Crittografia

Federico Matteoni

A.A. 2020/21

Indice

1 Introduzione alla Crittografia **5**

1.1 Introduzione . . . 5

1.1.1 Lo scenario . . . 5

1.1.2 Antichi esempi . . . 6

1.2 Livello di segretezza . . . 6

1.2.1 Chiavi segreta . . . 6

1.2.2 Crittoanalista . . . 7

1.2.3 Situazione attuale . . . 7

1.2.4 Cifrari odierni . . . 7

1.3 Rappresentazione matematica di oggetti . . . 8

1.4 Richiamo della teoria della calcolabilità . . . 9

1.4.1 Algoritmi . . . 10

1.4.2 Modelli di calcolo . . . 10

1.4.3 Decidibilità e trattabilità . . . 11

1.4.4 Tipologie di problemi . . . 11

1.4.5 Classi di complessità . . . 12

1.4.6 Certificato . . . 12

1.4.7 Classi co-P e co-NP . . . 13

2 Sequenze Casuali **15**

2.1 Esempi di algoritmi numerici . . . 15

2.2 Casualità . . . 15

2.2.1 Sequenze casuali . . . 16

2.2.2 Sorgente binaria casuale . . . 17

2.2.3 Generatore di sequenze brevi . . . 17

2.3 Test statistici . . . 17

2.4 Generatori crittograficamente sicuri . . . 18

Introduzione

Prof.ssa: Anna Bernasconi.
Vedremo i cifrari da un punto di vista prettamente algoritmico. Vedremo anche i cifrari storici, ormai non più utilizzabili, perché hanno "aperto la strada", per poi passare ai cifrari perfetti (soluzione ideale ma con costo elevato).
Poi esamineremo i cifrari simmetrici, a chiave pubblica, curve ellittiche, firma digitale, SSL. Protocolli zero knowledge, blockchain e crittografia quantistica.

Libro di testo: Bernasconi, Ferragina, Luccio - Elementi di Crittografia.

Esame Orali nel caso di esami a distanza, scritto nel caso di esami in presenza, closed-book.

Capitolo 1

Introduzione alla Crittografia

1.1 Introduzione

Crittografia Significa "*scrittura nascosta*", si intendono tecniche matematiche per mascherare i messaggi per non renderli leggibili a terzi (**crittografia**) o tentare di svelarli quando non si è il legittimo destinatario (**crittoanalisi**). Quindi tecniche di protezione e viceversa.
Esiste per i due mondi in contrapposizione: persone che vogliono scambiarsi privatamente informazioni e gli *impiccioni* che desiderano ascoltare o intromettersi nelle conversazioni altrui (per curiosità, investigazione o altri scopi).

Due gruppi di persone Chi vuole proteggersi userà **metodi di cifratura**, gli altri useranno **metodi di crittoanalisi**

Crittografia Metodi di Cifratura

Crittoanalisi Metodi di . . . **crittologia** studio comunicazione canali non sicuri e relativi problemi

1.1.1 Lo scenario

Alice vuole comunicare con Bob su un canale insicuro, quindi adottano un metodo di cifratura per spedire il messaggio in chiaro `m` sottoforma di crittogramma `c` (testo cifrato) che deve essere: incomprensibile al crittoanalista Eve (eavesdropper) in ascolto sul canale, ma facilmente decifrabile da Bob.

MSG Insieme dei messaggi in chiaro

CRITTO Insieme dei crittogrammi

$$C : MSG \to CRITTO$$
$$D : CRITTO \to MSG$$

Sono operazioni da poter fare in tempo polinomiale. C e D sono una l'inversa dell'altra, ma C **deve essere iniettiva**.

1.1.2 Antichi esempi

Erodoto Nelle *Storie*, V secolo a.C.
Messaggi tatuati sulla testa, coperti dai capelli e riscoperti rasando la testa.

Scitale Spartani. Asta cilindrica in due esemplari identici. Si avvolgeva una striscia di carta attorno al cilindro e scritta. La chiave del cifrario è il diametro dello scitale.

Enea Tattico Un libro qualsiasi con un insieme di lettere segnate, o sostituire le vocali con simboli grafici.

Cifrario di Cesare Il più antico cifrario di concezione moderna. L'idea di base è che il crittogramma è ottenuto dal messaggio in chiaro `m` sostituendo ogni lettera con quella di tre posizioni più avanti nell'alfabeto.
Es. A $\to$ D, Z $\to$ C. La segretezza dipende interamente dalla conoscenza del metodo, era destinato all'uso ristretto da un piccolo gruppo di persone.

1.2 Livello di segretezza

Classificazione in base al livello di segretezza

> Cifrari per **uso ristretto**
> Le tecniche con cui si calcola e decifra il crittogramma sono tenute segrete in ogni loro aspetto. Impiegati per comunicazioni classificate (diplomatiche o militari), non adatti per uso di massa.
>
> Cifrari per **uso generale**
> Ogni codice segreto non può essere mantenuto tale per troppo a lungo. La parte segreta si limita alla chiave, nota solamente agli utenti che stanno comunicando.
> Vengono studiati dalla comunità, coinvolgendo tantissime persone. Solo la chiave deve essere segreta.
> **Il nemico conosce il sistema**.
> Quindi C e D sono note, la chiave **segreta** `k` è usata come input sia in C che in D:
> `c = C(m, k)`, `m = D(c, k)`
> Se non si conosce `k`, anche conoscendo C e D non si possono estrarre informazioni dal crittogramma.
> Tenere segreta una sola chiave è più facile che segretare l'intero metodo. Tutti possono usare C e D pubbliche con chiavi diverse, e se un crittoanalista entra in possesso di una chiave posso generarne semplicemente una nuova.

1.2.1 Chiavi segreta

Se la segretezza dipende unicamente dalla chiave bisogna proteggersi dagli attacchi a forza bruta, quindi avere un gran numero di chiavi, così da essere immuni da chi le prova tutte.
Inoltre la chiave deve essere scelta in modo casuale e non prevedibile, sennò il crittoanalista può provare le chiavi ovvie.

Attacco esauriente Il crittoanalisa potrebbe sferrare un attacco a forza bruta verificando la significatività delle sequenze `D(c, k)` $\forall$ `k`.
Se $|\text{Key}| = 10^{20}$ e con un calcolatore che impiega 10^{-6} per calcolare `D(c, k)` servirebbe in media più di un milione di anni per scoprire il messaggi provando tutte le chiavi. Però la segretezza può essere violata con altre tecniche: esistono cifrari più sicuri di altri pur con uno spazio di chiavi più piccoli.
Un cifrario complicato non è necessariamente più sicuro e **mai sottovalutare la bravura del crittoanalista**.

1.2.2 Crittoanalista

Comportamento Il comportamento di un crittoanalista può essere:

Passivo, quando si **limita ad ascoltare** la comunicazione

Attivo, quando **agisce sul canale** disturbando la comunicazione o modificando il contenuto dei messaggi.

Attacchi a un sistema crittografico Hanno l'obiettivo di forzare un sistema. Il metodo e il livello di pericolosità dipendono dalle informazioni in possesso del crittoanalista:

Cipher Text Attack: conosce una serie di crittogrammi

Known Plain-Text Attack: conosce una serie di coppie `(m, c)`

Chosen Plain-Text Attack: si procura coppie `(m, c)` relative a messaggi in chiaro da lui scelti.
Tutta la crittografia a chiave pubblica è soggetta a questo tipo di attacco (avendo la chiave pubblica, cifro dei messaggi che penso possano passare e ascolto finché non trovo nella comunicazione i crittogrammi in mio possesso).

Man in the Middle Il crittoanalista si installa sul canale di comunicazione:

Interrompe le comunicazioni dirette tra gli utenti Alice e Bob

le **sostituisce con messaggi propri**

e **convince** ciascun utente **che tali messaggi provengano leggitimamente dall'altro** utente.

Quindi il crittoanalista **Eve si finge Bob agli occhi di Alice e Alice agli occhi di Bob**.

Esiti

Successo pieno, si scopre completamente D o si ottiene la chiave

Successo limitato, si scopre solo qualche informazione ma sufficiente per comprendere il messaggio

1.2.3 Situazione attuale

Cifrari perfetti Inattaccabili, esistono ma richiedono operazioni complesse, **chiavi lunghe tanto quanto il messaggio e mai riutilizzabili**.
Shannon, 1945 (pubblicato nel 1949 per motivi di segretezza militare): `m` e `c` appaiono totalmente scorrelati, come se `c` fosse una stringa casuale di bit.
Nessuna informazione può filtrare dal crittogramma. Vedremo la teoria matematica.

One-Time Pad Anche detto blocco monouso, sicuro ma per essere usato bene richiede chiavi segrete totalmente casuali e lunghe quanto il messaggio. Come generarla e come scambiarla?

Cifrari attuali Nella crittografia di massa non si usano cifrari perfetti, ma **cifrari *dichiarati* sicuri**, inviolati dagli esperti e che usano algoritmi solo esponenziali per decrittare senza chiave. Il tempo per violare un cifrario è enorme e rende l'operazione insostenibile → impossibilità *pratica* di forzare il cifrario.

Dichiarati sicuri Non è noto se questi problemi matematici richiedano algoritmi *necessariamente* esponenziali o se sono dovuti all'incapacità nostra di trovare metodi più efficienti. Si riconduce a $P = NP$

1.2.4 Cifrari odierni

Advanced Encryption Standard AES, simmetrico a blocchi con chiavi di 128-256bit, pubblicamente noto e realizzabile su computer di ogni tipo. Il messaggio è diviso a blocchi lunghi quanto la chiave.

Le chiavi Sono stabilite dai mezzi elettronici (PC, smartphone, terminale...) e su Internet si scambia una chiave per ogni sessione.

Scambio delle chiavi La chiave va comunicata in sicurezza su un canale non ancora sicuro. Un'intercettazione nello scambio della chiave compromette il sistema.
Nel 1976 viene proposto un algoritmo per generare e scambiare una chiave segreta su un canale insicuro, senza necessità di scambiare informazioni o di incontrarsi in precedenza.
Si chiama **protocollo DH**, ancora largamente utilizzato nei protocolli crittografici su Internet.
Si scambiano pezzi di chiave tramite la rete e unendole a informazioni locali si costruisce la chiave.

Chiave pubblica Diffie ed Hellman hanno anche proposto la crittografia a chiave pubblica.

Cifrari simmetrici: stessa chiave per cifrare e decifrare, nota solo ai due utenti che comunicano. La scelgono di comune accordo e la tengono segreta.

Cifrari asimmetrici: chiavi pubbliche usate per cifrare e chiavi private per decifrare.
`c = C(m, k`$_{pub}$`)`
`m = D(m, k`$_{prv}$`)`
Si rende necessario che la C sia una one-way trapdoor: calcolare il crittogramma deve essere facile (polinomiale), ma decifrare `c` deve essere computazionalmente difficile (a meno di conoscere la trapdoor, la chiave privata).

RSA Rivest, Shamir, Adleman, 1977. Propongono un sistema a chiave pubblico facile da calcolare e difficile da invertire.

Vantaggi

Comunicazione molti a uno
Tutti possono inviare in modo sicuro allo stesso destinatario usando la sua chiave pubblica, ma solo lui può decifrarli. Un crittoanalista non può decifrare anche se conosce C, D e k_{pub}

Se n utenti vogliono comunicare servono solo $2n$ chiavi invece delle $n(n-1)/2$ necessarie nei cifrari simmetrici (una coppia per ogni coppia di utenti)

Non è richiesto nessun scambio

Svantaggi

Sono molto lenti rispetto ai cifrari simmetrici (polinomi di terzo grado)

Sono esposti ad attacchi di tipo chosen plain-text, perché conosco la chiave pubblica
Scelgo un numero qualsiasi di messaggi in chiaro, costruisce i crittogrammi relativi e ascolta sul canale confrontando i crittogrammi in transito e se trova un riscontro sa esattamente qual è il messaggio passato.

Come si usa Oggi si usa un cifrario a chiave segreta (AES) per le comunicazioni di massa, e un cifrario a chiave pubblica per scambiare le chiavi segrete relative al primo senza incontri fisici tra gli utenti.
Diventa lento solo lo scambio delle chiavi. Siamo anche al sicuro da attacchi chosen plain-text perché se la chiave è scelta bene risulta imprevedibile dal crittoanalista.

1.3 Rappresentazione matematica di oggetti

Per rappresentare gli oggetti scegliamo dei **caratteri** da un **insieme finito** detto **alfabeto**.
Un **oggetto** è **rappresentato da una sequenza ordinata di caratteri dell'alfabeto**. L'ordine dei caratteri è importante: a **oggetti diversi corrispondono sequenze diverse** e **il numero di oggetti che si possono rappresentare non ha limiti**. Significa che fissando un numero n arbitrariamente grande possiamo sempre creare un numero di oggetti $> n$, con sequenze via via più grande.

Alfabeto Γ con $|\Gamma| = s$ e N oggetti da rappresentare.
$d(s, N)$: lunghezza della sequenza più lunga che rappresenta un oggetto dell'insieme. A noi interessa la rappresentazione che minimizza $d(s, N)$, cioè $d_{min}(s, N)$
Una rappresentazione è tanto più efficiente quanto $d(s, N)$ si avvicina a $d_{min}(s, N)$

Esempio $s = 1, \Gamma = \{0\}$ l'unica possibilità è variare la lunghezza $\Rightarrow d_{min}(1, N) = N$, estremamente sfavorevole.
$s = 2, \Gamma = \{0, 1\}$, $\forall\, k \geq 1$ ho 2^k sequenze di lunghezza k. Il numero totale di sequenze lunghe da 1 a k è $2^{k+1} - 2$ (si esclude anche la sequenza nulla). Con N oggetti da rappresentare $\Rightarrow k \geq \log_2(N + 2) - 1 \Rightarrow N$ sequenze diverse tutte di $\log_2(N)$ caratteri.

Efficiente Codifica efficiente quando c'è questa riduzione logaritmica, **efficiente** quando . Sequenze della stessa lunghezza è vantaggioso perché non servono caratteri separatori. Per questo è necessario che l'alfabeto contenga almeno due caratteri.
La **notazione posizionale** è una rappresentazione efficiente indipendentemente dalla base $s \geq 2$ scelta. Un intero N è rappresentato con un numero d di cifre $\mid \log_s(N) \leq d \leq \log_s(N) + 1$

1.4 Richiamo della teoria della calcolabilità

Problemi computazionali Formulati matematicamente di cui cerchiamo una soluzione algoritmica: **decidibili** (e **trattabili** o **non trattabili**), o **non decidibili**.
Calcolabilità $\rightarrow$ **Algoritmo** e **problema non decidibile**
Complessità $\rightarrow$ **Algoritmo efficiente** e **problema intrattabile**.

Numerabilità Due insiemi A e B hanno lo stesso numero di elementi $\Leftrightarrow$ si può stabilire una **corrispondenza biunivoca** tra i loro elementi.
Questo porta alla definizione di **numerabile**: un insieme è numerabile $\Leftrightarrow$ i suoi elementi possono essere messi in **corrispondenza biunivoca con i numeri naturali**.
Numerabile significa che **possiede un'infinità numerabile di elementi**. Esempi: l'insieme dei numeri naturali N, l'insieme degli interi Z (avendo n in corrispondenza biunivoca con $2n + 1$ per $n \geq 0$ e $n \leftrightarrow 2|n|$ per $n < 0$, dando la sequenza $0, -1, 1, -2, 2 \ldots$) o anche l'insieme dei naturali pari ($2n \leftrightarrow n$)

Enumerazione delle sequenze Si vuole elencare in uno ordine ragionevole le sequenze di lunghezza finita costruite su un alfabeto finito. Le sequenze non sono in numero finito, quindi non si potrà completare l'elenco.
Lo scopo è **raggiungere qualsiasi sequenza σ arbitrariamente scelta in un numero finito di passi**. σ deve dunque trovarsi a **distanza finita** dall'inizio dell'elenco. Non va bene l'ordine del dizionario perché non saprei la posizione della prima stringa che inizia con b perché le stringhe composte da tutte a sono infinite.
Si stabilisce un ordine tra i caratteri. Si ordinano prima in lunghezza crescente e, a pari lunghezza, in ordine alfabetico.

Esempio $\Gamma = \{a, b, \ldots, z\}$, avrei
$a, b, \ldots, z$,
$aa, ab, \ldots, az, ba, bb, \ldots, bz, \ldots, zz, \ldots$
Ad una sequenza arbitraria corrisponde un numero intero, e la sequenza s arbitraria si troverà tra quelle di lunghezza $|s|$ in posizione alfabetica. Quindi ad una sequenza arbitraria $\leftrightarrow n$ che indica la posizione nell'elenco, e ad un numero naturale $n \leftrightarrow$ la sequenza che occupa l'n-esima posizione nell'elenco.

La **numerazione delle sequenze è fattibile perché sono di lunghezza finita**, anche se illimitata. Cioè per qualunque intero d scelto a priori, esistono sequenze di lunghezza maggiore di d. Per sequenze di lunghezza infinita la numerazione non è possibile

Insiemi non numerabili Insiemi non equivalenti a N come R, $(0, 1)$, l'insieme di tutte le linee del piano, insieme delle funzioni in una o più variabili... $\Rightarrow$ l'**insieme dei problemi computazionali non è numerabile**. Perché un problema computazionale è sempre visualizzabile come una funziona matematica, che associa ad ogni insieme di dati espressi da k numeri interi il corrispondente risultato espresso da j numeri interi

$$f : N^k \rightarrow N^j$$

Quindi l'insieme di queste f **non è numerabile**.

Diagonalizzazione $F = \{$ funzioni $f \mid f : N \rightarrow \{0, 1\}\}$, ogni $f \in F$ è rappresentata da una sequenza infinita

x 0 1 2 3 ... n ...

$f(x)$ 0 1 0 1 ... 0 ...

ma se è possibile è rappresentabile con una regola (f 0 se x pari 1 se x dispari)
Per assurdo, ipotizzo F numerabile. Si può assegnare ad ogni funzione un numero progressivo nella numerazione e costruire una tabella infinita con tutte le funzioni.

x	0	1	2	3	4	5	6	7	8	...
$f_0(x)$	1	0	1	0	1	0	0	0	1	...
$f_1(x)$	0	0	1	1	0	0	1	1	0	...
$f_2(x)$	1	1	0	1	0	1	0	0	1	...
$f_3(x)$	0	1	1	0	1	0	1	1	1	...
$f_4(x)$	1	1	0	0	1	0	0	0	1	...

Definisco $g(x) = \begin{cases} 0 & f_x(x) = 1 \\ 1 & f_x(x) = 0 \end{cases} \Rightarrow g$ non può corrispondere a nessuna delle f_i della tabella, perché differisce da tutte le funzioni almeno nella diagonale principale.
$g(x) \mid 0\,1\,1\,1\ldots$
Per assurdo $\exists\, j \mid g(x) = f_j(x) \Rightarrow g(j) = f_j(j)$ ma per la definizione $g(j)$ è il complemento di $f_j(j)$, quindi $g(j) \neq f_j(j)$ **contraddizione**.
Per qualunque numerazione scelta esiste sempre almeno una funzione esclusa, quindi F non è numerabile.

1.4.1 Algoritmi

Algoritmi la **formulazione di un algoritmo**, una sequenza finita di operazioni, completamente e univocamente determinate, **dipende dal modello di calcolo utilizzato**.
Qualunque modello si scelga, gli algoritmi devono essere descritti da sequenze finite di caratteri di un alfabeto finito $\Rightarrow$ sono **possibilmente infiniti ma numerabili**.

Problemi computazionali Sono **funzioni matematiche** che associano ad ogni insieme di dati il corrispondente risultato, e **non sono numerabili** come visto prima.

Problema della rappresentazione C'è una drastica perdita di potenza, perché gli algoritmi sono numerabili ma sono meno dei problemi computazionali

$$|\{Problemi\}| >> |\{Algoritmi\}|$$

$\Rightarrow$ **esistono problemi privi di un corrispondente algoritmo di calcolo**. Per esempio, il problema dell'arresto.

Lezione di Turing *Non esistono algoritmi che decidono il comportamento di altri algoritmi esaminandoli dall'esterno, cioè senza passare dalla loro simulazione.*

1.4.2 Modelli di calcolo

La teoria della calcolabilità dipende dal modello di calcolo?
Oppure
la decidibilità è una proprietà del problema?

I linguaggi di programmazione esistenti sono tutti equivalenti?
Ce ne sono di alcuni più potenti/più semplici di altri?
Ci sono algoritmi descrivibili in un linguaggio ma non in un altro?
È possibile che problemi oggi irrisolvibili possano essere risolti in futuro con altri linguaggi o altri calcolatori?
Le teorie della calcolabilità e della complessità dipendono dal modello di calcolo?

Tesi di Church-Turing Tutti i *ragionevoli* modelli di calcolo **risolvono esattamente la stessa classe di problemi**, quindi **si equivalgono nella possibilità di risolvere i problemi** pur operando con diversa efficienza.

Tesi C-H: la decidibilità è una proprietà del problema

Incrementi qualitativi sui calcolatori o sui linguaggi di programmazione servono **solo** ad abbassare i tempi di esecuzione o rendere più agevole la programmazione.

1.4.3 Decidibilità e trattabilità

Ci sono quindi problemi che non possono essere risolti da nessun calcolatore, indipendentemente dal tempo impiegato (**problemi indecidibili**).
Ci sono poi problemi decidibili che possono richiedere tempi di risoluzione esponenziali nella dimensione dell'istanza (**problemi intrattabili**).
Ci sono poi problemi che possono essere risolti con algoritmi di costo polinomiale nella dimensione dell'inpu (**problemi trattabili**).
Abbiamo poi una famiglia di problemi il cui stato non è noto: clique (cricca), cammino hamiltoniano... Sappiamo risolverli (decidibili) con algoritmi di costo esponenziale, ma non abbiamo limiti inferiori esponenziali. I migliori limiti inferiori sono polinomiali: c'è un gap fra il limite inferiore (polinomiale) e costo della migliore soluzione a disposizione (esponenziale) (**presumibilmente intrattabili**).

Notazione

Studiamo la dimensione dei dati trattabili in funzione dell'incremento della velocità del calcolatori.
Dati i calcolatori C_1, C_2 (k volte più veloce di C_1) e tempo di calcolo a disposizione t, avrò n_1 dati trattabili in tempo t su C_1 e n_2 trattabili in tempo t su C_2.

Si **osserva** che **usare C_2 per un tempo t equivale a usare C_1 per un tempo $k \cdot t$.**

Algoritmi polinomiali Un algoritmo polinomiale che risolve il problema in $c \cdot n^s$ secondi, con c ed s costanti.

C_1 $c \cdot n_1^s = t \Rightarrow n_1 = (t/c)^{1/s}$

C_2 $c \cdot n_2^s = t \Rightarrow n_2 = k^{1/s} \cdot (t/c)^{1/s}$

$\Rightarrow$ $n_2 = k^{1/s} \cdot n_1$, miglioramento di un fattore **moltiplicativo** $k^{1/s}$

Algoritmi esponenziali Un algoritmo polinomiale che risolve il problema in $c \cdot 2^n$ secondi, con c costante.

C_1 $c2^{n_1} = t \Rightarrow 2^{n_1} = t/c$

C_2 $c2^{n_2} = k \cdot t \Rightarrow 2^{n_1} = k \cdot t/c = k2^{n_1}$

$\Rightarrow$ $n_2 = n_1 + \log_2(k)$, miglioramento di un fattore **additivo** $\log_2(k)$

Di conseguenza **un algoritmo efficiente è di gran lunga più importante di un calcolatore più potente**.

1.4.4 Tipologie di problemi

Dato un problema Π su un insieme di istanze in ingresso I con un insieme di soluzioni S.

Problemi decisionali Richiedono una risposta binaria $S = \{0, 1\}$, quindi istanze positive $x \in I \,|\, \Pi(x) = 1$ o negative $x \in I \mid \Pi(x) = 0$. Esempio: verifica se un numero è primo, o se un grafo è connesso.
La teoria della complessità computazionale è definita principalmente in termini di problemi di decisione: risposta binaria, quindi il tempo per restituire la risposta è costante, e la complessità di un problema è già presente nella versione decisionale.

Problemi di ricerca Data un'istanza x, richiede di restituire una soluzione s.

Problemi di ottimizzazione Data un'istanza x, si vuole trovare la **migliore** soluzione s tra tutte quelle possibili. Esempio: clique di dimensione massima, cammino minimo...

1.4.5 Classi di complessità

Dato un problema **decisionale** Π ed un algoritmo A, diciamo che A **risolve** Π se, data un'istanza di input x, $A(x) = 1 \Leftrightarrow \Pi(x) = 1$
A risolve Π in **tempo** $t(n)$ **e spazio** $s(n)$.

Classi Time e Space

Time($f(n)$): insieme dei **problemi decisionali che possono essere risolti in tempo O($f(n)$)**

Space($f(n)$): insieme dei **problemi decisionali che possono essere risolti in spazio O($f(n)$)**

Classe P Classe dei problemi risolvibili in **tempo** polinomiale nella dimensione dell'istanza di input.
Algoritmo polinomiale nel tempo: $\exists c, n_0 > 0 \mid$ il numero di passi elementari è al più n^c per ogni input di dimensione $n > n_0$.

Classe PSPACE Classe dei problemi risolvibili in **spazio** polinomiale nella dimensione dell'istanza di input. Molto più grande di P.
Algoritmo polinomiale nello spazio: $\exists\, c, n_0 > 0 \mid$ il numero di celle di memoria è al più n^c per ogni input di dimensione $n > n_0$.

Classe EXPTIME Classe dei problemi risolvibili in tempo esponenziale nella dimensione dell'istanza di input.

$$P \subseteq PSPACE \subseteq EXPTIME$$

Non è noto se queste inclusioni siano note, ad oggi. L'unico risultato dimostrato finora riguarda P ed EXPTIME: esiste un problema che può essere risolto in tempo esponenziale ma per cui il tempo polinomiale non è sufficiente (es: torri di Hanoi).

1.4.6 Certificato

Per alcuni problemi, per le istanze accettabili (istanze in cui la risposta del problema decisionale è si), è possibile certificare che quell'istanza è accettabile con un certificato y che può convincerci dell'accettabilità.
Per clique, il certificato è il sottoinsieme di k vertici che forma la clique. Per il cammino hamiltoniano è la permutazione degli n vertici che formano il cammino. Per SAT, sono le assegnazioni che rendono vera la formula. Il certificato ha dimensione polinomiale (k, n) e la verifica del certificato è lineare.
Una volta che ho il certificato lo vado a verificare: attestato breve di esistenza di una soluzione con determinate proprietà. Si definisce solo per istanze accettabili, perché spesso la non accettabilità non è facile costruire un certificato.

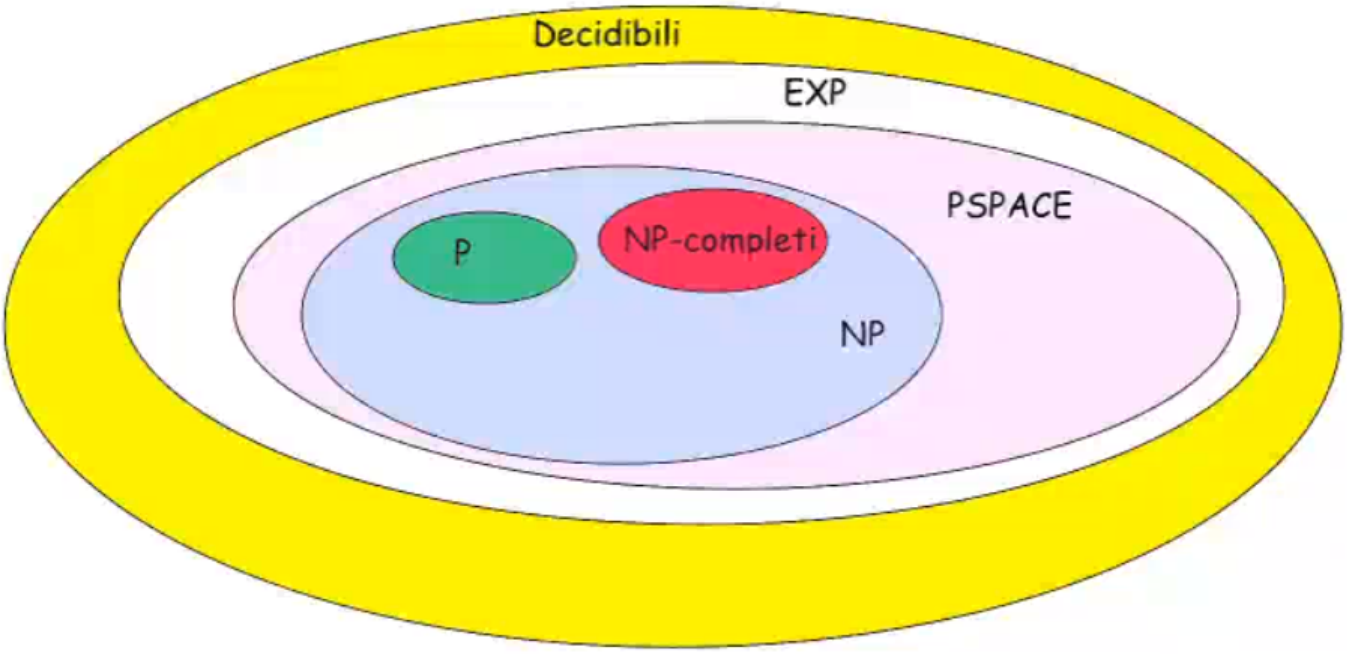
Idea Usare il costo della verifica di un certificato per un'istanza accettabile per **caratterizzare la complessità del problema** stesso.
Un problema è **verificabile in tempo polinomiale** se: tutte le istanze accettabili ammettono un certificato di lunghezza polinomiale ed esiste un algoritmo di verifica polinomiale in n.

Classe NP Classe dei problemi decisionali **verificabili in tempo polinomiale**. (NP = polinomiale su macchine non deterministiche)

P $\subset$ NP? Ovviamente si, ogni problema in P ammette un certificato verificabile in tempo polinomiale: eseguo l'algoritmo che risolve il problema per costruire il certificato.
Quello che non sappiamo è $P = NP$ oppure $P \neq NP$. Si pensa $P \neq NP$.
Si possono individuare i problemi più difficili in NP, ovvero quelli candidati ad appartenere ad NP se $P \neq NP$: sono i problemi NP-completi, quelli per cui se esiste un algoritmo polinomiale per risolvere un NP-completo allora tutti i problemi NP potrebbero essere risolti in tempo polinomiale e quindi P = NP.
Quindi tutti i problemi NP-completi sono risolvibili in tempo polinomiale oppure nessuno lo è.

Tutti i problemi NP-completi possono essere ridotti l'un l'altro, sono NP-equivalenti.

Gerarchia delle classi secondo le attuali congetture

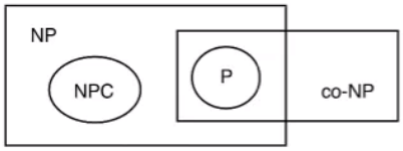


La fattorizzazione ad esempio $\in NP - (P \cup NPcompleti)$, infatti è risolto in tempo polinomiale su macchine quantistiche.

1.4.7 Classi co-P e co-NP

C'è molta differenza tra certificare l'esistenza e certificare la non esistenza di una soluzione. Dato un problema Π possiamo definire coΠ che accetta tutte e sole le istanze rifiutate da Π.
La classe coP è la classe per cui coΠ ∈ P. P = coP, i problemi complementari e i co-complementari (originali) si possono entrambi risolvere in tempo polinomiale: risolvo il problema complementare e complemento il risultato.

Questo non vale per coNP, la classe per cui coΠ ∈ NP. Si congettura che siano diverse, se la congettura è vera allora $P \neq NP$



Capitolo 2

Sequenze Casuali

2.1 Esempi di algoritmi numerici

Algoritmo di Euclide per il calcolo dell'MCD fra due interi Suppongo due interi a b a maggug b a magg 0 b maggug 0
MC(a,b) se b è 0 allora ritorno A altrimenti ritorno MCD(b, a mod b).

Valutazione complessità Istanza di input: a, b. Rappresentati ad esempio in base due, quindi la dimensione n dell'istanza di input I, cioè $n = |I| = \Theta(\log a + \log b) = \Theta(\log a)$.
Algoritmo ricorsivo, quindi bisogna valutare il numero delle chiamate ricorsive, che dipenderanno dai dati. Istanze in cui si termina subito (Es. a multiplo di b, cioè a mod b = 0). Il numero di chiamate cresce con log a, perché a mod b rimpiazza a.
Si osserva che a mod b $<$ a/2, questo perché a = qb + a mod b e siccome per ipotesi a maggug b allora b maggug 1 e lo è anche q, quindi a maggug b + a mod b magg a mod b + a mod b perché b magg a mod b. Quindi 2(a mod b) min a, quindi a mod b min a/2.
Prima chiamata su a, b. Seconda su b, a mod b.
Terza su (a mod b), (b mod (a mod b)).
Quindi ad ogni chiamata a si riduce almeno della metà, e lo possiamo fare al massimo $\log a$ volte, quindi avrò $O(\log a)$ ricorsioni.

Il costo del calcolo del modulo è $O(\log a \cdot \log b) = O(\log^2 a)$
Complessivamente $T(n) = O(\log^3 a) = O(n^3)$ **polinomiale nella dimensione dell'istanza** $|I|$ **(cioè nel numero di cifre), polilogaritmico nel valore dei dati**

Test di primalità Versione inefficiente.
Primo(N) per i da 2 minugu rad N i++, if N% i == 0 return false, altrimenti a fine ciclo restituisco true.
Uso la proprietà che se n non è primo ha almeno un divisore minugu rad n.

Valutazione di complessità $I = N$, $|I| = \Theta(N) = n$
Ho $\sqrt{N}$ iterazioni, ciascuna di costo $\Theta(\log^2 N)$

$T(n) = O(\sqrt{N} \cdot \log^2 N) = O(2^{\frac{n}{2}} \cdot n^2)$ **pseudopolinomiale**, cioè **polinomiale nel valore di N ma esponenziale nella dimensione** $|I| = n$.

2.2 Casualità

Problema Data una sequenza binaria, vogliamo capire se è una sequenza casuale o meno. Le sequenze casuali sono importanti primo per la generazioni delle classi, secondo perché in crittografia spesso si ricorrono ad algoritmi randomizzati che usano sequenze casuali per funzionare.

Significato algoritmico della casualità Vedendo la teoria di Kolmogorov. Prendiamo due sequenze

h: 1111...1 lunga n

h': 10110110101011010100101...0

La prima è molto facile da descrivere (*scrivi n "uni"*), descrivere la seconda è molto meno pratico: l'intuizione è che una sequenza casuale non si può descrivere in modo compatto.
Ponendo $n = 20$, la probabilità di generare h è $P(h = (1/20)^{20}$, $P(h') = (1/20)^{20}$ (1/2 per generare 1, 1/2 per generare $0\dots$).
A_h algoritmo che genera h. Formalizzabile semplicemente (*genera n uni*)
$|A_h| = \#$ bit di A_h codificato in binario $= \log n +$ const (la parte costante è la generazione, varia solo n) Con $\log n$ bit ne abbiamo descritti n.
$A_{h'} =$ print h', $|A_{h'}| > |n| = |h'|$

L'intuizione è **una sequenza binaria è casuale se non ammette un algoritmo di generazione la cui rappresentazione binaria sia più corta di** h. Se posso usare meno bit vuol dire che la sequenza ha una qualche regolarità.

Sistemi di calcolo Sono un infinità numerabile $S_1 \dots S_i \dots$
Prendiamo S_i. p programma che genera la sequenza h nel sistema S_i, cioè $S_i(p) = h$
Def: **la complessità di Kolmogorov di** h **nel sistema** S_i **è** $K_{S_i}(h) = min\{\ |p|\ \mid S_i(p) = h\}$

Se la sequenza h non segue alcuna legge semplice di regolarità, allora il più breve programma in grado di generarla dovrà contenerla al suo interno, cioè sarà almeno lungo quanto la sequenza stessa generandola trasferendola in output. Quindi $K_{S_i}(h) = |h| + \text{const}_i$. La costante è la parte di programma che trasferisce in output, dipende da S_i ma non ha h.

Tra tutti i sistemi di calcolo possibili ne esiste uno **universale** in grado di simulare tutti gli altri, lo chiamiamo S_u e lo prendiamo in considerazione.
Supponiamo $p \mid S_i(p) = h$, allora $S_u(\langle i, p\rangle) = S_i(p) = h$. Ottengo $q = \langle i, p\rangle$ programma che genera h in S_u

$|q| = |\langle i, p\rangle| = |i| + |p| = \log_2 i + |p|$ quindi la lunghezza di q dipende da i ma non da h.
$\forall\, h\ \forall\, i\ \ K_{S_u}(h) \leq K_{S_i}(h) + C_i$
L'uguale vale per le sequenze generate per simulazione di S_i non essendoci per S_u algoritmi più "brevi". Il minore vale per sequenze generabili con programmi più corti (ad esempio per simulazione su un altro sistema $S_j \neq S_i$).

Def La complessita di Kolmogorov di una sequenza h è $K(h) = K_{S_u}(h)$

2.2.1 Sequenze casuali

Sequenza casuale Una sequenza h è casuale se $K(h) \geq |h| - \lceil \log_2 h \rceil$
Non entra in gioco come genero la sequenza, la **casualità è una proprietà della sequenza**.

Conteggio delle sequenze $\forall\, n, \exists$ sequenze casuali (secondo Kolmogorov) di lunghezza n
Dim: n, $S = 2^n$ # sequenze binarie di lunghezza n
T = # sequenze di lunghezza n NON casuali. L'obiettivo è dimostrare che T < S.
$\dots \Rightarrow T \leq N < S \Rightarrow T < S$
Quindi non solo esistono ma sono anche la **maggioranza**, essendo enormemente più numerose di quelle non casuali. Lo vediamo studiando il rapporto

$$\frac{T}{S} \leq \frac{N}{S} = \frac{2^{n-\lceil \log n \rceil}}{2^n} - \frac{1}{2^n} < \frac{1}{2^{\lceil \log n \rceil}} \quad \lim_{n \to +\infty} \frac{T}{S} = 0$$

Stabilire la casualità Data una sequenza arbitraria di lunghezza n, stabilire se è casuale secondo Kolmogorov è un problema **indecidibile**.
Dim: per assurdo suppongo esista un algoritmo Random | Random(h) = 1 se h è casuale, 0 altrimenti. Possiamo costruire l'algoritmo Paradosso che enumera tutte le possibili sequenze binarie in ordine crescente di lunghezza.
Paradosso: for binary h da 1 → infty do (if $|h| - \lceil \log |h| \rceil > |P|$ && Random(h) == 1 return h)
P è una seq binaria che rappresenta la codifica del programma Paradosso e Random. Rappresenta il programma complessivo Paradosso + Random, quindi $|P| = |\text{Paradosso}| + |\text{Random}|$, costante che non dipende da h, perché la sequenza h non compare in P ma solo come nome di variabile. Il valore rimane registrato fuori dal programma.
Paradosso quindi restituisce come risultato la prima sequenza casuale | $|h| - \lceil \log_2 |h| \rceil > |P|$
Quindi $\exists$ sequenze casuali di qualsiasi lunghezza, quindi certamente ne esisterà una che soddisfa entrambe le condizioni dell'if, che viene generata.

Ma la prima condizione mi dice che il programma rappresentato da P è breve e genera h, quindi h non è casuale perché prodotta con un programma breve.
Quindi $K(h) \leq |P|$, cioè P genera h) ma $|P| < |h|l\lceil \log_2 |h| \rceil$ quindi h non è casuale.
Ma la seconda condizione dice h casuale, giungendo ad un **paradosso** dato dall'assumere l'esistenza di Random.

2.2.2 Sorgente binaria casuale

Generatore Genera una sequenza di bit con queste proprietà:

1. P(0) = P(1) = 1/2, cioè genera 1 o 0 a pari probabilità.
   Si può indebolire richiedendo P(0) > 0, P(1) > 0 immutabili nel tempo della generazione.

2. La generazione di un bit è indipendente dalla generazione degli altri.
   ⇒ non si può prevedere il valore di un bit osservando quelli già generati.

Perché possiamo indebolire la prima proprietà? Supponiamo di essere in un caso in cui P(0) > P(1), allora è **sempre possibile bilanciare la sequenza**.
Supponiamo di generare 001100111000010100 e si **dividono a coppie** 00 11 00 11 10 00 01 01 00 e si scartano le coppie uguali. Si associano le coppie miste, ad esempio 01 → 0 e 10 → 1. Si presentano in modo equiprobabile, quindi la sequenza si ribilancia ottenendo 100 (caso poco significativo perché sequenza corta).

Esistono queste sorgenti? Non si sa. Nella pratica non è possibile garantire la perfetta casualità o l'indipendenza. Quindi sfrutteremo le casualità presenti in processi fisici o processi software.

2.2.3 Generatore di sequenze brevi

Fenomeni casuali presenti in natura Ad esempio il rumore su un microfono o il tempo di decadimento di alcune particelle, sfruttabili come sorgenti di casualità.
Il problema di questo approccio è che bisogna non avere accesso fisico ai dispositivi usati (es: microfono manomesso), oltre alla difficoltà pratica di usare certe sorgenti.

Processi software Come la temperatura, la posizione della testina del disco fisico...

Pseudocasuale Si genera la casualità **mediante un algoritmo**, **cercandola all'interno di processi matematici**. **Generatore di numeri pseudo-casuali**: ad esempio `rand()` del C.
Perché pseudocasuali? Perché sono algoritmi deterministici e brevi, quindi non risultano casuali secondo Kolmogorov.

Come funzionano? Partono da un *seed* (seme), breve sequenza che viene amplificata per creare una sequenza più lunga. Quindi un generatore è un **amplificatore di casualità**.

Input: seme (sequenza o valore breve)

Output: flusso di dati

Sequenze diverse: $2^s << 2^n$ # sequenze possibili

Generatore lineare $x_i = (a \cdot x_{i-1} + b) \bmod n$ con a, b, n interi positivi.
Il seme è un valore intero iniziale casuale x_0, quindi quando $x_i = x_0$ la sequenza si ripete.
Dobbiamo avere MCD$(b, n) = 1$, $(a-1)$ divisibile per ogni fattore primo di n e $(a-1)$ dev'essere un multiplo di 4 se anche n lo è.
Servono per garantire che il generatore produca una permutazione degli interi da 0 a $m-1$

2.3 Test statistici

Per valutare le sequenze prodotte da un generatore pseudocasuale.
Si valuta se la sequenza presenta le proprietà tipiche di una sequenza casuale:

test di frequenza

poker test: se sottosequenze siano distribuite in modo equo

test di autocorrelazione: verifica che non ci siano regolarità nella sequenza ottenuta

run test: verifica se sottosequenze massimali di elementi tutti ripetuti abbiano una distribuzione esponenziale negativa, cioè più sono lunghe meno sono frequenti.

Per le applicazioni crittografiche si richiede anche il **test di prossimo bit**, molto severo che implica tutti gli altri 4 test statistici. Intuitivamente, verifica che sia impossibile prevedere gli elementi della sequenza prima di generarli.

Test di prossimo bit Un generatore binario **supera** il test di prossimo bit **se non esiste un algoritmo polinomiale in grado di prevedere l'$i+1$-esimo bit della sequenza a partire dalla conoscenza degli i bit precedentemente generati con probabilità maggiore di 1/2**.
Quindi se si hanno a disposizione risorse polinomiale non si può prevedere il prossimo bit. I generatori che superano questo test sono detti **generatori crittograficamente sicuri**.

Generatore polinomiale Non è crittograficamente sicuro.

2.4 Generatori crittograficamente sicuri

Come costruire generatori crittograficamente sicuri Si ricorre alle **funzioni one-way**: funzioni facili da calcolare ma difficili da invertire, cioè **computabili in tempo polinomiale ($x \mapsto f(x)$)**, ma **computazionalmente difficile invertire la funzione ($y \mapsto x = f^{-1}(y)$)**. Come costruire queste funzioni?

Idea Con f one-way, scelgo il seme x_0. Genero S: $x\ f(x)\ f(x_1) = f(f(x)) \ldots x_i = f(f(\ldots(x_{i-1})\ldots))$
Cioè si itera l'applicazione della funzione one-way un numero arbitrario di volte. L'**idea è restituire la sequenza al contrario**, perché se conosco x_{i+1} non riesco a calcolare facilmente x_i.
Ogni elemento della sequenza S si può calcolare efficientemente con l'elemento precedente, ma non dai valori successivi perché f è one-way. Si calcola S per un certo numero di passi senza svelare il risultato e si comunicano gli elementi in ordine inverso. Ogni elemento non è prevedibile in tempo polinomiale pur conoscendo quelli comunicati.

Generatori binari crittograficamente sicuri Si usano i **"predicati *hard core*" delle funzioni one-way**.
Un predicato hard core di una funzione one-way $f(x)$ è $b(x)$ se $b(x)$ è facile da calcolare quando x è noto, ma è difficile da prevedere se si conosce $f(x)$.

Un esempio di funzione one-way è l'elevamento a quadrato in modulo ($f(x) = x^2 \bmod n$ con n non primo). Un predicato hard core è "$b(x) = x$ è dispari"

Generatore BBS