Introduzione alla Complessità Computazionale

Docente: Renato Bruni

bruni@dis.uniroma1.it

Corso di: Ottimizzazione Combinatoria

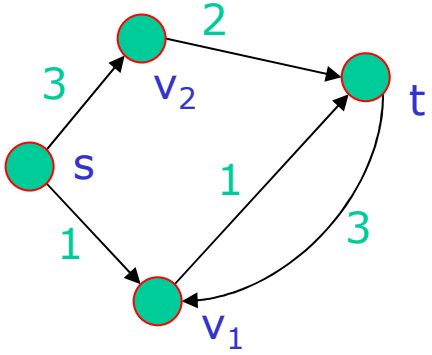
Il materiale presentato è derivato da quello del prof. A. Sassano

Istanze e Soluzioni di un Problema

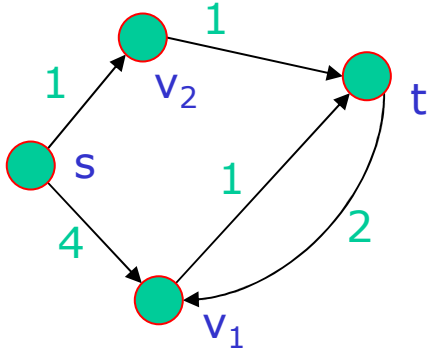
- **Risolvere un problema:** *Funzione F:* $\mathcal{I} \rightarrow \mathcal{S}$
  $\mathcal{I}$: *Insieme delle Istanze* (***INPUT***)
  $\mathcal{S}$: *Insieme delle Soluzioni* (***OUTPUT***)
  $F(I)$: *Soluzione associata all'Istanza* $I \in \mathcal{I}$

- **ISTANZA:** *Insieme delle informazioni di ingresso (dati):*
  Es. Un grafo G(N,A), le "lunghezze" degli archi e due nodi {s,t}

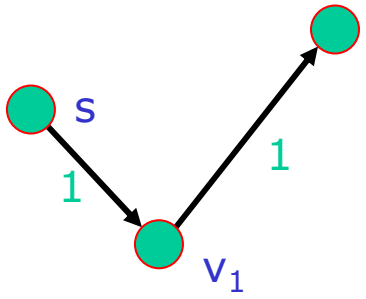
- **SOLUZIONE:** *Una particolare "proprietà" F(I) dell'istanza I*
  Es. Un Cammino di "lunghezza" minima da s a t su G(N,A)



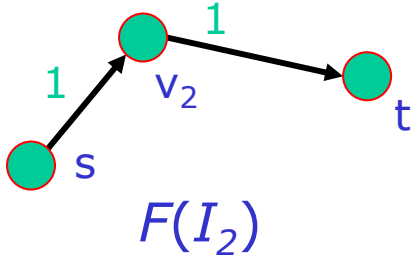
I_1



I_2



$F(I_1)$



$F(I_2)$

Algoritmi

- ALGORITMO:
  Procedura in grado di individuare, in PASSI SUCCESSIVI, una SOLUZIONE per una generica ISTANZA di un PROBLEMA.

Istanza e Soluzione debbono essere opportunamente codificate
(*ad es. univocamente associate a stringhe di "bit" (0,1)*)

- PROPRIETA' DI UN ALGORITMO
  - *CORRETTEZZA: applicato ad $I \in \mathcal{I}$ produce sempre $F(I)$*
  - *EFFICIENZA: in termini di risorse: SPAZIO e TEMPO*

✓ SPAZIO: Numero di bit necessari a memorizzare $I \in \mathcal{I}$ e a lavorare fino a produrre $F(I)$

✓ TEMPO: Numero di passi necessari fino a produrre $F(I)$

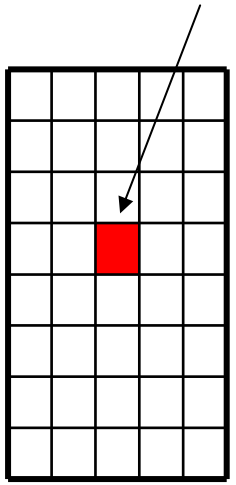
Come si contano i "passi" di un algoritmo ?

.. e come si misura l'efficienza ?

Ovviamente non possiamo calcolarli praticamente usando un computer:
oltre a essere molto complicato, i risultati cambierebbero cambiando computer…

Conteggio dei Passi

Serve un MODELLO (idealizzato) DI CALCOLO (= di calcolatore):
Ad esempio **Macchina ad accesso diretto** (Random Access Machine)



Questa macchina effettua operazioni primitive
su *celle* (parole) che contengono un numero costante
di "bit" (es. 32 o 64)

Ogni *operazione "primitiva"* richiede 1 passo

OPERAZIONI PRIMITIVE

- Operazioni logico-aritmetiche (+,-,*,/,if,..)
- Accesso a matrici e vettori (A[23], B[12,23])
- Assegnazione di valori a variabili (A[i]:=34)
- Chiamate di funzioni e "sub-routine"
- Confronti tra numeri (A[i]>A[j])
- ...

✓ I "loop" (for,while,repeat,..) richiedono un numero di passi
che è funzione della configurazione dei dati trattati

Esempio di Conteggio

- **PROBLEMA**: CALCOLO DELL'ELEMENTO MASSIMO IN UN VETTORE CON *d* ELEMENTI

  INSIEME DELLE ISTANZE $\mathcal{I}$: Vettori *A*[ ] con *d* elementi interi

  INSIEME DELLE SOLUZIONI $\mathcal{S}$: Insieme degli interi

ALGORITMO

- Max := *A*[1] [2 passi – accesso vettore+assegnazione]
- For *i*:=2 to *d* [2 passi – confronto (*i* con *d*)+incremento di *i*]

  do begin

  if *A*[*i*]> Max [2 passi – accesso vettore+confronto]

  then Max := *A*[*i*] [2 passi – accesso vettore+assegnazione]

  end;

Solo se *A*[*i*]>Max

d-1 volte !

NUMERO TOTALE PASSI (TEMPO) T :

$2+4(d-1) \leq T \leq 2+6(d-1)$

Il valore esatto dipende dall'Istanza !

Tempo di esecuzione di un Algoritmo

- **TEMPO DI ESECUZIONE** = *Numero di passi necessari ad un algoritmo per determinare la soluzione associata ad un'istanza*

DIPENDE DALLA:

✓ *Dimensione size(I) dell'Istanza* (es. # componenti del vettore)

✓ *Specifica Istanza* (es. valori delle componenti del vettore)

- **COMPLESSITA' DI UN ALGORITMO** = *Numero di passi necessari per determinare la soluzione di una generica istanza di dimensione size(I)*

↳ Quindi ci serve una *FUNZIONE DELLA SOLA DIMENSIONE* :

$c(size(I))$

Come definire formalmente la dimensione size(I) ?

Come definire la funzione complessità c(size(I)) ?

Definizione di *size*(*I*)

- DIMENSIONE *size*(*I*) = *Numero di celle necessarie a rappresentare i dati di ingresso (istanza I).*

- *size*(*I*) è una *funzione f*(*x*,*y*,..) dei parametri dell'istanza I

ESEMPIO:

- PROBLEMA DI PROGRAMMAZIONE LINEARE:
  $min\ c^T x$: $\{x: Ax=b,\ x \geq 0\}$

Parametri di una generica istanza I :

- n: numero variabili
- m: numero vincoli
- $A = [a_{hk}]$: matrice $(m \times n)$
- $b = [b_h]$: vettore con m componenti
- $c = [c_h]$: vettore con n componenti

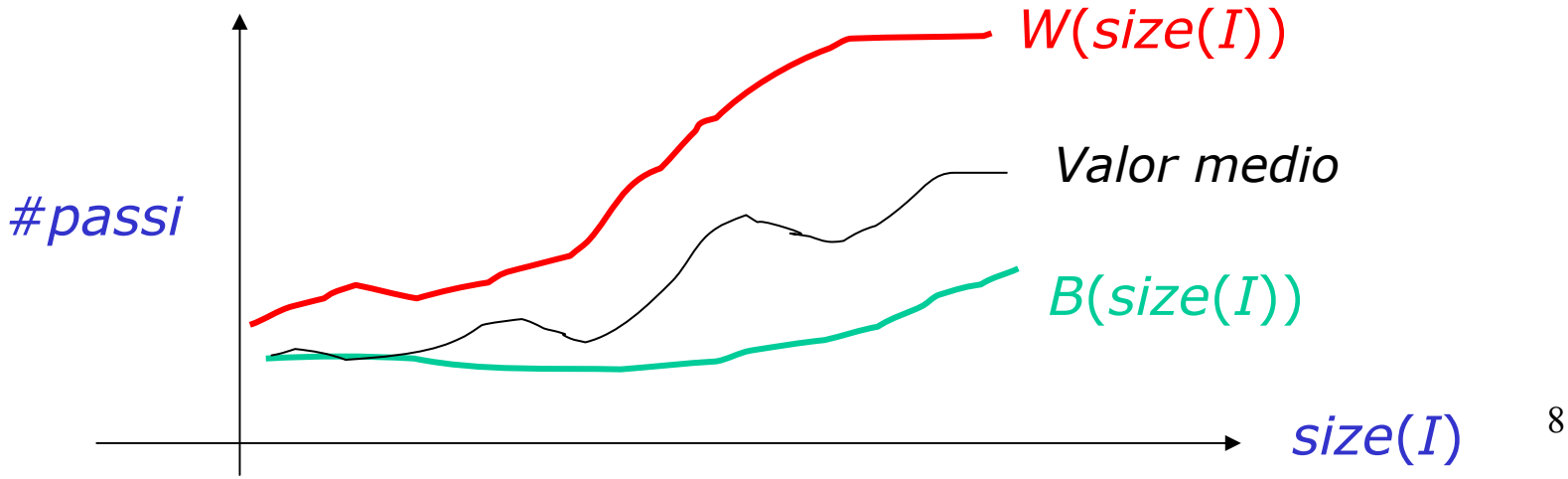
$size(I) = mn+n+m$

Definizione di *c(size(I))*

- Per definire *c(size(I))* *potremmo considerare il valor medio del tempo di esecuzione su tutte le istanze di dimensione size(I). Ma quale probabilità hanno le diverse istanze di presentarsi? Difficile determinarla (anche se non impossibile)!*
- *Allora considereremo i casi estremi:*

- COMPLESSITA' NEL *CASO PEGGIORE* *W(size(I))*
  - Funzione di *size(I)*
  - *Numero di passi necessari ad un algoritmo per determinare la soluzione della più difficile istanza di dimensione size(I)*

- COMPLESSITA' NEL *CASO MIGLIORE* *B(size(I))*
  - Funzione di *size(I)*
  - *Numero di passi necessari ad un algoritmo per determinare la soluzione della più facile istanza di dimensione size(I)*



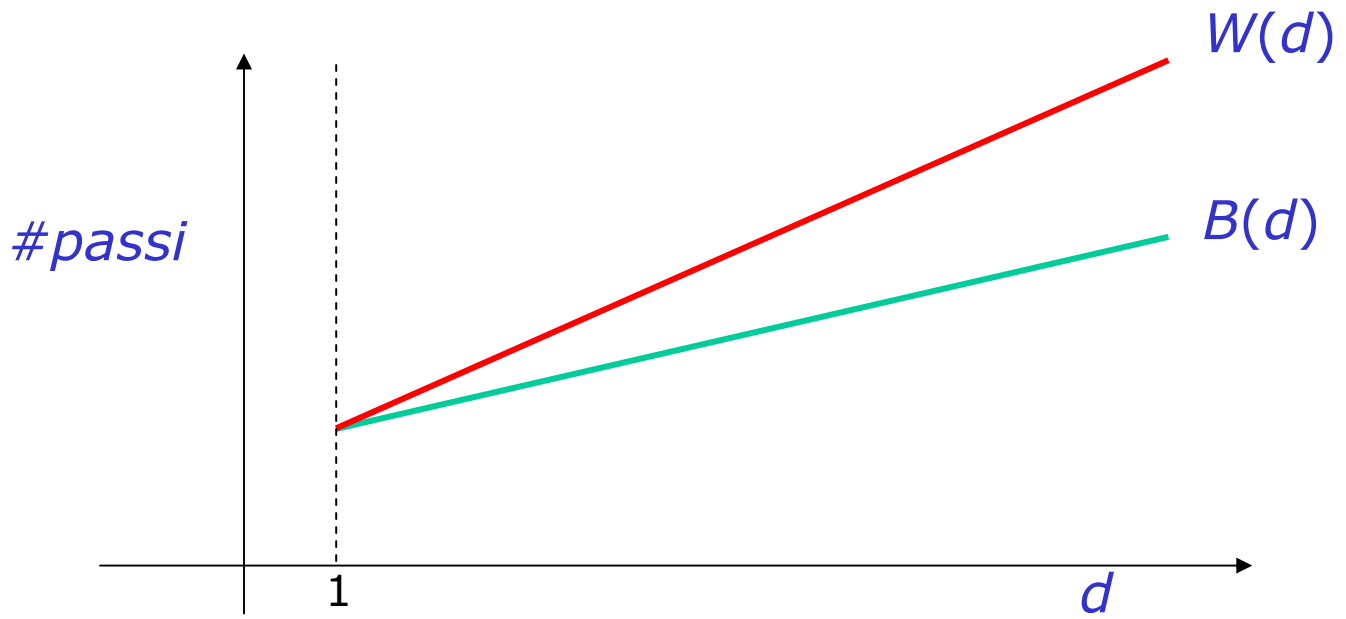
Caso Peggiore e Caso Migliore

- **PROBLEMA**: CALCOLO DELL'ELEMENTO MASSIMO IN UN VETTORE CON $d=size(I)$ ELEMENTI

NUMERO TOTALE PASSI T :
$2+4(d-1) \leq T \leq 2+6(d-1)$

- COMPLESSITA' NEL *CASO PEGGIORE* $W(d)=2+6(d-1)$

- COMPLESSITA' NEL *CASO MIGLIORE* $B(d)=2+4(d-1)$



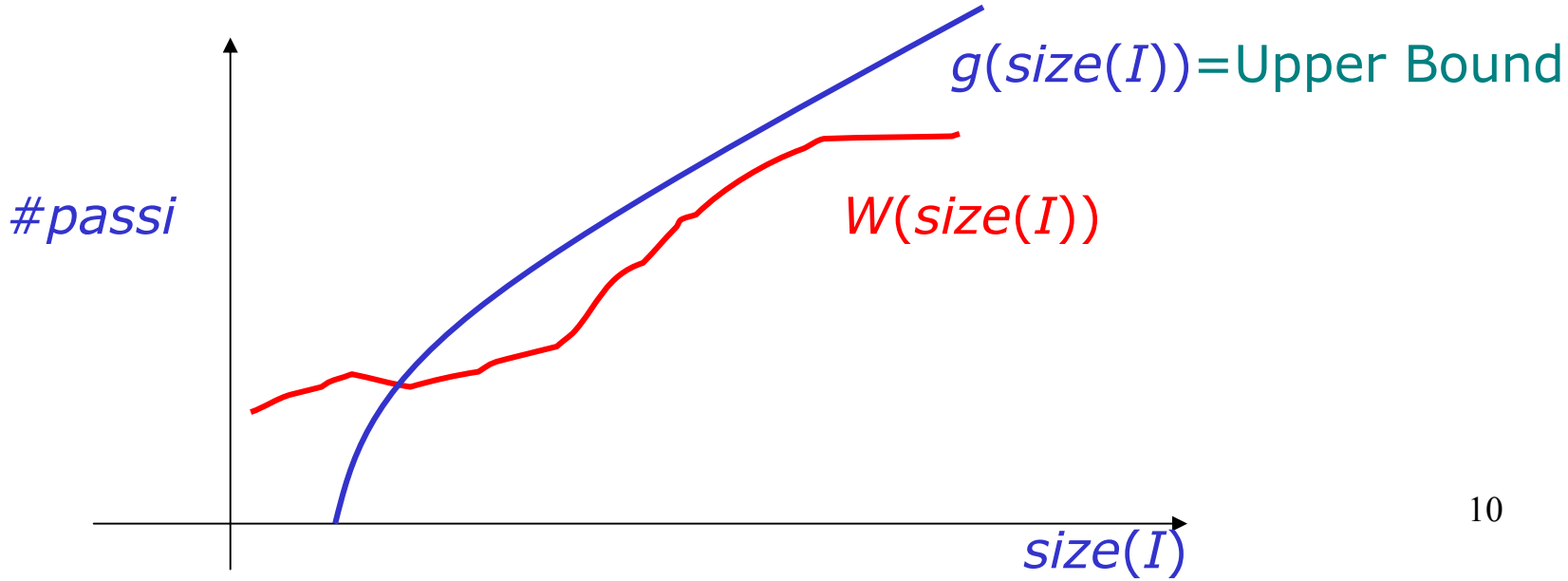
Scelta del Caso Peggiore

- *La complessità nel **CASO PEGGIORE** è la più utilizzata*

✓ *Il caso migliore si ottiene per istanze poco significative*

✓ *Il caso peggiore si ottiene per istanze "patologiche"*
ma comunque è una misura prudenziale della complessità

✓ *Conviene inoltre utilizzare una approssimazione superiore (Upper Bound)* facilmente calcolabile *g(size(I))* *della vera funzione W(size(I))*



Upper Bound: Funzione $\mathcal{O}(\ )$ "ordine di"

$f(x)$ è $\mathcal{O}(g(x)) \Leftrightarrow \exists\ x_0$: per $x \geq x_0$ $\quad f(x) \leq c_1 * g(x)$

Cioè $f(x)$ è ordine di $g(x)$ se, per valori di x abbastanza grandi, il valore di $f(x)$ è definitivamente inferiore a quello di $c_1 * g(x)$

$f(x) = x^2 + 100*x + 10000 \rightarrow \mathcal{O}(x^2)$

costante c_1

NOTA: se $x \leq 3 \rightarrow f(x) > x^2$ ma... se $x \geq 1000 \rightarrow f(x) < 2x^2$

Regola pratica per calcolare $g(x)$ dato $f(x)$:
- elimina coefficienti e costanti
- conserva l'addendo di $f(x)$ che "cresce" più rapidamente

$f(x) = 20*x^2 + log(x) + 10000 \rightarrow \mathcal{O}(x^2)$

$f(x) = 2^x + 20*x^2 + log(x) + 10000 \rightarrow \mathcal{O}(2^x)$

Complessità Polinomiale

- Quindi un algoritmo ha Complessità nel Caso Peggiore $\mathcal{O}(g(size(I)))$ se e solo se $g(size(I))$ *è Upper Bound di* $W(size(I))$

ESEMPIO:

- PROBLEMA: CALCOLO DELL' ELEMENTO MASSIMO IN UN VETTORE CON $d=size(I)$ ELEMENTI

- COMPLESSITA' NEL *CASO PEGGIORE* $W(d)=2+6(d-1)$ cioè $\mathcal{O}(d)$

- Un Algoritmo ha *Complessità Polinomiale* sse la sua Complessità nel Caso Peggiore è $\mathcal{O}(size(I)^k)$ con *k costante* (= è un polinomio)

ESEMPIO: PROGRAMMAZIONE LINEARE in questo caso $size(I)= mn+n+m$

Se esiste un algoritmo ***A*** *che risolve la PL in:*
$W(size(I))=m^4 + n^4 + 4mn + m^3$ *passi* (un polinomio di $size(I)$)

$W(size(I)) \leq c_1(mn+ n+ m)^4$

E quindi $\mathcal{O}(size(I)^4)$

⇒ ***A*** *polinomiale*

Complessità ed Efficienza

- Il valore k in $\mathcal{O}(size(I)^k)$ *è una misura dell'efficienza: un algoritmo* $\mathcal{O}(size(I)^2)$ *è migliore di un algoritmo* $\mathcal{O}(size(I)^3)$

- Naturalmente è possibile avere molti possibili andamenti per la complessità (costante, lineare, polinomiale, esponenziale, iperesponenziale, etc.)
- Un Algoritmo ha *Complessità Esponenziale se la complessità nel caso peggiore è limitata da un esponenziale, ma non da un polinomio, cioè cresce più velocemente di* $size(I)^k$ *per ogni possibile* k
- *Da notare che un esponenziale, da un certo punto in poi, cresce più velocemente di un qualsiasi polinomio*
- *E a noi interessa soprattutto quello che succede da un certo punto in poi, cioè quando volgiamo risolvere istanze grandi*

Confronto tra Vari Andamenti

log(n)	n	n^2	n^5	2^n
1	2	4	32	4
2	4	16	1024	16
3	8	64	32768	256
4	16	256	1048576	65536
5	32	1024	33554432	4,29E+09
6	64	4096	1,07E+09	1,84E+19
7	128	16384	3,44E+10	3,4E+38
8	256	65536	1,1E+12	1,16E+77
9	512	262144	3,52E+13	1,3E+154
10	1024	1048576	1,13E+15	#NUM!

$size(I)=n$

- Ci interessa soprattutto la demarcazione tra **polinomiale** e **esponenziale**, perché (Edmonds, 1965): un algoritmo polinomiale è *EFFICIENTE ("GOOD")*
- Uno esponenziale (o più) invece non è efficiente: le istanze oltre una certa $size(I)$ non le risolveremo praticamente mai, anche se i computer diventano progressivamente più veloci (abbiamo visto l'esempio molto inefficiente dell'enumerazione completa)

Modello di Calcolo Più Sofisticato

- Per vedere altre questioni usiamo un altro modello di calcolo: **Macchina di Turing**

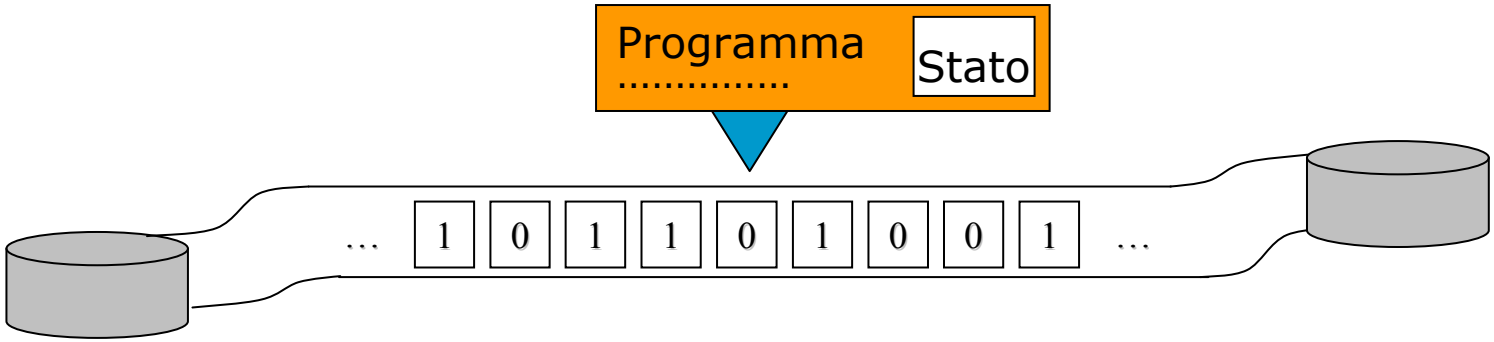
Una Macchina di Turing è un meccanismo calcolatore astratto formato da:

- **Nastro** (infinito, ogni cella contiene un simbolo)
- **Testina** (legge o scrive il nastro e si sposta rispetto a lui)
- **Programma** (per ogni stato e simbolo in input, una regola che dice cosa fare)
- **Stato** (appartiene ad un insieme di possibili stati)

La macchina ha uno stato iniziale, e dato un input, effettua una serie di operazioni producendo l'output (che potrebbe richiedere tempo infinito)

La tesi di Church postula che tutte le funzioni effettivamente calcolabili siano esprimibili con una Macchina di Turing

- MdT **deterministica**: dato uno stato e un input, esegue una determinata operazione
- MdT **non deterministica**: dato uno stato e un input, può eseguire una tra varie determinate operazioni



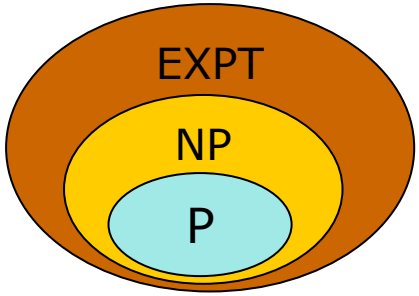
Classi di Complessità

A seconda dell'andamento della complessità usando il modello Macchina di Turing si definiscono delle classi di complessità (sempre nel caso peggiore)

Ad esempio:

- **Classe P** (problemi risolubili in tempo **polinomiale** su MdT **deterministica**)
- **Classe NP** (problemi risolubili in tempo **polinomiale** su MdT **non deterministica**)
- **Classe ExpT** (problemi risolubili in tempo **esponenziale** su MdT **deterministica**)
- **…**

- Le relazioni tra queste classi sono di inclusione, ma esistono ancora incertezze
- In particolare, sappiamo che $P \subseteq NP$ (un problema polinomiale per MdT D lo è anche per MdT ND)
- Ma **non sappiamo** se $P \subset NP$ o $P = NP$

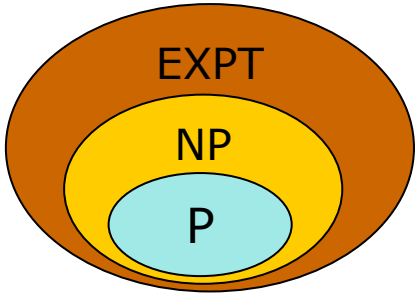


- Questo problema è aperto da 30 anni ed è uno dei **millenium problems**

La classe NP

- Quindi **non sappiamo** se P $\subset$ NP o P = NP

Questa questione è molto importante e molto studiata, perché:



- La parte NP-P (in giallo nella figura) contiene molti problemi **estremamente importanti** dal punto di vista pratico (ad es. molti problemi che vedremo, OC, PL01)
- Sono problemi che attualmente sappiamo risolvere solo in tempo **esponenziale** (quindi ci risultano difficili)
- Ma, lavorando come una **macchina non deterministica** (che nella realtà però non esiste!), potrebbero essere risolti in tempo **polinomiale** (quindi ci risulterebbero facili!)
- La PL invece appartiene alla classe P
- In altre parole, risolvere un problema di PL è facile, risolverne uno di PL01 o OC è in generale difficile
- Ma ci sono molte **sottoclassi** di PL01 e di OC che sono facili!

Complessità di un Problema

- Dato un problema, posso progettare **diversi** algoritmi per risolverlo (soprattutto per problemi non banali)
- Ma non è detto che **tutti abbiano la stessa complessità**: alcuni potrebbero anche essere corretti ma poco efficienti
- Allora la complessità di un problema si definisce pari a a quella dell'algoritmo **più semplice** per risolverlo
- I problemi per cui la complessità è >= di quella di **qualsiasi altro** problema in NP si chiamano **NP-difficili** (**NP-hard**): vuol dire che sono difficili almeno quanto NP
- Un problema si dice **NP-completo** quando è in NP ed è NP-hard
- Molti problemi di **OC** sono NP-completi

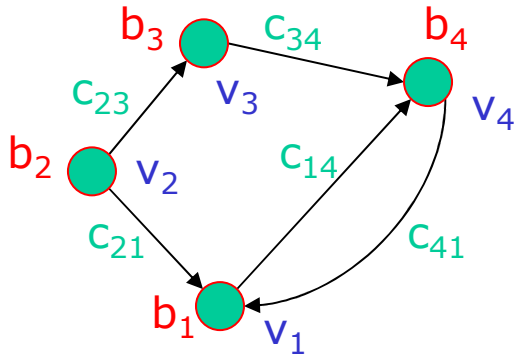
Esempio: Rappresentazione di Grafi

ISTANZA

$G(N,A)$ *Grafo orientato*

$[b_h]_{h \in N}$ *parametri di nodo*

$[c_{hk}]_{hk \in A}$ *parametri di arco*



PROBLEMA DELLA RAPPRESENTAZIONE di un Grafo Orientato $G(N,A)$ e dei parametri (lunghezze,capacità,costi...) associati a nodi e archi

IPOTESI: Ogni componente di $[c_{hk}]$ e $[b_h]$ richiede 1 cella

Rappresentazione della Struttura delle adiacenze di $G(N,A)$
$\rightarrow$ Rappresentazione delle Stelle Uscenti di tutti i nodi di N

Due modi possibili:

MATRICE DI ADIACENZA

LISTE DI ADIACENZA

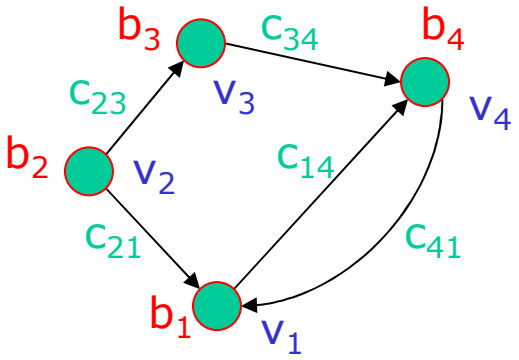
Matrice di Adiacenza

ISTANZA

$G(N,A)$ *Grafo orientato*

$[b_h]_{h \in N}$ *parametri di nodo*

$[c_{hk}]_{hk \in A}$ *parametri di arco*



Rappresentazione del Grafo:
MATRICE DI ADIACENZA:

$\mathcal{A} = [a_{hk}]$ $(N \times N)$

La componente a_{hk} è associata alla coppia ordinata $\{v_h, v_k\}$ di nodi:

$$\begin{cases} a_{hk}=1 & (v_h,v_k) \in A \\ a_{hk}=0 & (v_h,v_k) \notin A \end{cases}$$

	v_1	v_2	v_3	v_4
v_1	-	**0**	**0**	**1**
v_2	**1**	-	**1**	**0**
v_3	**0**	**0**	-	**1**
v_4	**1**	**0**	**0**	-

$\mathcal{A} = [a_{hk}]$

	v_1	v_2	v_3	v_4
v_1	-	-	-	c_{14}
v_2	c_{21}	-	c_{23}	-
v_3	-	-	-	c_{34}
v_4	c_{41}	-	-	-

$[c_{hk}]$

v_1	v_2	v_3	v_4
b_1	b_2	b_3	b_4

$[b_h]$

OCCUPAZIONE DI MEMORIA (celle):

$$\left.\begin{array}{rcl} \mathcal{A} & \rightarrow & |N|^2 \\ [c_{hk}] & \rightarrow & |N|^2 \\ [b_h] & \rightarrow & |N| \end{array}\right\} \Rightarrow |N|^2 + |N|^2 + |N|$$

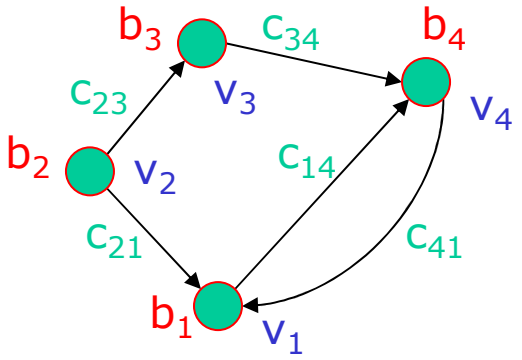
Liste di Adiacenza

ISTANZA

$G(N,A)$ *Grafo orientato*

$[b_h]_{h \in N}$ *parametri di nodo*

$[c_{hk}]_{hk \in A}$ *parametri di arco*



Tutte le informazioni relative agli archi (i parametri c_{hk} e le adiacenze) sono rappresentate da $|N|$ liste (una per nodo)

Lista $\mathcal{L}(u)$: $u \in N$ rappresenta gli archi della stella uscente di u

OCCUPAZIONE TOTALE DI MEMORIA:

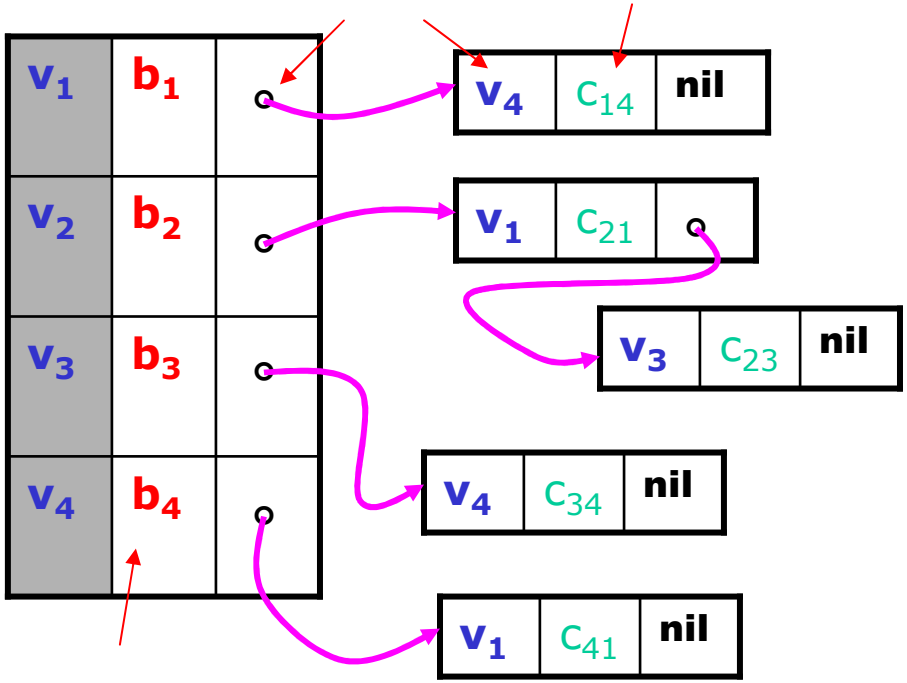
Indici di nodo $\rightarrow |N|$

$[b_h] \rightarrow |N|$

Puntatori arco $\rightarrow |A|$

$[c_{hk}] \rightarrow |A|$

$\Rightarrow 2(|A| + |N|)$



Occupazione di Memoria dell'Istanza Grafo

ISTANZA

$G(N,A)$ *Grafo orientato*

$[b_h]_{h \in N}$ *parametri di nodo*

$[c_{hk}]_{hk \in A}$ *parametri di arco*

PONIAMO: $|N|=n$ *e* $|A|=m$

Ipotesi: $n \leq m$

Allora vediamo la dimensione dell'istanza: $size(G(N,A)) = f(n, m)$

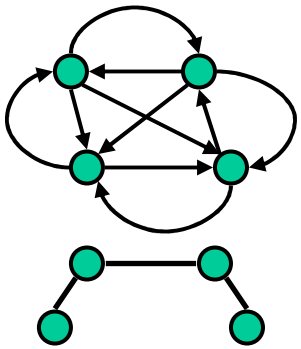
MATRICE DI ADIACENZA: $size_A = 2n^2 + n$

LISTE DI ADIACENZA: $size_L = 2m + 2n$

Che rapporto hanno tra loro *n* e *m* ?

$m \leq n^2$; $m \approx n^2$ *grafo denso*

$m \approx n$ *grafo sparso*



Quale rappresentazione conviene ?

DIPENDE DAL PUNTO DI VISTA e DALLE CARATTERISTICHE DEL GRAFO..

Complessità Polinomiale in Funzione di *size(G)*

MATRICE DI ADIACENZA: $size_A = 2n^2 + n$

LISTE DI ADIACENZA: $size_L = 2m + 2n$

1. Se vogliamo valutare la COMPLESSITA` (nel caso peggiore)

$size_A = 2n^2 + n \rightarrow \mathcal{O}(n^2)$

$size_L = 2m + 2n \rightarrow \mathcal{O}(m)$

Nel caso peggiore il grafo sarà denso quindi $m \approx n^2$ *e allora* *size(G)* sarà comunque *dell'ordine di* n^2

QUINDI: $size(G)=n^2$

DI CONSEGUENZA:

Un algoritmo su grafi ha Complessità Polinomiale se e solo se:
$W(size(G))=W(n^2) \leq c_1(n^2)^k$ *con k costante*

OVVERO:

Un algoritmo su grafi ha Complessità Polinomiale se e solo se la sua Complessità nel Caso Peggiore è $\mathcal{O}(n^k)$ *con k costante*

Dimensione di un'Istanza Grafo (in pratica)

MATRICE DI ADIACENZA: $size_A = 2n^2 + n$

LISTE DI ADIACENZA: $size_L = 2m + 2n$

2. Se vogliamo consumare poco SPAZIO DI MEMORIA

Se $m \approx n^2$ *(grafo denso)* $\rightarrow$ *MATRICE DI ADIACENZA*

$size_A = \mathcal{O}(n^2)$; $size_L = \mathcal{O}(m) \approx \mathcal{O}(n^2)$

Se $m \approx n$ *(grafo sparso)* $\rightarrow$ *LISTE DI ADIACENZA*

$size_A = \mathcal{O}(n^2)$; $size_L = \mathcal{O}(m) \approx \mathcal{O}(n)$

Per esempio, se il grafo ha $n = 10^4$ *e* $m = 10^5$ *(è abbastanza sparso)*

$size_A \approx 10^8 \approx$ *100 milioni di celle*

$size_L \approx 10^5 \approx$ *100.000 celle*