Complessità spaziale
o meglio
Complessità nello spazio

- Considera una Turing Machine <u>deterministica</u>
- M che <u>decide</u> un linguaggio L

La complessità spaziale di M (deterministica) è una funzione f:N->N, dove f(n) è il numero massimo di celle che M usa su ogni input di lunghezza n.

Per ogni stringa w la computazione di M(w) termina usando una quantità finita di spazio f(n), (senza limitazione di tempo)

- Considera una Turing Machine non deterministica
- M che <u>decide</u> un linguaggio L

La complessità spaziale di M è una funzione f:N->N, dove f(n) è il numero massimo di celle che per ogni ramo della computazione M usa su ogni input di lunghezza n.

Consideriamo tutte le stringhe di
Lunghezza n e T una funzione da N->N

$Space_M(T(n))$=Linguaggi decidibili (calcolabili) in spazio O(T(n)) deterministicamente per una qualsiasi stringa di lunghezza n

$NSpace_M T((n))$ =Linguaggi decidibili (calcolabili) in spazio O(T(n)) non deterministicamente per una qualsiasi stringa di lunghezza n

Esempio: $L_1 = \{a^n b : n \geq 0\}$

Può essere deciso in spazio $O(n)$

Altri esempi nella stessa classe

$$Space(n)$$

$$L_1 = \{a^n b : n \geq 0\}$$

$$\{ab^n aba : n, k \geq 0\}$$

$$\{b^n : n \text{ is even}\}$$

$$\{b^n : n = 3k\}$$

Esempi nella classe

$$Space(n\ \ )$$

$$\{a^n b^n : n \geq 0\}$$

$$\{ww^R : w \in \{a,b\}\}$$

$$\{ww : w \in \{a,b\}\}$$

Macchine calcolabili in spazio Polinomiale deterministico .

$$Space(O(n^k))$$

costante $k > 0$

Macchine calcolabili in spazio Polinomiale Non deterministico.

$$NSpace(O(n^k))$$

costante $k > 0$

chiaramente: $$Space(n^{k+1}) \supset Space(n^k)$$

$$NSpace(n^{k+1}) \supset NSpace(n^k)$$

La Classi di Complessità spaziale

$$PSpace = \bigcup_{k>0} PSpace(n^k)$$

Rapresenta:

- Algoritmi deterministici che usano spazio polinomiale
  - Problemi "trattabili nello spazio"

$$NPSpace = \bigcup_{k>0} NPSpace(n^k) \quad NP$$

Rapresenta:

- Algoritmi non deterministici che usano spazio polinomiale tempo complessità:

Il numero di passi (step)
durante una computazione

spazio complessità:

spazio usato
durante una computazione

Ricorda che una macchina di Turing deterministica M simula una macchina di Turing M' non deterministica con una complessità di tempo esponenziale rispetto alla complessità di M.

.

Teorema di Savitch: Per ogni funzione **f: N→R⁺**, dove **f(n)≥n**, abbiamo
NSPACE(f(n)) ⊆SPACE(f^2 (n)).

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

Teorema di Savitch:

Per ogni funzione **f**: $\mathbf{N} \rightarrow \mathbf{R}^+$, dove **f(n)≥n**, abbiamo

$$\mathbf{NSPACE(O(f(n)))} \subseteq \mathbf{SPACE}(O(f^2(n))).$$

teorema di Savitch dimostra che nello spazio la complessità di questa simulazione, non determinismo -> determinismo, aumenta in modo al più quadratico.

Si consideri una NMdT (non deterministica) calcolata in **spazio f(n).**

Cerchiamo di simularla il comportamento con una MdT (deterministica) con spazio polinomiale f^2**(n).**

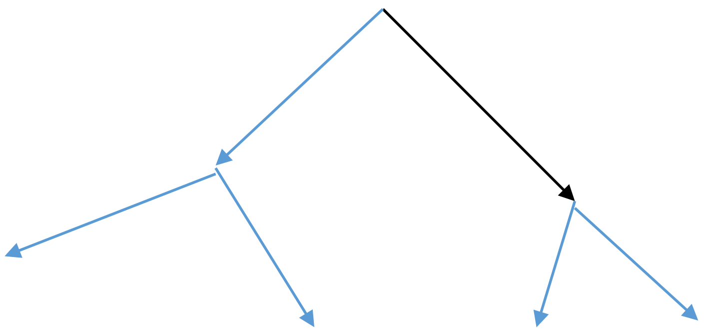
Un primo tentativo potrebbe essere quello di simulare tutti i possibili rami di computazione (di lunghezza finita) di NMdT, ma questo non permette di sovrascrivere l'area di memoria utilizzata, perché occorre tenere traccia delle scelte nondeterministiche fatte su un ramo specifico per scegliere un prossimo ramo.

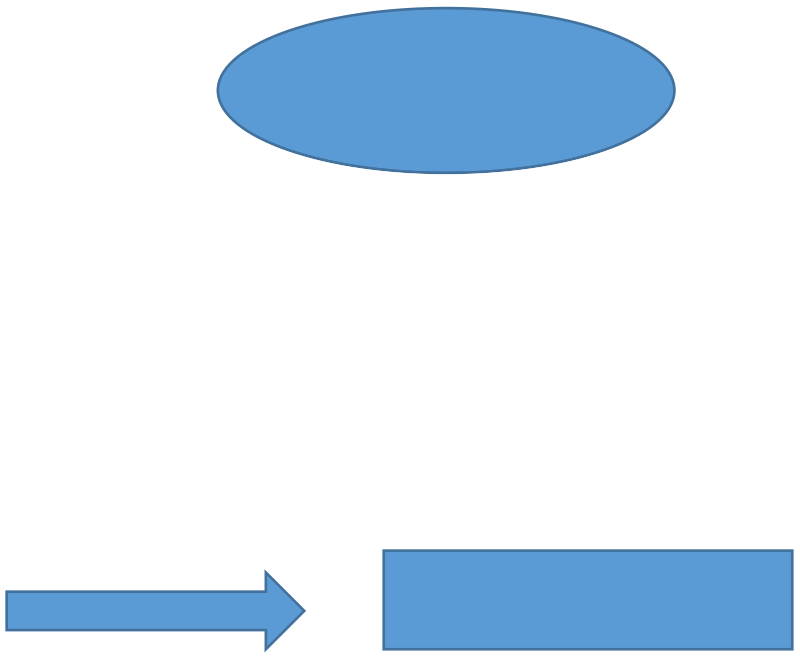
1, 2 11,12, 111,112,121, 122 …….

Ogni passo deve tenere traccia del cammino percorso. Nota che se usiamo uno spazio $f(n)$ il tempo che possiamo usare su quello spazio è $2^{O(f(n))}$ step (perché?) ed ogni step può essere una scelta non deterministica e quindi può accadere di dover memorizzare stringhe di lunghezza $2^{O(f(n))}$

Le scelte fatte per percorrere tutti i possibili cammini sono esponenziali in numero, anche se lo spazio necessario per calcolarli è polinomiale.
Dunque, questo tentativo richiede spazio esponenziale.

Metodo
Non
determinismo
Verso
determinismo
Perché non
funziona





1,2,11,12,21,22, …

Memoria che ci serve per percorrere tutti i cammini.
Esponenziale

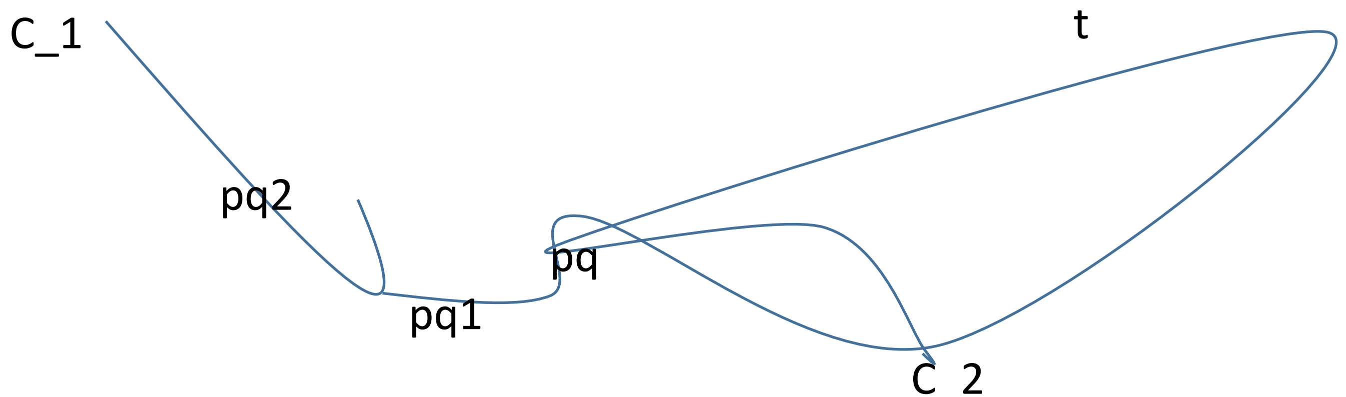
Memoria che si può riutilizzare

YELDABILITY PROBLEM

Verificare se una MNdT con 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 (time). (dove t verrà scelto come il numero massimo di operazioni che MNdT necessita per accettare w).

Si consideri una MdT non deterministica arbitraria.

Preso un intero t positivo, e due configurazioni c_1 e c_2 di NMdT Diremo che c1 produce c2 in un numero minore o uguali di passi t se NMdT può produrre c_2 a partire da c_1 in t (o meno passi).



input c_1,c_2 e t, dove **t≥0.**
c_1,c_2 sono configurazioni che usano al più uno spazio f(n) (se lo spazio occupato è minore possiamo raggiungere spazio f(n) aggiungendo caratteri blank);

sia t una potenza del 2 ($t=2^p$ per qualche p maggiore o uguale a 0), possibile?.

CANYELD ha come input anche tutti gli stati e i collegamenti tra di loro ovvero la delta

CANYIELD(c_1, c_2, 2^t) =
1. Se p=0, allora "test (se c_1=c_2" oppure "c_1 produce c_2 in un solo step)" *(in base alle delta di MNdT).*
Accept se il test ha successo, altrimenti Reject.

2. Se t>0, allora per ciascuna (tutte) configurazione c_m di N (che usano spazio ≤f(n)):
3. Run CANYIELD(c_1, c_m, 2^{t-1}).
4. Run CANYIELD(c_m, c_2, 2^{t-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

Passiamo a definire la MdT deterministica che simula MNdT. Prima abbiamo bisogno di fare una assunzione semplificativa:

1. Quando MNdT accetta, prima di fermarsi pulisce il nastro e ritorna all'inizio del nastro, dove entrerà in una (fissata) configurazione chiamata c_{accept}.

Notazione: Con w indichiamo l'input di N, n è la lunghezza di w e c_{start} è la configurazione iniziale di MNdT.

Denotiamo con d una costante tale che MNdT non usa più di d*f(n) configurazioni per computare w.

In pratica, $2^{d*f(n)}$ fornisce un upper bound per il tempo di esecuzione di MNdT su w (perché).

Con queste assunzioni, abbiamo che MNdT accetta w se e solo se si può andare da c_{start} a c_{accept} in al massimo $2^{d*f(n)}$ passi.

Analizziamo la seguente MdT deterministica:

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

Quindi M simula NMdT

Analizziamo la complessità di spazio di M.

- Ogni volta che CANYIELD si autochiama (ricorsione) memorizza lo stato corrente (i valori c_1,c_2 e t) così da poter essere richiamati (usati) 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: **$d*f(n)*O(f(n))=O(f^2(n))$**

Ritorniamo per un momento sull'affermazione

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

M deve sapere il valore di f(n) quando invoca CANYIELD, questo lo dobbiamo calcolare. Quindi la complessità aumenta.

Un modo semplice per risolvere questo problema è quello di modificare M (chiamata 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.

Per garantire che M’ non continui ad aumentare i (e quindi a consumare spazio) nei casi in cui NMdT non accetta w, si effettua il seguente controllo prima di passare da i a i+1 (nuovo valore di f(n)):
Utilizzando CANYIELD, controlla che MNdT può raggiungere una configurazione che utilizza più di i celle del nastro (da c_{start}). Se questo non accade: non ha alcun senso cercare per i+1 quindi M’ va in reject.

Per memorizzare il valore i (f(n)): occorre spazio O(log f(n)). Inoltre, questo spazio può essere riciclato ad ogni iterazione.
Dunque, la complessità dello spazio di M’ rimane $O(f^2(n))$.

Fine corso

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.

Sappiamo che

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

Quindi sappiamo che l'ultima delle tre inclusioni deve essere necessariamente un inclusione (non una uguaglianza).

:

$P \subseteq NP \subseteq PSPACE=NPSPACE \subseteq EXPTIME$.

$P \neq EXPTIME$.

Questo corso è dedicato alla città e agli abitanti che mi hanno ospitato in questo periodo.