Programmazione

– Calcolatori e programmi –

Francesco Tiezzi

Scuola di Scienze e Tecnologie

Sezione di Informatica

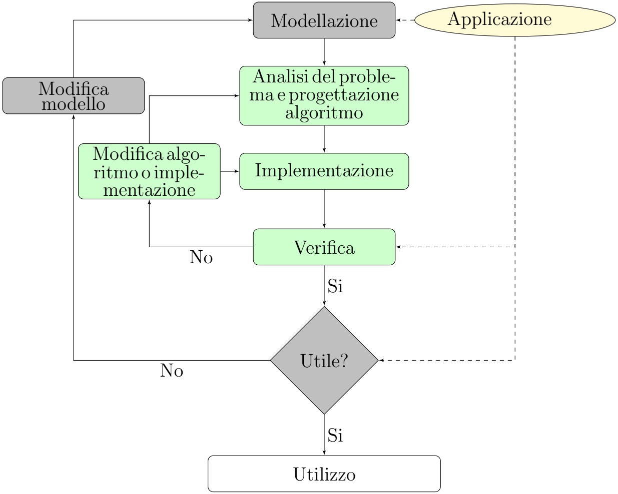
Università di Camerino

Lucidi originali di Pierluigi Crescenzi

Cosa è l'informatica?

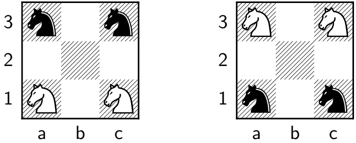
- Più facile dire cosa non è
  - Poco a vedere con "alfabetizzazione informatica" (saper usare un computer per scrivere un testo o navigare in Internet)
  - Non consiste semplicemente nello scrivere programmi
- Denning et al (1989)
  - *L'informatica è lo studio sistematico dei processi algoritmici che descrivono e trasformano l'informazione: la loro teoria, analisi, progettazione, efficienza, implementazione e applicazione*
  - Domanda fondamentale
    - *Che cosa può essere (efficientemente) automatizzato?*
- Metodo algoritmico
  - Formulare algoritmi che risolvano un problema
  - Trasformare questi algoritmi in programmi
  - Verificare la correttezza e l'efficacia di tali programmi analizzandoli ed eseguendoli

Il metodo algoritmico

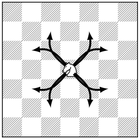


Il metodo algoritmico: trovare il giusto modello

- Qual è la sequenza di mosse più breve che consente ai cavalli di passare dalla configurazione a sinistra a quella a destra?

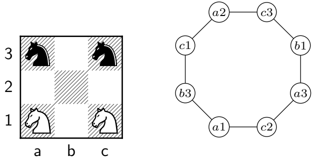


- Possibili mosse del cavallo



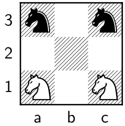
Il modello

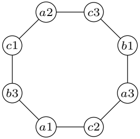
- Rappresentare il problema mediante una relazione di raggiungibilità



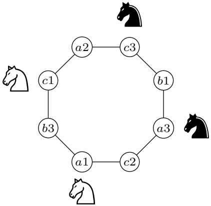
Il modello

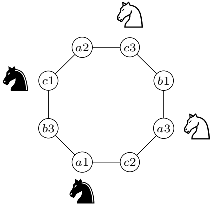
- Rappresentare il problema mediante una relazione di raggiungibilità





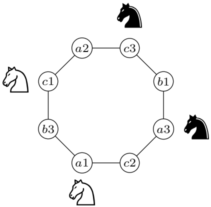
- Il problema diventa: trovare il minimo numero di mosse per andare dalla configurazione a sinistra a quella a destra

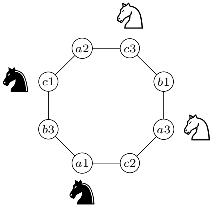




La soluzione: algoritmo

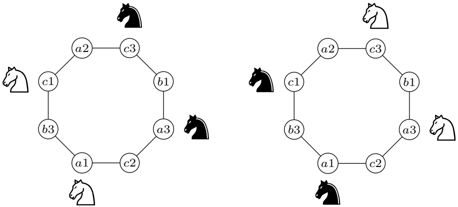
- Trovare il minimo numero di mosse per andare dalla configurazione a sinistra a quella a destra





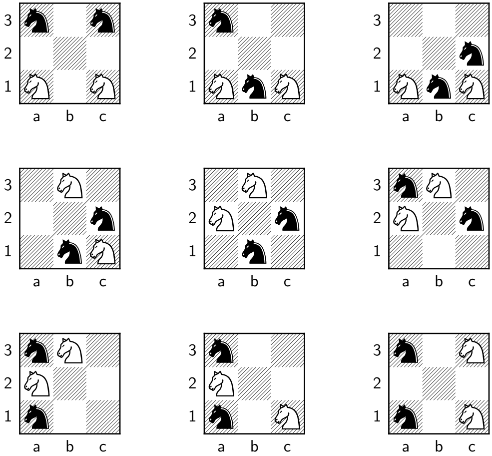
La soluzione: algoritmo

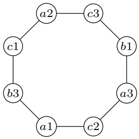
- Trovare il minimo numero di mosse per andare dalla configurazione a sinistra a quella a destra



- Ruotare i cavalli di quattro posizioni in senso orario (o antiorario)

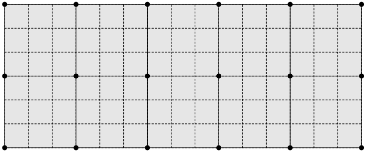
La soluzione: le prime 8 mosse





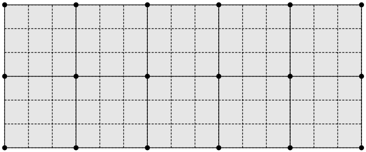
Il metodo algoritmico: trovare il giusto algoritmo

- Problema
  - Piastrellare una stanza rettangolare di dimensione $n \times m$ con il minor numero possibile di mattonelle quadrate di uguale dimensione
    - Esempio: $n = 6$ e $m = 15$



Il metodo algoritmico: trovare il giusto algoritmo

- Problema
  - Piastrellare una stanza rettangolare di dimensione $n \times m$ con il minor numero possibile di mattonelle quadrate di uguale dimensione
    - Esempio: $n = 6$ e $m = 15$

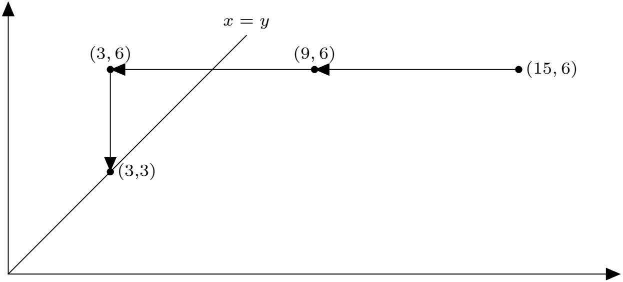


- Modello
  - Determinare il massimo numero intero che divide sia n che m
  - Calcolare il *massimo comun divisore* (MCD) di n e m
    - $MCD(6, 15) = 3$

- Primo algoritmo basato su definizione
  - Supponiamo che $n < m$ e che n non divide m
  - Esaminiamo tutti i numeri d tra $n/2$ e 2 (in ordine inverso)
    - Se d divide n e divide m, allora $MCD(n, m) = d$
  - $MCD(n, m) = 1$
- Esempio: $n = 111$ e $m = 259$
  - $n/2 = 55$
  - Tutti i numeri tra 55 e 37 non dividono 111
  - 37 divide $111 = 37 \times 3$ e divide $259 = 37 \times 7$
  - $MCD(111, 259) = 37$

- Primo algoritmo basato su definizione
  - Supponiamo che $n < m$ e che n non divide m
  - Esaminiamo tutti i numeri d tra $n/2$ e 2 (in ordine inverso)
    - Se d divide n e divide m, allora $MCD(n, m) = d$
  - $MCD(n, m) = 1$
- Esempio: $n = 111$ e $m = 259$
  - $n/2 = 55$
  - Tutti i numeri tra 55 e 37 non dividono 111
  - 37 divide $111 = 37 \times 3$ e divide $259 = 37 \times 7$
  - $MCD(111, 259) = 37$
- Caso pessimo
  - $MCD(n, m) = 1$
    - Bisogna provare tutti i numeri tra $n/2$ e 2

- Secondo algoritmo: formulazione geometrica



- Secondo algoritmo: formulazione algoritmica
  - Fintanto che $n \neq m$, se $n < m$ poni m uguale a $m - n$, altrimenti poni n uguale a $n - m$
  - Quando $n = m$, il loro valore è il MCD
- Correttezza
  - Segue dal fatto che, se $x > y$, $MCD(x, y) = MCD(x - y, y)$
    - Ogni numero che divide sia x che y, divide $x - y$
    - Ogni numero che divide sia $x - y$ che y, divide x

- Secondo algoritmo: formulazione algoritmica
  - Fintanto che $n \neq m$, se $n < m$ poni m uguale a $m - n$, altrimenti poni n uguale a $n - m$
  - Quando $n = m$, il loro valore è il MCD
- Correttezza
  - Segue dal fatto che, se $x > y$, $MCD(x, y) = MCD(x - y, y)$
    - Ogni numero che divide sia x che y, divide $x - y$
    - Ogni numero che divide sia $x - y$ che y, divide x
- Caso pessimo
  - n molto grande e m molto piccolo
  - Non molto diverso dal primo algoritmo

Algoritmo di Euclide

- Miglioramento rispetto al secondo algoritmo
  - Cercare di raggiungere un punto sull'asse delle ascisse, saltando direttamente al punto a esso più vicino

Algoritmo di Euclide

- Miglioramento rispetto al secondo algoritmo
  - Cercare di raggiungere un punto sull'asse delle ascisse, saltando direttamente al punto a esso più vicino
  - Fintanto che $n \neq 0$ e $m \neq 0$, se $n < m$ passa alla coppia $(n, m \mod n)$, altrimenti passa alla coppia $(m, n \mod m)$
    - $x \mod y$: resto della divisione di x per y

Algoritmo di Euclide

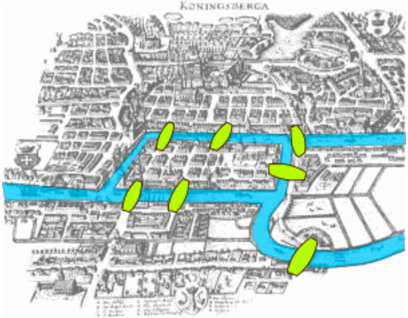
- Miglioramento rispetto al secondo algoritmo
  - Cercare di raggiungere un punto sull'asse delle ascisse, saltando direttamente al punto a esso più vicino
  - Fintanto che $n \neq 0$ e $m \neq 0$, se $n < m$ passa alla coppia $(n, m \mod n)$, altrimenti passa alla coppia $(m, n \mod m)$
    - $x \mod y$: resto della divisione di x per y
- Correttezza
  - Segue dal fatto che, se $x > y$, $MCD(x, y) = MCD(y, x \mod y)$
    - Se $r = x \mod y$, allora $x = qy + r$ e $r = x - qy$
    - Ogni numero che divide sia x che y, divide r
    - Ogni numero che divide sia y che r, divide x

Algoritmo di Euclide

- Miglioramento rispetto al secondo algoritmo
  - Cercare di raggiungere un punto sull'asse delle ascisse, saltando direttamente al punto a esso più vicino
  - Fintanto che $n \neq 0$ e $m \neq 0$, se $n < m$ passa alla coppia $(n, m \mod n)$, altrimenti passa alla coppia $(m, n \mod m)$
    - $x \mod y$: resto della divisione di x per y
- Correttezza
  - Segue dal fatto che, se $x > y$, $MCD(x, y) = MCD(y, x \mod y)$
    - Se $r = x \mod y$, allora $x = qy + r$ e $r = x - qy$
    - Ogni numero che divide sia x che y, divide r
    - Ogni numero che divide sia y che r, divide x
- Efficienza
  - Ottimale (ma esula da questo corso)

Il metodo algoritmico: un esempio più complesso

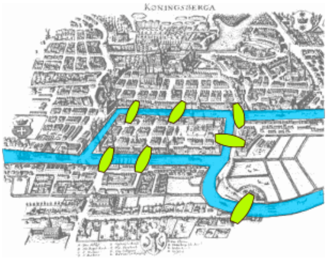
- Il problema dei ponti di Königsberg

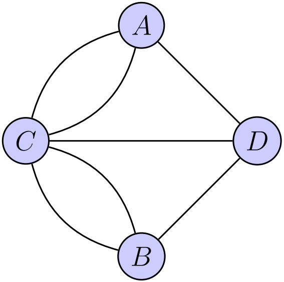


- Possibile con una passeggiata seguire un percorso che attraversi ogni ponte una e una volta soltanto e tornare al punto di partenza?

Il modello

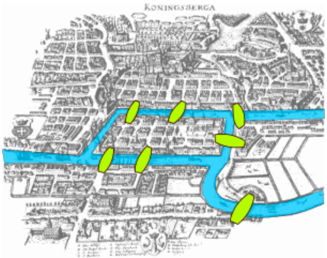
- Rappresentare relazione di connessione
  - Multigrafo

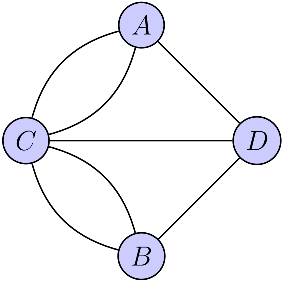




Il modello

- Rappresentare relazione di connessione
  - Multigrafo





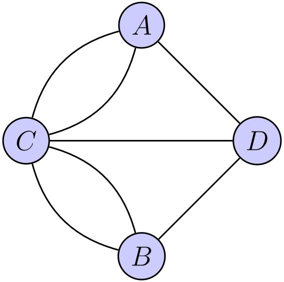
- Esiste un circuito che attraversa ogni arco una e una sola volta?
  - Esiste un circuito Euleriano?

Teorema di Eulero

- Esiste un circuito Euleriano se e solo se il multigrafo è connesso e ogni nodo ha grado pari
  - Grafo connesso: ogni nodo può raggiungere ogni altro nodo
  - Grado di un nodo: numero di archi incidenti

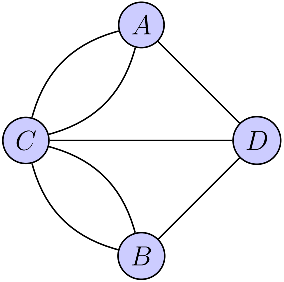
Teorema di Eulero

- Esiste un circuito Euleriano se e solo se il multigrafo è connesso e ogni nodo ha grado pari
    - Grafo connesso: ogni nodo può raggiungere ogni altro nodo
    - Grado di un nodo: numero di archi incidenti
- Multigrafo di Königsberg non ha circuito Euleriano



Teorema di Eulero

- Esiste un circuito Euleriano se e solo se il multigrafo è connesso e ogni nodo ha grado pari
  - Grafo connesso: ogni nodo può raggiungere ogni altro nodo
  - Grado di un nodo: numero di archi incidenti
- Multigrafo di Königsberg non ha circuito Euleriano

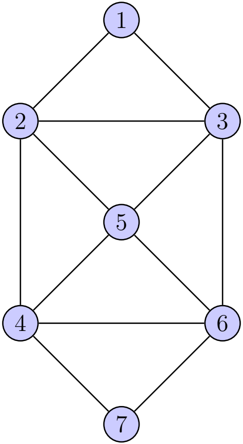


- Inizio della teoria dei grafi

Algoritmo

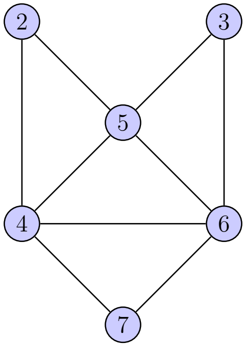
- Si parte da un vertice arbitrario e si percorre il grafo cancellando gli archi percorsi
  - Raggiunto un vertice si riparte da questo
- Il vertice iniziale diventa di grado 1 mentre i vertici successivamente toccati hanno il grado diminuito di 2 (cancellati o ancora pari)
- Archi e vertici sono finiti e si ritorna al vertice di partenza (unico dispari)
  - Si determina così un ciclo
- Si ripete fino a che possibile
  - Si cancellano dal grafo gli archi del ciclo, e se esistono ancora archi si sceglie un vertice del ciclo e si ripete il procedimento
  - Si determina un nuovo ciclo che si inserisce sul ciclo precedente

Esempio



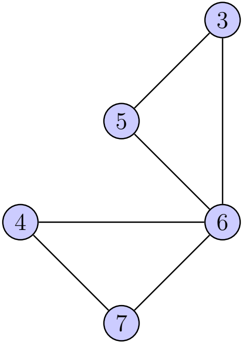
- Scegliendo sempre il vertice di indice più basso
  - Ciclo 1-2-3-1

Esempio



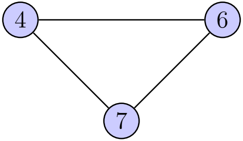
- Scegliendo sempre il vertice di indice più basso
  - Ciclo 1-2-3-1
  - Ciclo 2-4-5-2
    - Ciclo complessivo 1-2-4-5-2-3-1

Esempio



- Scegliendo sempre il vertice di indice più basso
  - Ciclo 1-2-3-1
  - Ciclo 2-4-5-2
    - Ciclo complessivo 1-2-4-5-2-3-1
  - Ciclo 3-5-6-3
    - Ciclo complessivo 1-2-4-5-2-3-5-6-3-1

Esempio



- Scegliendo sempre il vertice di indice più basso
  - Ciclo 1-2-3-1
  - Ciclo 2-4-5-2
    - Ciclo complessivo 1-2-4-5-2-3-1
  - Ciclo 3-5-6-3
    - Ciclo complessivo 1-2-4-5-2-3-5-6-3-1
  - Ciclo 4-6-7-4
    - Ciclo complessivo (e finale) 1-2-4-6-7-4-5-2-3-5-6-3-1

Algoritmi

- **Informatica**: studio sistematico dei processi algoritmici che descrivono e trasformano l'informazione: la loro teoria, analisi, progettazione, efficienza, implementazione e applicazione
- **Algoritmo**: successione finita di istruzioni o passi che definiscono le operazioni da eseguire su dei dati (che formano l'istanza di un problema) per ottenere dei risultati (intesi come la soluzione dell'istanza specificata)
  - Proprietà
    - Finito
    - Generale
    - Non ambiguo
    - Corretto
    - Efficiente
  - Esempio
    - Algoritmo di Euclide

Problemi indecidibili

- Non tutti i problemi (computazionali) ammettono algoritmi di risoluzione: **problema della fermata** (Turing, 1937)

Dato un generico algoritmo (o programma) A e dato un input x, A con x in ingresso **termina** o **va in ciclo**?

Problemi indecidibili

- Non tutti i problemi (computazionali) ammettono algoritmi di risoluzione: **problema della fermata** (Turing, 1937)

Dato un generico algoritmo (o programma) A e dato un input x, A con x in ingresso **termina** o **va in ciclo**?

- Esempio: algoritmo `P`
  - Con input n
    1. Pone $f = 2$
    2. Fintanto che $f < n$ e $n \mod f$ è diverso da 0, aumenta f di 1
    3. Se $f = n$, allora il numero è primo, altrimenti non lo è
- Per un numero n qualsiasi, `P(n)` termina?

Problemi indecidibili

- Non tutti i problemi (computazionali) ammettono algoritmi di risoluzione: **problema della fermata** (Turing, 1937)

Dato un generico algoritmo (o programma) A e dato un input x, A con x in ingresso **termina** o **va in ciclo**?

- Esempio: algoritmo `P`
  - Con input n
    1. Pone $f = 2$
    2. Fintanto che $f < n$ e $n \mod f$ è diverso da 0, aumenta f di 1
    3. Se $f = n$, allora il numero è primo, altrimenti non lo è
- Per un numero n qualsiasi, `P(n)` termina?
  - Si, perché f è aumentato di 1 a ogni ripetizione del passo 2 e a un certo punto deve divenire uguale a n

Congettura di Goldbach

- Formulata nel 1742
  - Ogni numero pari $n \geq 4$ è uguale alla somma di due numeri primi

Congettura di Goldbach

- Formulata nel 1742
  - Ogni numero pari $n \geq 4$ è uguale alla somma di due numeri primi
- Esempio: algoritmo `G`
  - Senza nessun input
    1. Pone $n = 2$
    2. Aumenta n di 2
    3. Per ogni p con $2 \leq p < n$ e $q = n - p$: se p e q sono primi vai al passo 2.
    4. La congettura è falsa
- Quest'algoritmo termina?

Congettura di Goldbach

- Formulata nel 1742
  - Ogni numero pari $n \geq 4$ è uguale alla somma di due numeri primi
- Esempio: algoritmo `G`
  - Senza nessun input
    1. Pone $n = 2$
    2. Aumenta n di 2
    3. Per ogni p con $2 \leq p < n$ e $q = n - p$: se p e q sono primi vai al passo 2.
    4. La congettura è falsa
- Quest'algoritmo termina?
  - Termina se e solo se trova $n \geq 4$ per cui non esistono due primi p e q t.c. $n = p + q$
  - Termina se e solo se la congettura di Goldbach è falsa (problema aperto)

Problema della fermata

- Un algoritmo è una sequenza di simboli
  - Un algoritmo può essere dato in pasto a un altro algoritmo

Problema della fermata

- Un algoritmo è una sequenza di simboli
  - Un algoritmo può essere dato in pasto a un altro algoritmo
- Supponiamo esista un algoritmo $\mathtt{T}(A, x)$ che, in tempo finito, risponde SI se $A(x)$ termina, risponde NO se va in ciclo
  - È legale invocare $\mathtt{T}(A, A)$
- Esempio: algoritmo `F`
  - Con input A
    1. Se $\mathtt{T}(A, A)$ risponde SI allora va in ciclo, altrimenti termina
- `F( F )` termina?

Problema della fermata

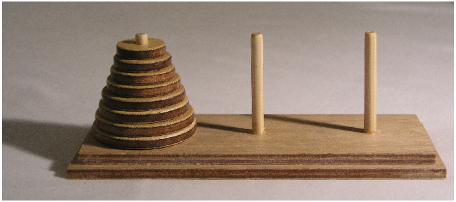
- Un algoritmo è una sequenza di simboli
  - Un algoritmo può essere dato in pasto a un altro algoritmo
- Supponiamo esista un algoritmo $\mathtt{T}(A, x)$ che, in tempo finito, risponde SI se $A(x)$ termina, risponde NO se va in ciclo
  - È legale invocare $\mathtt{T}(A, A)$
- Esempio: algoritmo `F`
  - Con input A
    1. Se $\mathtt{T}(A, A)$ risponde SI allora va in ciclo, altrimenti termina
- `F( F )` termina?
  - Se `F( F )` termina, allora $\mathtt{T}(\mathtt{F}, \mathtt{F})$ risponde NO, ovvero `F( F )` non termina
  - Se `F( F )` non termina, allora $\mathtt{T}(\mathtt{F}, \mathtt{F})$ risponde SI, ovvero `F( F )` termina
  - Contraddizione

Problema della fermata

- Un algoritmo è una sequenza di simboli
  - Un algoritmo può essere dato in pasto a un altro algoritmo
- Supponiamo esista un algoritmo $\mathtt{T}(A, x)$ che, in tempo finito, risponde SI se $A(x)$ termina, risponde NO se va in ciclo
  - È legale invocare $\mathtt{T}(A, A)$
- Esempio: algoritmo `F`
  - Con input A
    1. Se $\mathtt{T}(A, A)$ risponde SI allora va in ciclo, altrimenti termina
- `F( F )` termina?
  - Se `F( F )` termina, allora $\mathtt{T}(\mathtt{F}, \mathtt{F})$ risponde NO, ovvero `F( F )` non termina
  - Se `F( F )` non termina, allora $\mathtt{T}(\mathtt{F}, \mathtt{F})$ risponde SI, ovvero `F( F )` termina
  - Contraddizione
- Il problema della fermata è **indecidibile**
- Altri problemi lo sono: stabilire equivalenza tra 2 programmi

Torri di Hanoi

- 3 pioli
- $n = 64$ dischi sul primo piolo (vuoti gli altri due)
- Ogni mossa sposta un disco in cima a un piolo
- Un disco non può poggiare su uno più piccolo
- Spostare tutti i dischi dal primo al terzo piolo
  - Leggenda: finito lo spostamento, il mondo scomparirà



Torri di Hanoi

- 3 pioli
- $n = 64$ dischi sul primo piolo (vuoti gli altri due)
- Ogni mossa sposta un disco in cima a un piolo
- Un disco non può poggiare su uno più piccolo
- Spostare tutti i dischi dal primo al terzo piolo
  - Leggenda: finito lo spostamento, il mondo scomparirà
- Algoritmo `H` con input n, p, s, t
  1. Se $n = 1$ sposta il disco da p a t
  2. Altrimenti
     - 2.1 Esegue $\mathtt{H}(n-1, p, t, s)$
     - 2.2 Sposta un disco da p a t
     - 2.3 Esegue $\mathtt{H}(n-1, s, p, t)$

Torri di Hanoi

- 3 pioli
- $n = 64$ dischi sul primo piolo (vuoti gli altri due)
- Ogni mossa sposta un disco in cima a un piolo
- Un disco non può poggiare su uno più piccolo
- Spostare tutti i dischi dal primo al terzo piolo
  - Leggenda: finito lo spostamento, il mondo scomparirà
- Algoritmo `H` con input n, p, s, t
  1. Se $n = 1$ sposta il disco da p a t
  2. Altrimenti
     2.1 Esegue $\mathtt{H}(n-1, p, t, s)$
     2.2 Sposta un disco da p a t
     2.3 Esegue $\mathtt{H}(n-1, s, p, t)$
  - Termina
    - A ogni nuova esecuzione il numero di dischi è diminuito di 1
  - Quanti passi esegue?

Numero di mosse

- Se $n = 1$: 1

Numero di mosse

- Se $n = 1$: 1
- Se $n = 2$: 3

Numero di mosse

- Se $n = 1$: 1
- Se $n = 2$: 3
- Se $n = 3$: 7

Numero di mosse

- Se $n = 1$: 1
- Se $n = 2$: 3
- Se $n = 3$: 7
- Se $n = 4$: 15

Numero di mosse

- Se $n = 1$: 1
- Se $n = 2$: 3
- Se $n = 3$: 7
- Se $n = 4$: 15
- Se $n = 5$: 31

Numero di mosse

- Se $n = 1$: 1
- Se $n = 2$: 3
- Se $n = 3$: 7
- Se $n = 4$: 15
- Se $n = 5$: 31
- In generale: $2^n - 1$

Numero di mosse

- Se $n = 1$: 1
- Se $n = 2$: 3
- Se $n = 3$: 7
- Se $n = 4$: 15
- Se $n = 5$: 31
- In generale: $2^n - 1$
- Dimostrazione
  - Caso base $n = 1$: $2^1 - 1 = 1$
  - Passo induttivo: $(2^{n-1} - 1) + 1 + (2^{n-1} - 1) = 2^n - 1$

Numero di mosse

- Se $n = 1$: 1
- Se $n = 2$: 3
- Se $n = 3$: 7
- Se $n = 4$: 15
- Se $n = 5$: 31
- In generale: $2^n - 1$
- Dimostrazione
  - Caso base $n = 1$: $2^1 - 1 = 1$
  - Passo induttivo: $(2^{n-1} - 1) + 1 + (2^{n-1} - 1) = 2^n - 1$
- 1 mossa/sec: circa 585 miliardi di anni!

Tempo esponenziale $2^n - 1$

- 1 operazione/sec

n	5	10	15	20	25	30	35	40
tempo	31 s	17 m	9 h	12 g	1 a	34 a	1089 a	34865 a

- Aumentare di un fattore **moltiplicativo** X (ossia X operazioni/sec) migliora **solo** di un fattore **additivo**
  - Tempo: circa $2^{n-\log_2 X}$
    - Solo $\log_2 X$ dischi in più rispetto a 1 operazione/sec

Tempo esponenziale $2^n - 1$

- 1 operazione/sec

n	5	10	15	20	25	30	35	40
tempo	31 s	17 m	9 h	12 g	1 a	34 a	1089 a	34865 a

- Aumentare di un fattore **moltiplicativo** X (ossia X operazioni/sec) migliora **solo** di un fattore **additivo**
  - Tempo: circa $2^{n-\log_2 X}$
    - Solo $\log_2 X$ dischi in più rispetto a 1 operazione/sec
- Per quanto la tecnologia possa migliorare, la fine del mondo è lontana...

Ordinamento

Data una sequenza di n elementi e una loro relazione d'ordine $\leq$*, disporli in modo che risultino ordinati (per esempio, in modo crescente) secondo la relazione* $\leq$

- Milioni di applicazioni

Ordinamento

Data una sequenza di n elementi e una loro relazione d'ordine $\leq$*, disporli in modo che risultino ordinati (per esempio, in modo crescente) secondo la relazione* $\leq$

- Milioni di applicazioni
- Algoritmo di ordinamento per selezione
  - Esegue n passi
  - Passo i: **seleziona** il minimo tra i rimanenti $n - i + 1$ elementi e lo mette in posizione i

Ordinamento

Data una sequenza di n elementi e una loro relazione d'ordine $\leq$, disporli in modo che risultino ordinati (per esempio, in modo crescente) secondo la relazione $\leq$

- Milioni di applicazioni
- Algoritmo di ordinamento per selezione
  - Esegue n passi
  - Passo i: **seleziona** il minimo tra i rimanenti $n - i + 1$ elementi e lo mette in posizione i
- Analisi del tempo
  - n passi e passo i richiede $n - i + 1$ confronti

$$\sum_{i=1}^{n}(n - i + 1) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$$

  - Vedrete algoritmi più efficienti

Esponenziale versus polinomiale

- Due miliardi di operazioni al secondo

Esponenziale versus polinomiale

- Due miliardi di operazioni al secondo
- Esponenziale

n	10	20	30	40	50	60	70
tempo: 2^n	512ns	524μs	537ms	9m	7g	18a	187s

Esponenziale versus polinomiale

- Due miliardi di operazioni al secondo
- Esponenziale

n	10	20	30	40	50	60	70
tempo: 2^n	512ns	524μs	537ms	9m	7g	18a	187s

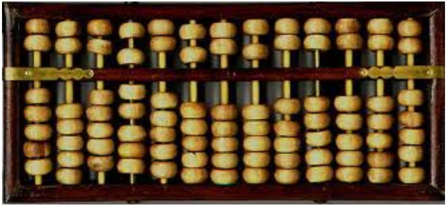
- Quadratico

n	10	20	30	40	50	60	70
tempo: n^2	50ns	200ns	450ns	800ns	1μs	2μs	3μs

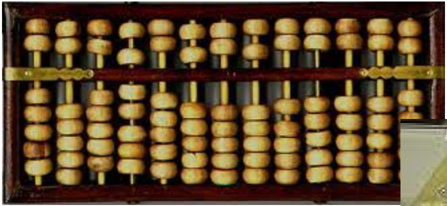
Nozioni di base

- Un calcolatore consiste di hardware e di software
  - **Hardware**: unità di elaborazione centrale, memoria principale, memoria ausiliaria, periferiche
  - **Software**: istruzioni raccolte in programma

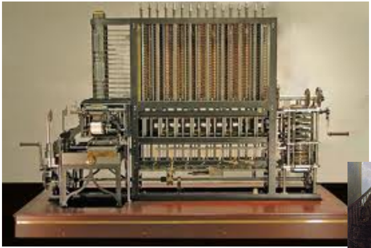
Primi strumenti di calcolo



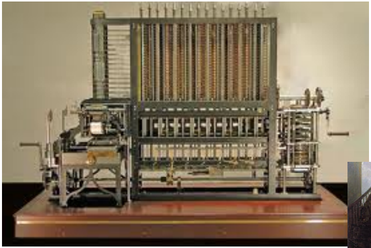
Primi strumenti di calcolo

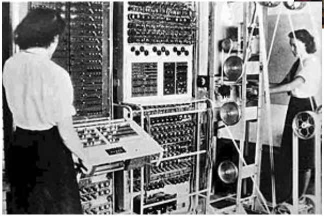


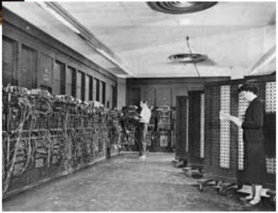
Primi calcolatori



Primi calcolatori

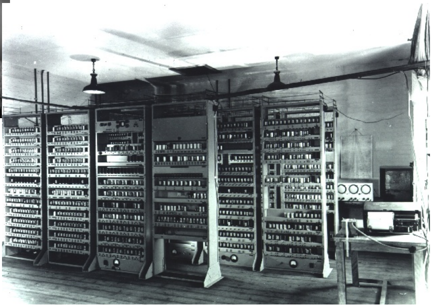




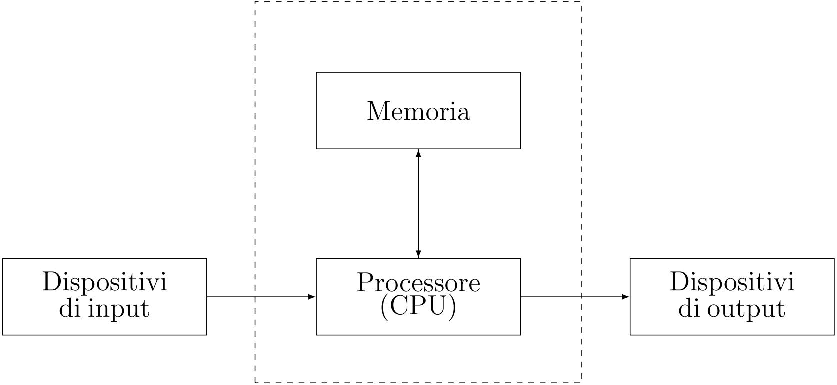


La macchina di Von Neumann

La macchina di Von Neumann



Componenti hardware principali



Componenti hardware principali

- **CPU**: dispositivo che esegue le istruzioni di un programma
  - Solo operazioni molto semplici, come trasferimento di un dato oppure operazioni aritmetiche elementari
- **Memoria principale**: veloce, ma costosa e volatile (conserva il programma attualmente in esecuzione ed i dati da esso usati)
- **Memoria ausiliaria**: meno costosa e che perdura anche in assenza di elettricità, ma più lenta (utilizzata per conservare programmi e dati in modo più o meno permanente)

- **Bit**: può assumere due soli valori (0 ed 1)
- **Byte**: pari a 8 bit (2^8 possibili valori)
- **Locazione di memoria**: sequenza di byte adiacenti associata al dato il cui indirizzo è l'indirizzo del primo byte di sequenza

INDIRIZZO	DATO	
...	...	...
484	00011110	Primo dato: 2 byte
485	00001001	
486	00000100	Secondo dato: 1 byte
487	01001100	Terzo dato: 4 byte
488	01111100	
489	01010101	
490	01001001	
491	01000111	Quarto dato: 2 byte
492	01001001	
...	...	...

Codice ASCII

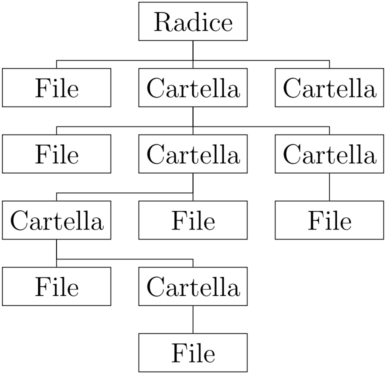
	000	001	010	011	100	101	110	111
0000		□	□	□	□	□	□	□
0001	□						□	□
0010	□	□	□	□	□	□	□	□
0011	□	□	□	□	□	□	□	□
0100		!	”	#	$	%	&	’
0101	(	)	*	+	,	-	.	/
0110	0	1	2	3	4	5	6	7
0111	8	9	:	;	<	=	>	?
1000	@	A	B	C	D	E	F	G
1001	H	I	J	K	L	M	N	O
1010	P	Q	R	S	T	U	V	W
1011	X	Y	Z	[	\	]	^	_
1100	‘	a	b	c	d	e	f	g
1101	h	i	j	k	l	m	n	o
1110	p	q	r	s	t	u	v	w
1111	x	y	z	{	|	}	˜	□

Numerazione binaria

- Come in quella decimale, posizione di una cifra indica valore relativo
  - Il sistema binario usa potenze crescenti di 2

Binario	Decimale	Binario	Decimale
0000	0	1000	8
0001	1	1001	9
0010	2	1010	10
0011	3	1011	11
0100	4	1100	12
0101	5	1101	13
0110	6	1110	14
0111	7	1111	15

- Notazione realmente usata: complemento a due


Radice
File
Cartella
Cartella
File
Cartella
Cartella
Cartella
File
File
File
Cartella
File

Algoritmi e programmi

- **Informatica**: fusione di *informazione* e *automatica*
  - Studio sistematico degli algoritmi che descrivono e trasformano l'informazione: la loro teoria, analisi, progetto, efficienza, realizzazione e applicazione
- **Algoritmo**: successione finita di istruzioni o passi che definiscono le operazioni da eseguire su dei dati (che formano l'istanza di un problema) per ottenere dei risultati (intesi come la soluzione dell'istanza specificata)
  - Interagisce con un ambiente esterno dal quale acquisisce dei dati e verso il quale comunica dati o messaggi

Equazione di primo grado

1. Inizio dell'algoritmo $\{$
2. leggi i coefficienti a e b;
3. se $a \neq 0$, $x = -b/a$; vai a 6;
4. se $b \neq 0$, comunica che l'equazione è impossibile; vai a 7;
5. comunica che l'equazione è indeterminata; vai a 7;
6. comunica il valore di x;
7. $\}$ Fine dell'algoritmo

Equazione di secondo grado

1. Inizio dell'algoritmo {
2. leggi i coefficienti a, b e c;
3. $\Delta = b^2 - 4ac$;
4. se $\Delta < 0$, comunica che l'equazione è impossibile; vai a 7;
5. $x_1 = \frac{-b-\sqrt{\Delta}}{2a}$ e $x_2 = \frac{-b+\sqrt{\Delta}}{2a}$;
6. comunica i valori di x_1 e x_2;
7. } Fine dell'algoritmo

Somma di numeri interi

1. Inizio dell'algoritmo $\{$
2. leggi n;
3. $i = 0$ e $s = 0$;
4. se $i > n$, vai a 6;
5. aggiungi i a s e incrementa i di 1; vai a 4;
6. comunica il valore di s;
7. $\}$ Fine dell'algoritmo

Somma di numeri interi

1. Inizio dell'algoritmo $\{$
2. leggi n;
3. $i = 0$ e $s = 0$;
4. se $i > n$, vai a 6;
5. aggiungi i a s e incrementa i di 1; vai a 4;
6. comunica il valore di s;
7. $\}$ Fine dell'algoritmo

Formula di Gauss

1. Inizio dell'algoritmo $\{$
2. leggi n;
3. $s = \frac{n(n+1)}{2}$; comunica il valore di s;
4. $\}$ Fine dell'algoritmo

Il problema delle 12 monete

- Di 12 monete, una (e una sola) è falsa (peso diverso)
- Problema: disponendo di bilancia a 2 piatti, individuare moneta falsa e stabilire se più pesante o più leggera

Il problema delle 12 monete

- Di 12 monete, una (e una sola) è falsa (peso diverso)
- Problema: disponendo di bilancia a 2 piatti, individuare moneta falsa e stabilire se più pesante o più leggera
  - Numero possibili soluzioni: 24

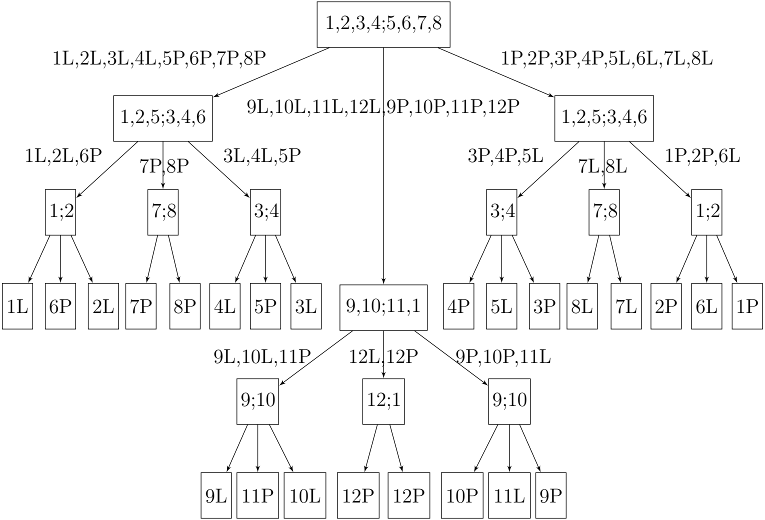
Il problema delle 12 monete

- Di 12 monete, una (e una sola) è falsa (peso diverso)
- Problema: disponendo di bilancia a 2 piatti, individuare moneta falsa e stabilire se più pesante o più leggera
  - Numero possibili soluzioni: 24
  - Ogni pesata genera 3 alternative
    - 1 pesata distingue fra 3 situazioni differenti
    - 2 pesate possono distinguere tra 9 situazioni differenti
    - 3 pesate possono distinguere tra 27 situazioni differenti
  - Non si può risolvere il problema con meno di 3 pesate
  - Esiste un algoritmo che impieghi effettivamente 3 pesate?

Il problema delle 12 monete

- Di 12 monete, una (e una sola) è falsa (peso diverso)
- Problema: disponendo di bilancia a 2 piatti, individuare moneta falsa e stabilire se più pesante o più leggera
  - Numero possibili soluzioni: 24
  - Ogni pesata genera 3 alternative
    - 1 pesata distingue fra 3 situazioni differenti
    - 2 pesate possono distinguere tra 9 situazioni differenti
    - 3 pesate possono distinguere tra 27 situazioni differenti
  - Non si può risolvere il problema con meno di 3 pesate
  - Esiste un algoritmo che impieghi effettivamente 3 pesate?
    - Confrontare inizialmente 2 monete non porta a soluzione: 20 soluzioni con 2 pesate
    - Confrontare due coppie di monete: 16 soluzioni con 2 pesate
    - Confrontare terne di monete: 12 soluzioni con 2 pesate

Soluzione al problema delle 12 monete

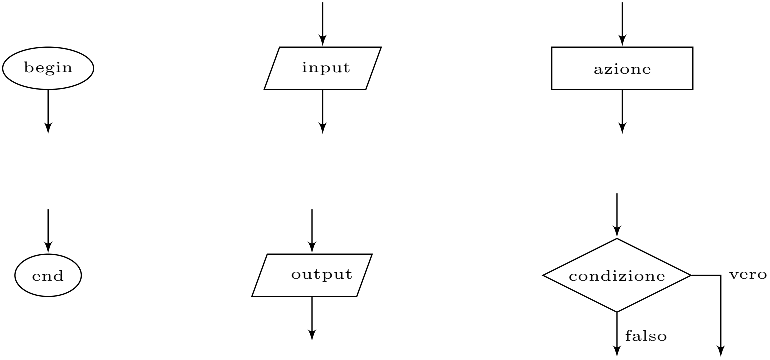


Proprietà degli algoritmi

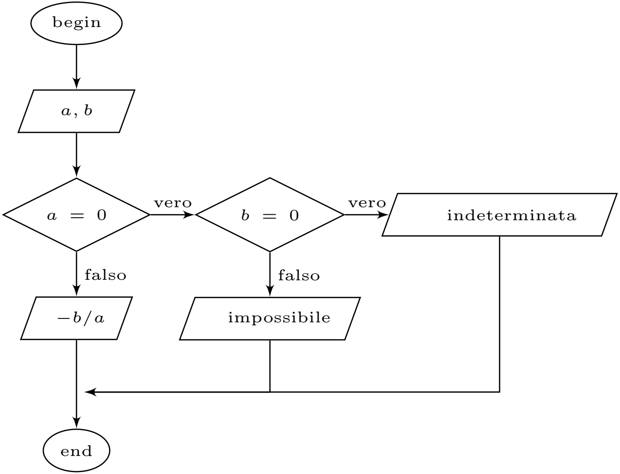
- Un algoritmo deve essere
  - Finito
  - Generale
  - Non ambiguo
  - Corretto
  - Efficiente

Descrizione degli algoritmi

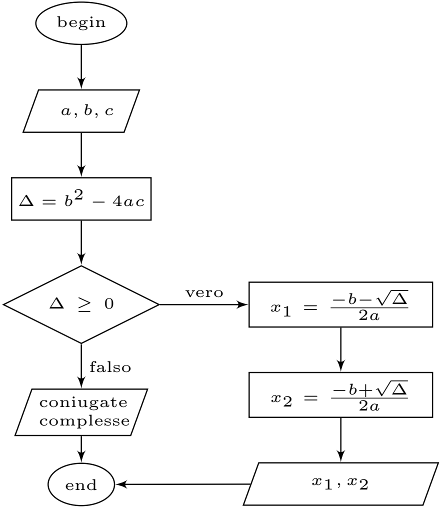
- Linguaggio di programmazione
- Pseudo-codice
- Diagrammi a blocchi



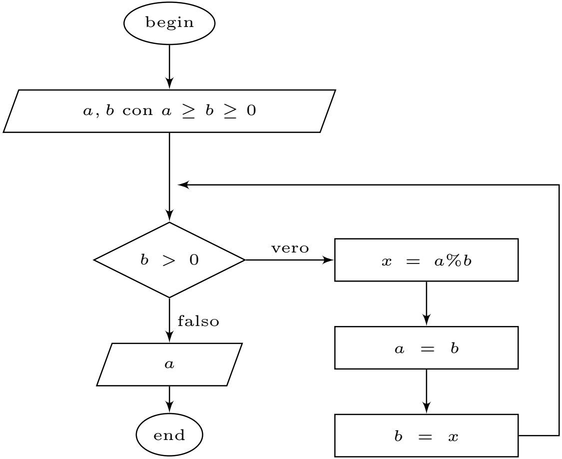
Diagrammi a blocchi ed equazioni di primo grado



Diagrammi a blocchi ed equazioni di secondo grado



Diagrammi a blocchi e algoritmo di Euclide

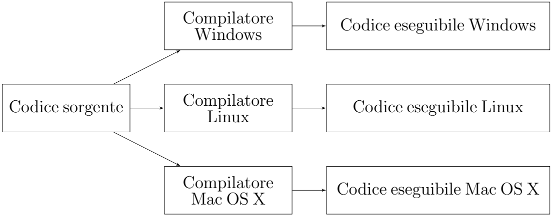


Programmi

- Programma: algoritmo scritto in linguaggio di programmazione
- Opera su
  - Dati in input
  - Dati di supporto
- Produce dati in output
- Diversi tipi
  - Sistema operativo
  - Programmi applicativi
    - Già esistenti
    - Creati dall'utente

- **Linguaggio macchina**
  - Consiste di sequenze di 0 e 1
  - Eseguito direttamente dal calcolatore
- **Linguaggio assembler**
  - Di tipo simbolico
  - Richiede una traduzione aggiuntiva molto semplice
- **Linguaggio ad alto livello**
  - Compromesso tra linguaggio naturale e linguaggio macchina
  - Esempi: FORTRAN, ALGOL, COBOL, LISP, APL, PROLOG, BASIC, Pascal, C, Ada, C++, Java, Python
- Esempio: somma di due numeri
  - Linguaggio macchina: 00000000010000011000001000000100000
  - Linguaggio assembler: `add $3, $2, $1`
  - Linguaggio ad alto livello: `c = a+b;`

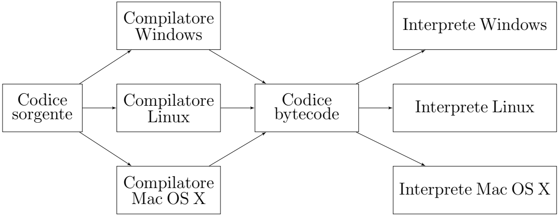
- **Compilatore**: traduce programma in linguaggio ad alto livello in programma in linguaggio macchina (più o meno)



- **Interprete**: traduce ed esegue una dopo l'altra istruzioni programma sorgente

L'approccio di Java

- Uso di codice intermedio detto byte-code
  - Linguaggio macchina di calcolatore virtuale



- Principale vantaggio: portabilità