m = k + \tau$: $g^m = g^{k+\tau} = 1$.

- Quindi $g^k = 1$ implica $k = l \cdot N$ [k multiplo di N].
- Il periodo di ogni elemento è un divisore di N:

$$a^\tau = (g^k)^\tau = g^{k\tau} \Rightarrow k\tau = l \cdot N.$$

quindi il periodo di g_k è il rapporto tra n ed MCD tra N e k.

Il periodo di un elemento di ordine k

- In un gruppo ciclico di ordine N; se $a = g_k = g^k$ il suo periodo è pari a N diviso per (k,N). Abbiamo visto che

$$k\tau = l \cdot N.$$

Il periodo di un elemento di ordine k

- In un gruppo ciclico di ordine N; se $a = g_k = g^k$ il suo periodo è pari a N diviso per (k,N). Abbiamo visto che

$$k\tau = l \cdot N.$$

- Sia M=(k,N) il MCD tra N e k. Possiamo porre:

$$\begin{cases} k' & \stackrel{def}{=} & k/M \\ N' & \stackrel{def}{=} & N/M \end{cases}$$

Il periodo di un elemento di ordine k

- In un gruppo ciclico di ordine N; se $a = g_k = g^k$ il suo periodo è pari a N diviso per (k,N). Abbiamo visto che

$$k\tau = l \cdot N.$$

- Sia M=(k,N) il MCD tra N e k. Possiamo porre:

$$\begin{cases} k' & \stackrel{def}{=} & k/M \\ N' & \stackrel{def}{=} & N/M \end{cases}$$

- Quindi

$$k'\tau = l \cdot N'.$$

il più piccolo dei τ si ottiene per $l = k'$ e $\tau = N'$. infatti k' e N' sono primi tra loro e quindi per l'unicità della decomposizione in fattori τ è un multiplo di $N' = N/M$.

Quanti generatori ha un gruppo ciclico?

- Se un elemento di G è una potenza k del generatore prima con $N = |G|$, allora ha per periodo N. Di conseguenza ogni g_k potenza prima con N è un generatore del gruppo ciclico.

$$g_k : (k, N) = 1 \Rightarrow \tau(g_k) = N.$$

Quanti generatori ha un gruppo ciclico?

- Se un elemento di G è una potenza k del generatore prima con $N = |G|$, allora ha per periodo N. Di conseguenza ogni g_k potenza prima con N è un generatore del gruppo ciclico.

$$g_k : (k, N) = 1 \Rightarrow \tau(g_k) = N.$$

- Il numero di generatori di un gruppo ciclico è pari alla funzione di Eulero della cardinalità. Nel caso $G = (\mathbb{Z}_n^*, x)$

$$|G| = |\mathbb{Z}_n^*| = \phi(n) = N.$$

$$\#gen = \phi\left(|\mathbb{Z}_n^*|\right) = \phi(N) = \phi(\phi(n)).$$

Generatori di $\mathbb{Z}_p^*$

- Dimostreremo che $(\mathbb{Z}_p^*, x)$ (con p primo) ammette sempre un generatore e dunque ne ammette $\phi(N) = \phi(\phi(n))$. Useremo i seguenti lemmi:

Generatori di $\mathbb{Z}_p^*$

- Dimostreremo che $(\mathbb{Z}_p^*, x)$ (con p primo) ammette sempre un generatore e dunque ne ammette $\phi(N) = \phi(\phi(n))$. Useremo i seguenti lemmi:
- Lemma 1: "Commensurabilità": Se un elemento ha due ciclicità prime tra loro allora è l'elemento neutro.

Generatori di $\mathbb{Z}_p^*$

- Dimostreremo che $(\mathbb{Z}_p^*, x)$ (con p primo) ammette sempre un generatore e dunque ne ammette $\phi(N) = \phi(\phi(n))$. Useremo i seguenti lemmi:
- Lemma 1: "Commensurabilità": Se un elemento ha due ciclicità prime tra loro allora è l'elemento neutro.
- Lemma 2: "Prodotto": Se due elementi hanno periodi primi tra loro, il loro prodotto ha per periodo il prodotto dei periodi.

Generatori di $\mathbb{Z}_p^*$

- Dimostreremo che $(\mathbb{Z}_p^*, x)$ (con p primo) ammette sempre un generatore e dunque ne ammette $\phi(N) = \phi(\phi(n))$. Useremo i seguenti lemmi:
- Lemma 1: "Commensurabilità": Se un elemento ha due ciclicità prime tra loro allora è l'elemento neutro.
- Lemma 2: "Prodotto": Se due elementi hanno periodi primi tra loro, il loro prodotto ha per periodo il prodotto dei periodi.
- Lemma 3: Unicità delle radici di ogni grado in un campo.

Generatori di $\mathbb{Z}_p^*$

- Dimostreremo che $(\mathbb{Z}_p^*, x)$ (con p primo) ammette sempre un generatore e dunque ne ammette $\phi(N) = \phi(\phi(n))$. Useremo i seguenti lemmi:
- Lemma 1: "Commensurabilità": Se un elemento ha due ciclicità prime tra loro allora è l'elemento neutro.
- Lemma 2: "Prodotto": Se due elementi hanno periodi primi tra loro, il loro prodotto ha per periodo il prodotto dei periodi.
- Lemma 3: Unicità delle radici di ogni grado in un campo.
- Mettendo insieme i lemmi si deduce che il massimo periodo di $\mathbb{Z}_p^*$ è $p-1$ e dunque ammette un generatore ed è ciclico.