Lezione12: classi di complessità

Lezione del 16/04/2024

Alla ricerca della macchina più veloce

- Ci siamo lasciati con la storia della correlazione polinomiale:

Tutti i modelli (deterministici) sono correlati polinomialmente

- E va bene. Tuttavia,
  - se ho una macchina di Turing T che decide linguaggio $L \subseteq \Sigma^*$ tale che, per ogni $x \in \Sigma^*$, $dtime(T, x) \leq |x|^3$
  - e un'altra macchina T_4 che decide lo stesso linguaggio L e tale che , per ogni $x \in \Sigma^*$, $dtime(T_4, x) \leq \frac{x^3}{4} \ldots$
- Beh, mi sa tanto che mi conviene scegliere T_4, per decidere L!
- Ma se poi progetto ancora un'altra macchina T_8 che decide lo stesso linguaggio L e tale che $dtime(T_8, x) \leq \frac{x^3}{8} \ldots$ Allora, cavolo, sceglierò quest'ultima!
- Fino a quando riuscirò a progettare la macchina che impiega meno tempo di tutte!
- Ma nella Teoria della Complessità Computazionale le cose non sono proprio così…

Alla ricerca della macchina più veloce

- **Teorema 6.7 [Accelerazione lineare].**
  Sia $L \subseteq \Sigma^*$ *un linguaggio deciso da una macchina di Turing deterministica ad un nastro* T *tale che, per ogni* $x \in \Sigma^*$, $dtime(T,x) = t(|x|)$ *e sia* $k > 0$ *una costante. Allora:*
  - *esiste una macchina di Turing* *ad un nastro* T_1 *tale che* T_1 *decide* L *e, per ogni* $x \in \Sigma^*$, $dtime(T_1, x) \leq \frac{t(|x|)}{k} + O(|x|^2)$
  - *esiste una macchina di Turing* *a due nastri* T_2 *tale che* T_2 *decide* L *e, per ogni* $x \in \Sigma^*$, $dtime(T_2, x) \leq \frac{t(|x|)}{k} + O(|x|)$
- Questo teorema ci dice che, dato un qualunque algoritmo, esiste sempre un algoritmo più veloce del primo di un fattore costante!
- Resta da capire: perché i due addendi $O(|x|^2)$ e $O(|x|)$?
  - essi derivano dal fatto che, per poter essere più veloci, le macchine T_1 e T_2 devono innanzi tutto codificare in forma compressa il proprio input (vedi prossimo teorema): se la codifica compressa viene scritta su un nastro apposito (come fa T_2 sul suo secondo nastro) sono sufficienti $O(|x|)$ passi, se si dispone di un solo nastro (il caso di T_1) occorrono $O(|x|^2)$ passi
- Non dovete studiare la dimostrazione del Teorema 6.7

Risparmiare memoria

- Si può dimostrare qualcosa di analogo nel caso della funzione dspace
- **Teorema 6.6 [Compressione lineare].**
  Sia $L \subseteq \Sigma^*$ *un linguaggio deciso da una macchina di Turing deterministica ad un nastro* T *tale che, per ogni* $x \in \Sigma^*$, $\mathrm{dspace}(T, x) = s(|x|)$ *e sia* $k > 0$ *una costante. Allora:*
  - *esiste una macchina di Turing ad un nastro* T_1 *tale che* T_1 *decide* L *e, per ogni* $x \in \Sigma^*$, $\mathrm{dspace}(T_1, x) \leq \frac{s(|x|)}{k} + O(|x|)$
- Questo teorema ci dice che, dato un qualunque algoritmo, esiste sempre un algoritmo che una una frazione costante della memoria del primo!
- Resta da capire: perché l'addendo $O(|x|)$?
  - deriva dal fatto che l'input di T_1 è lo stesso di T. Pertanto T_1 deve innanzi tutto codificare in forma compressa il proprio input e poi lavorare sull'alfabeto compresso: osservate che l'alfabeto compresso è Σ^k (ossia, un carattere dell'alfabeto compresso è una parola di k caratteri di Σ) e che l'alfabeto di T_1 è $\Sigma^k \cup \Sigma$
- Non dovete studiare neanche la dimostrazione del Teorema 6.6

Classi di complessità (deterministiche)

- Siamo pronti a raggruppare i linguaggi in base all'efficienza delle macchine che li decidono – e siamo a pag. 9 della dispensa 6
  - per esempio, potremmo considerare l'insieme dei linguaggi tali che la ***la migliore macchina che li decide*** ha una certa efficienza
- E che vuol dire?
  - un linguaggio L è un insieme di parole – contiene, tipicamente, infinite parole
  - e una macchina che decide L, tipicamente, esegue un numero diverso di operazioni quando opera su input diversi – *anche su input diversi che hanno la stessa lunghezza*
  - ve la ricordate, ad esempio, la cara, vecchia, T_{PPAL}, che accettava parole palindrome di lunghezza pari sull'alfabeto {a,b}? Ebbene: T_{PPAL}(abababab) rigetta dopo aver eseguito 10 quintuple, T_{PPAL}(abbbbbba) accetta dopo aver eseguito all'incirca 45 quintuple (deve fare avanti e indietro un sacco di volte!)
  - e considerazioni analoghe possono essere fatte per la misura dspace)
- Cosa significa dire che una macchina che decide un linguaggio ha una certa efficienza?
  - che si comporta "bene" (con quella efficienza) *almeno su qualche input*?
  - o che si comporta "bene" *su ogni input*?

Classi di complessità (deterministiche)

- Siamo pronti a raggruppare i linguaggi in base all'efficienza delle macchine che li decidono
  - per esempio, potremmo considerare l'insieme dei linguaggi tali che la ***la migliore macchina che li decide*** ha una certa efficienza
- La risposta corretta è la seconda: vogliamo che la macchina che decide un linguaggio $L \subseteq \Sigma^*$ si comporti "bene" su ***ogni*** parola $x \in \Sigma^*$
- Poi, non possiamo scegliere la "***migliore***" macchina che decide un linguaggio
  - perché se un linguaggio è deciso da una macchina che ha una certa efficienza, quel linguaggio è deciso anche da una macchina che è efficiente il doppio. O il triplo. O il quadruplo…
- E per risolvere questa questione ricorriamo alla notazione O: diciamo che *un linguaggio L appartiene all'insieme caratterizzato dalla "efficienza temporale" individuata dalla funzione totale e calcolabile f se* ***esiste*** *una macchina T che decide (o accetta) L e che, per ogni parola x sull'alfabeto di L, termina in* **O(f(|x|))** *istruzioni*
- E analogamente a proposito di "efficienza spaziale"
- OSSERVATE BENE: **è sparita la richiesta di "*migliore*" macchina che decide L…**

Classi di complessità deterministiche

- Le classi che misurano "efficienza temporale" nel caso deterministico si chiamano **DTIME**: data una **funzione totale e calcolabile f**,

  DTIME[f(n)] = { L ⊆{0,1}* tali che <u>esiste</u> una macchina deterministica T che decide L e, per ogni x ∈ {0,1}*, dtime(T,x) ∈ O(f(|x|)) }

  - in Teoria della Complessità Computazionale si parla di classi invece che di insiemi
  - ATTENZIONE: **dtime** (minuscolo) è la misura di complessità, ossia, una funzione; **DTIME** (maiuscolo) è una classe di complessità, ossia, un insieme!
  - Vi rendete conto, spero, che DTIME[f(n)] = DTIME[f(n)/2] = DTIME[2 f(n)+58] = …
  - come è giusto che sia a seguito del **Teorema di accelerazione lineare**.

- Le classi che misurano "efficienza spaziale" nel caso deterministico si chiamano **DSPACE**: data una **funzione totale e calcolabile f**,

  DSPACE[f(n)] = { L ⊆{0,1}* tali che <u>esiste</u> una macchina deterministica T che decide L e, per ogni x ∈ {0,1}*, dspace(T,x) ∈ O(f(|x|)) }

Classi di complessità non deterministiche

- Le stesse considerazioni che ci hanno condotto a definire le classi di complessità deterministiche, possono essere ripetute anche nel caso non deterministico
- Le classi che misurano "efficienza temporale" nel caso non deterministico si chiamano **NTIME**: data una **funzione totale e calcolabile f**,

  NTIME[f(n)] = { L ⊆{0,1}* tali che esiste una macchina non deterministica NT che ACCETTA L e, per ogni x ∈ L, ntime(NT,x) ∈ O(f(|x|)) }

  - Ma perché una classe non deterministica è definita in base al tempo di accettazione, invece che del tempo di decisione? Ricordate quello che abbiamo detto la scorsa lezione: se sappiamo che un linguaggio è accettato entro un certo numero di istruzioni, sappiamo che quel linguaggio è decidibile, *ma non sappiamo quanto tempo occorre a rigettare le parole del suo complemento*!
  - E a noi interessa accettare le parole del linguaggio – non di rifiutare quelle del complemento!
- Le classi che misurano "efficienza spaziale" nel caso non deterministico si chiamano **NSPACE**: data una **funzione totale e calcolabile f**,

  NSPACE[f(n)] = { L ⊆{0,1}* tali che esiste una macchina non deterministica NT che ACCETTA L e, per ogni x ∈ L , nspace(NT,x) ∈ O(f(|x|)) }

Classi complemento

- Sia f una **funzione totale e calcolabile**
- La classe **coDTIME[f(n)]** contiene i linguaggi il cui complemento è contenuto in DTIME[(f(n)]:

  coDTIME[f(n)] = $\{L \subseteq \{0,1\}^*$ tali che $L^C \in$ DTIME[f(n)] $\}$

- La classe **coDSPACE[f(n)]** contiene i linguaggi il cui complemento è contenuto in DSPACE[(f(n)]:

  coDSPACE[f(n)] = $\{L \subseteq \{0,1\}^*$ tali che $L^C \in$ DSPACE [f(n)] $\}$

- La classe **coNTIME[f(n)]** contiene i linguaggi il cui complemento è contenuto in NTIME[(f(n)]:

  coNTIME[f(n)] = $\{L \subseteq \{0,1\}^*$ tali che $L^C \in$ NTIME[f(n)] $\}$

- La classe **coNSPACE[f(n)]** contiene i linguaggi il cui complemento è contenuto in NSPACE[(f(n)]:

  coNSPACE[f(n)] = $\{L \subseteq \{0,1\}^*$ tali che $L^C \in$ NSPACE [f(n)] $\}$

- Le definizioni formali sono a pag. 10 della dispensa 6

Un paio di questioni

- Innanzi tutto, perché ci limitiamo a considerare linguaggi definiti sull'alfabeto {0,1}?
  - In realtà, lo facciamo perché è più comodo
  - ma potremmo utilizzare un alfabeto qualsiasi (e, quando ci farà comodo, lo faremo)
  - tanto, sappiamo che se un linguaggio è deciso da una macchina definita su un alfabeto qualsiasi, allora esiste anche una macchina definita su {0,1} che lo decide (Lezione a distanza 2)
  - E le due macchine, sappiamo, sono pure polinomialmente correlate!
  - Sennò, che le abbiamo studiate a fare tutte queste belle cose?
- Poi, alla funzione f che che definisce una classe di complessità (ad esempio, DTIME[**f(n)**]) diamo il nome di ***funzione limite***
- Ma perché viene sempre richiesto che una funzione limite sia totale e calcolabile?
  - Perché che ce ne facciamo di sapere che una certa computazione T(x) esegue al più f(|x|) istruzioni se f(|x|) non sappiamo quant'è?

Relazioni fra classi di complessità

- Siamo al paragrafo 6.4
- **Teorema 6.8**: *Per ogni funzione totale calcolabile* $f : \mathbb{N} \rightarrow \mathbb{N}$,
  DTIME[f (n)] ⊆ NTIME[f (n)] e DSPACE[f (n)] ⊆ NSPACE[f (n)].
  - Facile: una macchina di Turing deterministica è una particolare macchina di Turing non deterministica avente grado di non determinismo pari ad 1 e, inoltre, ogni parola decisa in *un certo numero* di passi è anche accettata in quel *un certo numero* di passi, e una parola decisa utilizzando *un certo numero* di celle è anche accettata in quel *un certo numero* di celle
- **Teorema 6.9**: *Per ogni funzione totale calcolabile* $f : \mathbb{N} \rightarrow \mathbb{N}$,
  DTIME[f (n)] ⊆ DSPACE[f (n)] e NTIME[f (n)] ⊆ NSPACE[f (n)].
  - segue direttamente dal **Teorema 6.1**. Sia L ⊆ {0,1}* tale che L ∈ DTIME[f (n)]:
  - allora, esiste una macchina di Turing deterministica T che decide L e tale che, per ogni x ∈ {0,1}*, **dtime(T,x) ∈ O(f(|x|))**
  - poiché **dspace(T,x) ≤ dtime(T,x)**
  - allora, **dspace(T,x) ≤ dtime(T,x) ∈ O(f(|x|))**
  - questo implica che dspace(T,x) ∈ O(f(|x|)) e che, dunque, L ∈ DSPACE[f(n)].
  - Analogo il caso non deterministico

Relazioni fra classi di complessità

- **Teorema 6.10**: *Per ogni funzione totale calcolabile* $f : \mathbb{N} \rightarrow \mathbb{N}$,

  $DSPACE[\, f(n)] \subseteq DTIME[2^{O(1)\, f(n)}]$ e $NSPACE[\, f(n)] \subseteq NTIME[2^{O(1)\, f(n)}]$.

  - Anche in questo caso, la prova segue direttamente dal **Teorema 6.1**.
  - Sia $L \subseteq \{0,1\}^*$ tale che $L \in DSPACE[f(n)]$: allora, esiste una macchina di Turing deterministica T che decide L e tale che, per ogni $x \in \{0,1\}^*$, $dspace(T,x) \in O(f(|x|))$.
  - Poiché

$$\mathbf{dtime(T,x) \leq dspace(T,x)\,|Q|\,(|\Sigma|+1)^{dspace(T,x)}} = \mathbf{dspace(T,x)}\,|Q|\;3^{dspace(T,x)} =$$

$$= \mathbf{2^{\log dspace(T,x)}}\,|Q|\;[2^{\log 3}]^{dspace(T,x)}$$

$$= |Q|\;2^{\log dspace(T,x)\; +\, dspace(T,x)\,\log 3} \leq |Q|\;2^{[1+\log 3]\, dspace(T,x)}$$

  - allora $dtime(T,x) \in O(2^{O(1) f(|x|)})$
  - e, dunque, $L \in DTIME[\, 2^{O(1)\, f(n)}]$.

  - La dimostrazione per il caso non deterministico è analoga.

Relazioni fra classi di complessità

- **Teorema 6.11**: *Per ogni funzione totale calcolabile* $f : \mathbb{N} \rightarrow \mathbb{N}$,
  DTIME[f (n)] = coDTIME[f (n)] e DSPACE[f (n)] = coDSPACE[f (n)].
  - Sia $L \subseteq \{0,1\}^*$ tale che $L \in$ DTIME[$f(n)$]: allora, esiste una macchina di Turing deterministica T che decide L e tale che, per ogni $x \in \{0,1\}^*$, dtime(T,x) $\in O(f(|x|))$.
  - Poiché T decide L, allora $T(x)=q_A$ se $x \in L$, e $T(x)=q_R$ se $x \in \{0,1\}^*-L = L^C$
  - Costruiamo una macchina T' identica a T tranne per il fatto che, rispetto a T, gli stati di accettazione e di rigetto di T' sono invertiti,
  - allora, per ogni $x \in \{0,1\}^*$, dtime(T',x) $\in O(f(|x|))$,
  - e, inoltre, $T'(x)=q_R$ se $x \in L$, e $T'(x)=q_A$ se $x \in \{0,1\}^*-L = L^C$.
  - Dunque, T' decide L^C e, per ogni $x \in \{0,1\}^*$, dtime(T',x) $\in O(f(|x|))$.
  - Quindi, $L^C \in$ DTIME[$f(n)$].
  - Poiché L è un qualunque linguaggio in DTIME[$f(n)$] e, quindi, L^C è un qualunque linguaggio in coDTIME[$f(n)$], questo significa che:
    - per ogni linguaggio $L^C \in$ coDTIME[$f(n)$], $L^C \in$ DTIME[$f(n)$] – ossia, coDTIME[f (n)] $\subseteq$ DTIME[f (n)]
    - per ogni linguaggio $L \in$ DTIME[$f(n)$], poiché $L^C \in$ DTIME[$f(n)$], allora $L \in$ coDTIME[$f(n)$], ossia
      DTIME[f (n)] $\subseteq$ coDTIME[f (n)]
  - La dimostrazione per DSPACE e coDSPACE è analoga.

Classi… "poco precise"

- Attenzione: l'utilizzo di O nella definizione delle classi di complessità ha come conseguenza che esse non caratterizzino con precisione i linguaggi
  - nel senso che, se dimostriamo che un certo linguaggio L è contenuto, ad esempio, in DTIME[f(n)] (per qualche funzione totale e calcolabile f), allora… esiste una serie *infinita* di classi DTIME nelle quali L è contenuto!
  - andiamo a chiarire
- Ricordiamo che, date $f : \mathbb{N} \rightarrow \mathbb{N}$ e $g : \mathbb{N} \rightarrow \mathbb{N}$ due funzioni, **$f(n) \in O(g(n))$** se
  - esistono $n_0 \in \mathbb{N}$ e $c \in \mathbb{N}$ tali che, per ogni $n \geq n_0$, $f(n) \leq c\, g(n)$
  - $\exists\, n_0 \in \mathbb{N}\ \exists\, c \in \mathbb{N} : \forall n \geq n_0\ [\, f(n) \leq c\, g(n)\,]$
- Ossia, **$O(f(n)) \subseteq O(g(n))$** e da questo segue il seguente teorema
- **Teorema 6.12:** ***Per ogni coppia di funzioni totali calcolabili $f : \mathbb{N} \rightarrow \mathbb{N}$ e $g : \mathbb{N} \rightarrow \mathbb{N}$ tali che***

  $\exists\, n_0 \in \mathbb{N} : \forall n \geq n_0\ [\, f(n) \leq g(n)\,]$ – ossia $f(n) \leq g(n)$ *definitivamente*

  DTIME[f (n)] ⊆ DTIME[g(n)] **NTIME[f (n)] ⊆ NTIME[g(n)],**
  DSPACE[f (n)] ⊆ DSPACE[g(n)] **NSPACE[f (n)] ⊆ NSPACE[g(n)].**

  - infatti, $O(f(n)) \subseteq O(g(n))$

Classi… “poco precise”

- Ok, allora il Teorema 6.12 ci dice che, se collochiamo un linguaggio L, ad esempio, in DTIME[f(n)], allora L appartiene anche a tutte le classi DTIME[g(n)] tali che, definitivamente, $f(n) \leq g(n)$

- E questo, fate attenzione, significa che: **se collochiamo un linguaggio L, ad esempio, in DTIME[f(n)], questo non implica che L non possa appartenere anche a qualche classe DTIME[r(n)] tali che, definitivamente, $r(n) \leq f(n)$!**

- Che, detto altrimenti, significa che qualcuno potrebbe progettare per decidere L un algoritmo più efficiente del nostro!
- Perciò, aver collocato un linguaggio L, ad esempio, in DTIME[f(n)], è aver fatto solo metà del lavoro
  - l’altra metà sarebbe dimostrare che L non appartiene a DTIME[r(n)] per alcuna funzione r(n) tale che, definitivamente, $r(n) \leq f(n)$!
  - e questo è un compito parecchio (assai) più complesso

Qualcosa di strano…

- Ok, allora il Teorema 6.12 ci dice che, se collochiamo un linguaggio L, ad esempio, in DTIME[f(n)], allora L appartiene anche a tutte le classi DTIME[g(n)] tali che, definitivamente, $f(n) \leq g(n)$
- Di contro, nella definizione di una teoria della complessità in grado di classificare significativamente i linguaggi in classi di complessità crescente,
  - perché, in definitiva, noi vorremmo poter dire; "questo problema è **più difficile** di quest'altro"
- sarebbe auspicabile che **DTIME[f(n)] <u>non</u> fosse contenuto in DTIME[g(n)] quando f(n) è *molto* più grande di g(n)** – ad esempio, quando $f(n) = 2^{g(n)}$!
- Ma, invece:
- **Teorema 6.13 (Gap Theorem):** *<u>Esiste</u> una funzione totale calcolabile* $f : \mathbb{N} \rightarrow \mathbb{N}$ *tale che*

$$\mathrm{DTIME}[\, 2^{f(n)} \,] \subseteq \mathrm{DTIME}[f(n)].$$

  - <u>(non dovete studiare la dimostrazione! NO!)</u>
- Ops!
- E allora?!
- Il seguito alla prossima lezione…

Ricapitoliamo

- Abbiamo già visto che, se collochiamo un linguaggio L, ad esempio, in DTIME[f(n)], allora L appartiene anche a <u>tutte</u> le classi DTIME[$f(n)^k$] per ogni $k \in \mathbb{N}$
  - perché, definitivamente, $f(n) \leq f(n)^k$

- ossia, abbiamo una gerarchia di infinite classi di complessità

  DTIME[f(n) $\subseteq$ DTIME[$f(n)^2$] $\subseteq$ DTIME[$f(n)^3$] $\subseteq$ ... $\subseteq$ DTIME[$f(n)^k$] $\subseteq$ D
- e, in generale, data una funzione totale e calcolabile f, è vero che

  DTIME[f(n) $\subseteq$ DTIME[g(n)]

  per qualunque altra funzione totale e calcolabile g tale che $g(n) \geq f(n$ definitivamente DTIME[f(n) $\subseteq$ DTIME[g(n)]

- D'altra parte, nella definizione di una teoria della complessità in grado di classificare significativamente i linguaggi in classi di complessità crescente,
- sarebbe auspicabile che **DTIME[f(n)] non fosse contenuto in DTIME[g(n)] quando f(n) è *molto* più grande di g(n)** – ad esempio, quando **$f(n) = 2^{g(n)}$** !
  - perché, in definitiva, quel che ci interessa è poter dire; "questo problema è più difficile di quest'altro"

Ricapitoliamo

- Nella definizione di una teoria della complessità in grado di classificare significativamente i linguaggi in classi di complessità crescente,
- sarebbe auspicabile che **DTIME[f(n)] non fosse contenuto in DTIME[g(n)] quando f(n) è *molto* più grande di g(n)** – ad esempio, quando **$f(n) = 2^{g(n)}$** !
  - perché, in definitiva, quel che ci interessa è poter dire; "questo problema è più difficile di quest'altro"
- Ma, invece:
- **Teorema 6.13 (Gap Theorem):** *Esiste una funzione totale calcolabile* $f : \mathbb{N} \rightarrow \mathbb{N}$ *tale che*

$$\mathrm{DTIME}[\, 2^{f(n)} \,] \subseteq \mathrm{DTIME}[f(n)].$$

- Ops! E allora?!
- E, allora, questi comportamenti "strani" si verificano quando le funzioni limite sono anch'esse "strane"
  - e, se date un'occhiata alla dimostrazione del Gap Theorem, vedete quanto è "strana" f
- E, allora, definiamo un insieme (anzi, due) di funzioni che non sono strane per niente

Funzioni time- e space-constructible

- **Definizione 6.1**: Una funzione totale e calcolabile $f : \mathbb{N} \to \mathbb{N}$ è **time-constructible** se esiste una macchina di Turing *T* di tipo trasduttore che,
  - preso in input un intero n espresso **in notazione unaria** (ossia, come sequenza di n '1'),
- scrive sul nastro output il valore f(n) in unario e $dtime(T,n) \in O(f(n))$.

- **Definizione 6.2**: Una funzione totale calcolabile $f : \mathbb{N} \to \mathbb{N}$ e\` **space-constructible** se esiste una macchina di Turing T di tipo trasduttore che,
  - preso in input il valore n espresso **in notazione unaria**,
- scrive sul nastro output il valore f (n) in unario e $dspace(T,n) \in O(f(n))$.

- D'ora in poi scriveremo 1^n per intendere una sequenza di n '1' – ossia, "n espresso in notazione unaria"
  - per esser chiari: "5 in notazione unaria" = 1^5 = 11111

Funzioni time- e space-constructible

- Attenzione: l'input n di una macchina che testimonia la time-contructibility (o la space-constructibility) di una funzione f deve essere **in notazione unaria**
  - ad esempio, 5 è espresso come 1^5 = 11111
  - questo significa che **la lunghezza dell'input è uguale al valore dell'input: $|n| = n$**
- e quella macchina **scrive sul nastro di output il valore f(n) in notazione unaria**
  - ad esempio, se $f(n)=n^2+3$, la macchina che testimonia la time-contructibility di f scrive 1^{12} = 111111111111 quando calcola f(3)

- Una funzione time-constructible è molto più che una funzione totale e calcolabile
- **è una funzione che può essere calcolata in tempo proporzionale al suo valore**
  - in soldoni, scrivere un '1' sul nastro di output richiede alla macchina che la calcola di eseguire un numero costante di istruzioni (in media)

- E analogamente per le funzioni space-constructible

Funzioni time- e space-constructible

- Tutte le funzioni "regolari" con le quali abbiamo normalmente a che fare sono sia time-constructible che space-constructible – ad esempio
  - tutti i polinomi – ossia, $f(n) = n^k$, con k costante
  - le funzioni esponenziali – ossia, $f(n) = 2^n$, o anche $f(n) = n^n$,
  - e tantissime altre
  - grosso modo, le funzioni "regolari" sono time- e space-constructible

- In Appendice alla dispensa 6 trovate dimostrazioni di time-constructibility
- Sono da considerarsi **utili esercizi**
  - esercizi sul progetto di macchine di Turing
  - esercizi sull'analisi di complessità di macchine di Turing
  - vi invito (per il vostro bene) a svolgerli per conto vostro e poi a confrontare la vostra soluzione con quella che trovate in Appendice

Ciao ciao, gap theorem!

- **La funzione totale calcolabile f : ℕ → ℕ tale che DTIME[$2^{f(n)}$] ⊆ DTIME[f(n)] definita nel gap theorem non è time-constructible!**
- E, infatti valgono i seguenti teoremi
  - **la cui dimostrazione non dovete studiare** (non è neanche riportata sulla dispensa – se vi interessa, la trovate sul libro di testo che avete utilizzato per il primo modulo)

- **Teorema 6.14 [Teorema di gerarchia spaziale]**: *Siano* f : ℕ → ℕ *e* g : ℕ → ℕ *due funzioni tali che f è space-constructible e*

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0 .$$

Allora, DSPACE[g(n)] ⊂ DSPACE[f (n)], *ossia, esiste un linguaggio* L *tale che*

L ∈ DSPACE[f(n)] *e* L ∉ DSPACE[g(n)].

- **Teorema 6.15 [Teorema di gerarchia temporale]**: *Siano* f : ℕ → ℕ *e* g : ℕ → ℕ *due funzioni tali che* f *è time-constructible e*

$$\lim_{n\to\infty} \frac{g(n)\log g(n)}{f(n)} = 0$$

Allora, DTIME[g(n)] ⊂ DTIME[f (n)] *ossia, esiste un linguaggio* L *tale che*

L ∈ DTIME[f(n)] *e* L ∉ DTIME[g(n)].

Ciao ciao, gap theorem!

- Ma qual è il significato dei teoremi di gerarchia spaziale e temporale?
- Se $\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$, allora f(n) cresce “asintoticamente più velocemente” di g(n)
  - ossia, man mano che n cresce, la distanza fra g(n) e f(n) aumenta sempre di più
  - o, se preferite, f(n) diventa enormemente grande per valori di n molto più piccoli di quelli che occorrono a g(n) per diventare altrettanto grande
- E un discorso analogo vale se $\lim_{n \to \infty} \frac{g(n) \log g(n)}{f(n)} = 0$
- Quindi, il teorema di gerarchia temporale ci dice che
  - **quando f è time-constructible**
  - **DTIME[f(n)] non è contenuto in DTIME[g(n)] quando f(n) è *molto* più grande di g(n)** – ad esempio, quando **f (n) = $2^{g(n)}$** !
- E analogamente per quanto afferma il teorema di gerarchia spaziale relativamente alle classi DSPACE quando f è space-constructible