Aniello Murano

Space Complexity

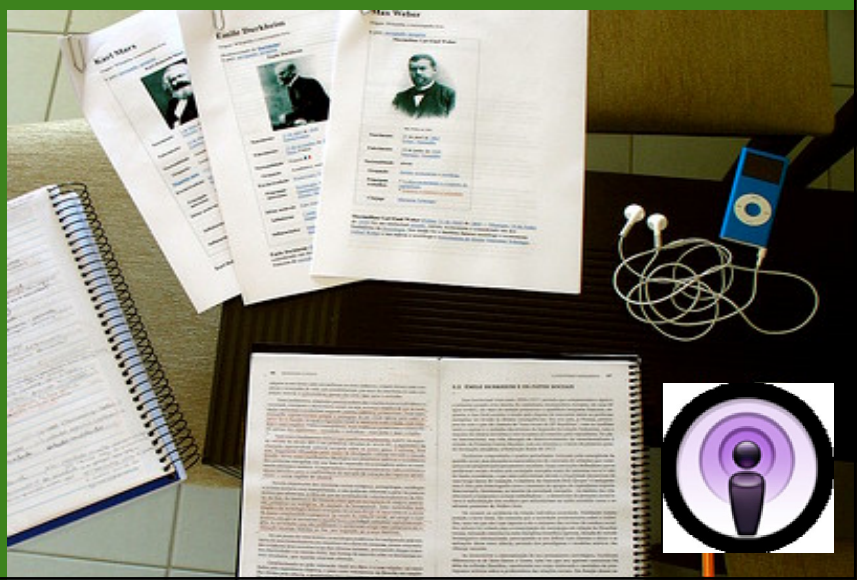
Lezione n.
Parole chiave:
Space

Corso di Laurea:
Informatica

Codice:

Email Docente:
murano@ na.infn.it

A.A. 2008-2009



Definizione Space-complexity

- **Definizione:** Sia M un macchina di Turing determinisca che si ferma su tutti gli input. La complessità di spazio (space complexity) di M è la funzione **f: N→N**, dove f(n) è il massimo numero di celle del nastro che M legge su un input di lunghezza n. Se la complessità di spazio di M è f(n), allora diremo che M lavora in spazio f(n).

  Se M è una macchina di Turing non deterministica dove tutti i rami della computazione si fermano su tutti i possibili input, definiamo la sua complessità di spazio f(n) come il massimo numero di celle del nastro che M vede su tutti i rami della computazione per un input di lunghezza n.

Space e NSpace

- **Definizione:** Sia **f: N→R+** una funzione. Le classi di complessità di spazio *SPACE(f(n))* e *NSPACE((f(n))* sono definite come segue:
  - **SPACE((f(n)) =** {**L** | **L** è un linguaggio deciso da una macchina di Turing deterministica in spazio **O(f(n))**}.
  - **NSPACE(f(n))** = {**L** | **L** è un linguaggio deciso da una macchina di Turing non deterministica in spazio **O(f(n))**}.

Lo Spazio è più potente del Tempo - Esempio 1

- **Esempio 1: SAT è spazio-lineare!**
- Sappiamo che il problema SAT è NP-completo, e che quindi non può essere risolto con una MdT deterministica in tempo polinomiale.
- Lo spazio ha però una caratteristica vincente rispetto al tempo: può essere riutilizzato!
- L'idea per la nostra MdT che risolve il problema SAT in spazio lineare, è quella di assegnare una cella ad ogni variabile, e riutilizzarla ogni volta che facciamo un nuovo assegnamento:
  - **M =** "Su input **< ϕ>** , dove ϕ è una formula booleana con m variabili:
    1. Per ciascun assegnamento vero delle variabili $x_1,...x_m$ di ϕ
    2. Valuta ϕ su queste assegnazioni di verità
    3. Se ϕ viene valutata 1, accept, altrimenti reject. "
- Ogni iterazione del loop richiede solo memoria aggiuntiva per ricordare l'assegnamento corrente di verità e per valutare il valore di ϕ.
- Tutto ciò richiede spazio O(n). Questo spazio può essere riciclato alla successiva iterazione. Così, la complessità di spazio di questo algoritmo rimane lineare.

Not ALL è spazio lineare - Esempio 2

- Sia ALL_{NFA}, il linguaggio degli automi che accettano tutte le stringhe su un dato alfabeto. Formalmente, $ALL_{NFA} = \{\langle A \rangle \mid A \text{ è un NFA ed } L(A)=\Sigma^*\}$
- Sia $Comp(ALL_{NFA})$ il complemento di ALL_{NFA}.
- $Comp(ALL_{NFA})$ è accettato da una MdT nondeterministica in spazio lineare.
- Intuitivamente, il non determinismo è usato per trovare una stringa che sia rifiutata dall'NFA.
- Una quantità di spazio lineare nel numero degli stati dell'automa sotto esame è sufficiente a tenere traccia degli stati in cui l'automa può trovarsi in un determinato momento.
- Si ricordi che ad ogni istante, un automa nondeterministico con Q stati può essere in un insieme di stati pari a 2^Q.
- Formalmente la MdT N che risolve in spazio lineare $Comp(ALL_{NFA})$ è la seguente:
- N = "su ingresso <M>: *(dove <M> è un NFA)*
  1. Si marca lo stato iniziale dell'NFA;
  2. Per 2^Q volte (dove Q è il numero di stati di M): Si seleziona in modo non deteriministico un simbolo d'ingresso e aggiorno le posizioni del marcatore sugli stati di M;
  3. Se tutti i marcatori non vedono uno stato di accettazione, allora accept; altrimenti reject."
- Si noti che ogni iterazione del passo 2 non fa altro che simulare la lettura del simbolo selezionato in modo non deterministico.
- Per costruzione di N, si ha necessità solo di spazio lineare per memorizzare le posizioni dei marcatori e per il contatore del ciclo descritto al passo 2. Spazio che ad ogni iterazione del ciclo è possibile riutilizzare.
- Si noti che al momento non si sa se $Comp(ALL_{NFA})$ è in NP e neppure in Co-NP.

Teorema di Savitch

- Si ricordi che una macchina di Turing deterministica M può sempre simulare una corrispondente macchina di Turing M′ non deterministica con una complessità di tempo esponenziale rispetto alla taglia di M.
- Il seguente *teorema di Savitch* dimostra invece che nello spazio la complessità di questa simulazione aumenta in modo al più quadratico.
- **Teorema di Savitch:** Per ogni funzione $f: N \to R^+$, dove $f(n) \geq n$, abbiamo

$$NSPACE(f(n)) \subseteq SPACE(f^2(n)).$$

- La dimostrazione di questo teorema rivela che una MdT deterministica può simulare una TM non deterministica usando soltanto una piccola parte di spazio aggiuntivo.
- Questo è dovuto al fatto che lo spazio può essere riutilizzato, mentre il tempo no.
- Se potessimo utilizzare lo stesso concetto con il tempo avremmo una prova che P=NP.

Preambolo alla dimostrazione del teorema di Savitch

- Si consideri una MdT non deterministica N di spazio f(n).
- Cerchiamo di simularne il comportamento con una MdT deterministica con spazio polinomiale in f(n).
- Un primo tentativo potrebbe essere quello di simulare tutti i possibili rami di computazione di N, ma questo non permette di sovrascrivere l'area di memoria utilizzata, perché occorre tenere traccia delle scelte nondeterministiche fatte su un ramo specifico per scegliere un prossimo ramo. Le scelte fatte su un ramo sono esponenziali in numero (anche se lo spazio è polinomiale). Dunque, questo tentativo richiede spazio esponenziale.
- Una soluzione polinomiale è invece offerta dalla soluzione (in spazio polinomiale) di un problema intermedio, che consiste nel verificare se N su un generico input w può passare da una configurazione iniziale c_1 a quella finale c_2 in un numero di passi minore o uguale a un numero dato *t* (intuitivamente, t è il numero massimo di operazioni che N necessita per accettare w).
- Chiamiamo questo problema *yieldability problem*, e lo risolviamo definendo la MdT deterministica ricorsiva CANYIELD definita nella prossima diapositiva.

Dimostrazione del teorema di Savitch 1

- **Dimostrazione:**
  - Si Consideri una MdT non deterministica arbitraria N.
  - Preso un intero t positivo, e due configurazioni c_1 e c_2 di N. Diremo che c_1 produce c_2 in un numero minore di passi t se N può andare da c_1 a c_2 in t o meno passi.
  - Il seguente algoritmo ricorsivo deterministico decide il problema della "produzione" quando t è una potenza del 2 ($t=2^p$ per qualche p maggiore o uguale a 0).

  - **CANYIELD(c_1, c_2, 2^p)** = "Con input c_1,c_2 e p, dove **$p \geq 0$** e c_1,c_2 sono configurazioni che usano al più spazio f(n) (se lo spazio occupato è minore possiamo raggiungere spazio f(n) aggiungendo caratteri blank):
    1. Se **p=0**, allora "test se **$c_1=c_2$**" oppure "**c_1** produce **c_2** in un solo step" in base alle derivazioni di **N**. *Accept* se il test ha successo, altrimenti *reject*.
    2. Se **p>0**, allora per ciascuna configurazione **c_m** di **N** che usa spazio **$\leq$f(n)** :
    3. Run **CANYIELD(c_1, c_m, 2^{p-1})** .
    4. Run **CANYIELD(c_m, c_2, 2^{p-1})** .
    5. Se lo step 3 e 4 entrambi accept, allora accept.
    6. Se non hanno entrambi accettato allora, *reject*."

Dimostrazione del teorema di Savitch 2

- Passiamo adesso a definire la MdT deterministica che simula N. Prima abbiamo bisogno di fare alcune assunzioni semplificative (senza perdita di generalità):
  - Quando N accetta, prima di fermarsi pulisce il nastro e ritorna all'inizio del nastro, dove entrerà in una (fissata) configurazione chiamata c_{accept}.
  - Con w indichiamo l'input generico di N, n è la lunghezza di w e c_{start} è la configurazione iniziale di N su w.
- Denotiamo con d una costante tale che N non usa più di $2^{d*f(n)}$ configurazioni per computare w. In pratica, $2^{d*f(n)}$ fornisce un upper bound per il tempo di esecuzione di N su w.
- Con queste assunzioni, abbiamo che N accetta w se e solo se può andare da c_{start} a c_{accept} in $2^{d*f(n)}$ passi o meno. Di conseguenza, la seguente MdT deterministica M simula N se:

- **M** = "Su input **w**:
  - L'output è il risultato di **CANYIELD(c_{start}, c_{accept}, $2^{d*f(n)}$)** ."

Dimostrazione del teorema di Savitch 3

- Visto che CANYIELD risolve l'yieldability problem, M simula correttamente N.
- Rimane adesso da analizzare la complessità di spazio di M.
- Ogni volta che CANYIELD invoca se stesso ricorsivamente, memorizza lo stato corrente e i valori c_1,c_2 e p così da poter essere richiamati al ritorno dalla ricorsione.
- Ciascun livello della ricorsione quindi usa spazio O(f(n)) aggiuntivo.
- Il numero delle chiamate ricorsive è invece pari a d*f(n).
- Dunque, lo spazio totale usato da N è $d*f(n)*O(f(n))=O(f^2(n))$, come voluto.

Dimostrazione del teorema di Savitch 4

- Ritorniamo per un momento sull'affermazione

  M = "Su input **w**: L'output è il risultato di **CANYIELD(c_{start}, c_{accept}, $2^{d*f(n)}$)**"

- C'è una cosa che sembra esserci sfuggito: M deve sapere il valore di f(n) quando invoca CANYIELD!
- Un modo semplice per risolvere questo problema è quello di modificare M (in M′) in modo che provi per f(n) = 1,2,3, ... Per ogni valore f(n) = i, M′ usa CANYIELD per accettare e determinare se la configurazione è raggiungibile (accetta se è questo è il caso).
- Per garantire che M′ non continui ad aumentare i (e quindi a consumare spazio) nei casi in cui N non accetta w, M′ effettua il seguente controllo prima di passare da i a i+1:
  - Utilizzando CANYIELD, controlla che qualche configurazione che utilizzano più di i celle del nastro sia raggiungibile da c_{start}.
  - Se ciò non avviene, non ha alcun senso cercare per i+1, e quindi M' va in reject.
- Per memorizzare il valore i occorre spazio O(log f(n)). Inoltre, questo spazio può essere riciclato ad ogni iterazione.
- Dunque, la complessità dello spazio di M′ rimane $O(f^2(n))$.

Definizione di classe Pspace

- **Definizione: PSPACE** è la classe dei linguaggi che sono decisi in spazio polinomiale da macchine di Turing deterministiche. In altre parole,

  $$\mathbf{PSPACE = SPACE(n) \cup SPACE(n^2) \cup SPACE(n^3) \cup \ldots}$$

NPSPACE può essere definita in modo simile. Tuttavia, quest'ultima non è una classe molto interessante perché, come sappiamo dalle conseguenze del teorema di Savitch, coincide con PSPACE.

Questo è quello che conosciamo:

$$\mathbf{P \subseteq NP \subseteq PSPACE=NPSPACE \subseteq EXPTIME}.$$

Tuttavia non conosciamo se qualcuna delle inclusioni possa essere sostituita con un uguaglianza (un altro dei problemi aperti). Tuttavia però possiamo provare che

$$\mathbf{P \neq EXPTIME}.$$

Quindi sappiamo con sicurezza che l'ultima delle tre inclusioni deve essere necessariamente un inclusione (no un uguaglianza).