Lezione di Informatica Teorica: Classi di Complessità Spaziale

Appunti da Trascrizione Automatica

30 giugno 2025

Indice

1 Introduzione alla Complessità Spaziale

Finora ci siamo concentrati sulla *complessità temporale*, ovvero il tempo impiegato da un algoritmo. Ora introduciamo la *complessità spaziale*, che misura la quantità di memoria necessaria per l'esecuzione di un algoritmo. Intuitivamente, una macchina con tempo polinomiale è meno potente di una macchina con spazio polinomiale, perché lo spazio può essere riutilizzato, permettendo alla macchina di eseguire più a lungo. Come vedremo, la classe **P** (problemi risolvibili in tempo polinomiale) è contenuta in **PSPACE** (problemi risolvibili in spazio polinomiale). Tuttavia, non sappiamo se **P** sia strettamente contenuta in **PSPACE**, un problema analogo a **P** vs **NP**.

1.1 Modello di Macchina di Turing per la Complessità Spaziale

Per definire la complessità spaziale, in particolare per classi con vincoli di spazio molto ridotti (es. logaritmico), utilizziamo un modello di Macchina di Turing (MT) leggermente modificato:

- **Nastro di Input:** Un nastro di sola lettura. La testina può muoversi in entrambe le direzioni, ma non può modificare il contenuto. Questo nastro contiene l'input W.
- **Nastro di Lavoro (Work Tape):** Un nastro infinito di lettura-scrittura. La testina può muoversi in entrambe le direzioni e modificare il contenuto. Questo nastro è usato per i calcoli intermedi e lo spazio utilizzato su questo nastro sarà quello che misureremo.
- **Nastro di Output (solo per trasduttori):** Un nastro di sola scrittura. La testina può muoversi solo in una direzione (generalmente a destra). Questo nastro viene usato per produrre l'output finale.

La separazione dei nastri è cruciale per le classi a basso spazio. Se misurassimo lo spazio sull'input tape, qualsiasi algoritmo che legge l'intero input userebbe almeno spazio lineare, rendendo difficile definire classi come LogSpace.

1.2 Definizioni Fondamentali

Definizione 1.1 (Computation Space)**.** *Il **Computation Space** di una macchina di Turing M su un input W, denotato $Space_M(W)$, è il numero di celle* distinte *visitate (anche solo lette, non necessariamente scritte) da M sul suo* nastro di lavoro *durante il processo di W.*

- *Se M è **deterministica**, $Space_M(W)$ è semplicemente il numero di celle distinte visitate.*
- *Se M è **non deterministica**, $Space_M(W)$ è il* massimo *numero di celle distinte visitate su work tape in qualsiasi dei suoi branch di computazione.*

Definizione 1.2 (Space Function)**.** *Una funzione $S : \mathbb{N} \to \mathbb{N}$ è una **Space Function** se $S(n)$ è strettamente positiva per ogni n ed è non-decreasing.*

Definizione 1.3 (Running Space)**.** *Il **Running Space** di una macchina di Turing M è $S(n)$ se per ogni stringa W (a parte un numero finito di eccezioni), il $ComputationSpace$ di M su W è bounded da $S(|W|)$, ovvero $Space_M(W) \in O(S(|W|))$.*

1.3 Classi di Complessità Spaziale

Similmente alle classi temporali (DTIME, NTIME), definiamo le classi spaziali:

Definizione 1.4 (DSPACE($S(n)$))**.** ***DSPACE($S(n)$)*** *è l'insieme di tutti i linguaggi L tali che esiste una Macchina di Turing deterministica M che decide L e il cui Running Space è in $O(S(n))$.*

$$DSPACE(S(n)) = \{L \mid \exists M \text{ deterministica t.c. } L(M) = L \text{ e } Space_M(n) \in O(S(n))\}$$

Definizione 1.5 (NSPACE($S(n)$))**.** ***NSPACE($S(n)$)*** *è l'insieme di tutti i linguaggi L tali che esiste una Macchina di Turing non deterministica N che decide L e il cui Running Space è in $O(S(n))$.*

$$NSPACE(S(n)) = \{L \mid \exists N \text{ non-deterministica t.c. } L(N) = L \text{ e } Space_N(n) \in O(S(n))\}$$

Definizione 1.6 (Classi Logaritmiche)**.** *Definiamo due classi di complessità spaziale fondamentali:*

- ***L (LogSpace):*** *La classe di linguaggi decidibili da una MT deterministica in spazio logaritmico.*

$$L = DSPACE(\log n)$$

- ***NL (Nondeterministic LogSpace):*** *La classe di linguaggi decidibili da una MT non deterministica in spazio logaritmico.*

$$NL = NSPACE(\log n)$$

Proposizione 1.1. *Vale la relazione di contenimento: $L \subseteq NL$.*

Analogamente a P vs NP, non sappiamo se L = NL o se L $\subset$ NL.

2 Esempi di Problemi nelle Classi Spaziali

2.1 Linguaggio 0^n1^n in L

Consideriamo il linguaggio $L_0 = \{0^n1^n \mid n > 0\}$. Avevamo visto un algoritmo deterministico per questo linguaggio che marcava o cancellava i simboli sull'input tape, ma questo non è permesso con il nuovo modello di MT. Se copiassimo l'input sul work tape, useremmo spazio lineare, quindi $L_0 \notin$ L con quell'algoritmo.

Esempio 2.1 (Decisore per 0^n1^n in LogSpace)**.** *Per decidere L_0 in spazio logaritmico, la MT deterministica può fare quanto segue:*

1. *Scorre l'input tape, contando il numero di* 0 *e memorizzando il conteggio sul work tape in binario. La rappresentazione binaria di n (numero di* 0*s) occupa $O(\log n)$ spazio.*
2. *Una volta terminati gli* 0*s, riposiziona la testina dell'input tape all'inizio degli* 1*s.*
3. *Scorre gli* 1*s, contando il loro numero e memorizzando il conteggio sul work tape in binario, sovrascrivendo il conteggio precedente.*
4. *Confronta i due conteggi binari (uno dopo gli* 0*s, uno dopo gli* 1*s). Se sono uguali e la stringa è della forma corretta (*0...01...1*), la MT accetta; altrimenti, rigetta.*

Lo spazio utilizzato sul work tape è limitato alla memorizzazione di pochi numeri binari (il conteggio, eventuali puntatori, ecc.), ciascuno dei quali occupa $O(\log n)$ spazio. Pertanto, $L_0 \in L$.

2.2 Problema della Raggiungibilità (REACHABILITY) in NL

Il problema **REACHABILITY** (o **ST-Connectivity**) è un problema classico in teoria dei grafi.

Definizione 2.1 (REACHABILITY). *Sia $G = (V, E)$ un grafo orientato, e siano $s, t \in V$ due vertici. Il problema **REACHABILITY** consiste nel decidere se esiste un cammino (path) da s a t in G.*

$$REACHABILITY = \{(G, s, t) \mid G \text{ è un grafo orientato e esiste un cammino da } s \text{ a } t \text{ in } G\}$$

Algoritmi standard come BFS (Breadth-First Search) o DFS (Depth-First Search) risolvono REACHABILITY in tempo polinomiale, ma tipicamente richiedono spazio $O(|V| + |E|)$ o $O(|V|)$ per memorizzare i nodi visitati o le code/stack. Questo è spazio lineare, non logaritmico.

Esempio 2.2 (Algoritmo non deterministico per REACHABILITY in LogSpace). *Sia M una MT non deterministica. Per decidere se esiste un cammino da s a t in $G = (V, E)$, M esegue il seguente algoritmo:*

```
Function NDTM_REACHABILITY(G, s, t):
    # G: grafo, s: nodo sorgente, t: nodo destinazione

    current_node = s  # Inizializza il nodo corrente con la sorgente
    path_length_counter = 1 # Contatore della lunghezza del cammino (numero di nodi visitati)

    # Se il nodo di partenza è già il target, accetta immediatamente
    if current_node == t:
        ACCEPT

    # La MT "guessta" i passi successivi
    while path_length_counter <= |V|:
        # Non deterministicamente, "guessta" un vertice successivo
        # Il guess non richiede di memorizzare tutti i possibili successori,
        # ma solo di selezionarne uno alla volta.
        next_node = GUESS_VERTEX_FROM_V()

        # Verifica se esiste un arco da current_node a next_node in G
        # L'accesso al grafo G avviene tramite il nastro di input, senza occupare spazio sul work
        if (current_node, next_node) in E:
            current_node = next_node
            path_length_counter = path_length_counter + 1

            # Se siamo arrivati al target, il cammino è stato trovato
            if current_node == t:
                ACCEPT
            # Altrimenti, continua a gueessare
        else:
            # Se il guess non porta a un arco valido, questo ramo di computazione RIGETTA
            REJECT

    # Se il contatore supera |V|, significa che siamo entrati in un ciclo
    # o abbiamo esplorato un cammino troppo lungo senza trovare t.
    # In ogni caso, questo ramo di computazione RIGETTA.
```

35 *REJECT*

Analisi dello Spazio:

- `current_node`*: per memorizzare l'identificativo di un nodo in un grafo con* $|V|$ *nodi, sono necessari* $O(\log |V|)$ *bit.*
- `next_node`*: stessa cosa,* $O(\log |V|)$ *bit.*
- `path_length_counter`*: un contatore fino a* $|V|$*, richiede* $O(\log |V|)$ *bit.*

In totale, lo spazio utilizzato sul nastro di lavoro è $O(\log |V|)$*, che è* $O(\log n)$ *dove n è la dimensione dell'input (che include la rappresentazione del grafo, quindi* $|V|$ *è* $O(n)$*).*

Correttezza:

- ***Completezza:*** *Se esiste un cammino da s a t in G, allora esiste un ramo di computazione non deterministica che sceglie correttamente i nodi successivi lungo quel cammino e accetta quando raggiunge t. Il vincolo* `path_length_counter` $\leq |V|$ *assicura che anche in presenza di cicli, il cammino non venga esplorato all'infinito e la MT termini. Se un cammino esiste, ne esiste uno senza cicli di lunghezza al più* $|V| - 1$*.*
- ***Soundness:*** *Se la MT accetta, significa che ha trovato una sequenza di nodi validi che formano un cammino da s a t.*

Poiché l'algoritmo non deterministico decide correttamente REACHABILITY e usa spazio logaritmico, concludiamo che ***REACHABILITY*** $\in$ *NL.*

3 Riduzioni LogSpace e NL-Completezza

Per studiare le relazioni tra le classi di complessità spaziale (in particolare tra L e NL), è necessario definire un tipo di riduzione più restrittivo rispetto alle riduzioni polinomiali usate per NP-Completezza. Le riduzioni polinomiali sono "troppo potenti" per le classi a basso spazio (es. L $\subseteq$ P). Se un problema in L fosse NL-completo sotto riduzione polinomiale, non si otterrebbe una buona separazione.

3.1 Trasduttore LogSpace

Per definire le riduzioni LogSpace, introduciamo il concetto di trasduttore.

Definizione 3.1 (Trasduttore LogSpace)**.** *Un* ***Trasduttore LogSpace*** *è una Macchina di Turing che calcola una funzione* $f : \Sigma^* \to \Gamma^*$ *e che soddisfa le seguenti proprietà:*

- *Ha un nastro di input (sola lettura), un nastro di lavoro (lettura-scrittura) e un nastro di output (sola scrittura, unidirezionale).*
- *Il suo spazio di lavoro (sull'work tape) è limitato a* $O(\log n)$ *dove n è la lunghezza dell'input.*

Il nastro di output è di sola scrittura e unidirezionale per assicurarsi che lo spazio speso per l'output non venga conteggiato come spazio di lavoro (non è riutilizzabile per calcoli intermedi) e che il trasduttore non possa usare l'output tape come work tape ausiliario.

3.2 Riduzione LogSpace

Definizione 3.2 (Riduzione LogSpace ($\leq_L$))**.** *Siano A e B due linguaggi. Diciamo che A si riduce in LogSpace a B, denotato $A \leq_L B$, se esiste una funzione $f : \Sigma^* \to \Gamma^*$ calcolabile da un Trasduttore LogSpace, tale che per ogni stringa w:*

$$w \in A \iff f(w) \in B$$

Le riduzioni LogSpace godono di proprietà di transitività, analoghe alle riduzioni polinomiali.

3.3 NL-Completezza

Definizione 3.3 (NL-Completo)**.** *Un linguaggio L_{NL} è **NL-Completo** se soddisfa due condizioni:*

1. ***Appartenenza:*** *$L_{NL} \in NL$ (Membership).*
2. ***Durezza (Hardness):*** *Per ogni linguaggio $L' \in NL$, si ha $L' \leq_L L_{NL}$ (NL-Hardness).*

Analogamente al Teorema di Cook-Levin per NP-completezza, un linguaggio NL-completo è considerato il più difficile nella classe NL. Un risultato importante afferma che:

Teorema 3.1. *Se un linguaggio NL-Completo appartiene a L, allora $L = NL$.*

3.4 REACHABILITY è NL-Completo

Teorema 3.2 (NL-Completezza di REACHABILITY)**.** *Il problema **REACHABILITY** è NL-Completo.*

Dimostrazione. Dobbiamo dimostrare due cose:

1. **REACHABILITY $\in$ NL:** Già dimostrato con l'algoritmo non deterministico in LogSpace.
2. **REACHABILITY è NL-Hard:** Dobbiamo mostrare che per ogni linguaggio $L' \in$ NL, si ha $L' \leq_L$ REACHABILITY.

Sia L' un linguaggio arbitrario in NL. Per definizione, esiste una Macchina di Turing non deterministica M che decide L' usando $O(\log n)$ spazio sul suo nastro di lavoro, dove n è la lunghezza dell'input. Per dimostrare che $L' \leq_L$ REACHABILITY, dobbiamo costruire una funzione f (calcolabile da un trasduttore LogSpace) che prende in input una stringa w e produce un'istanza (G_w, s, t) di REACHABILITY, tale che $w \in L' \iff (G_w, s, t) \in$ REACHABILITY.

L'idea chiave è rappresentare le computazioni di M su w come cammini in un grafo. I nodi del grafo G_w saranno le *configurazioni istantanee* (IDs) di M.

Configurazione di una MT M: Una configurazione istantanea (ID) di una MT M in un dato momento della computazione è definita dallo stato corrente di M, dal contenuto del nastro di lavoro e dalla posizione di entrambe le testine (input e lavoro).

- Stato corrente $q \in Q$ (insieme degli stati di M). Numero di stati è costante.
- Posizione testina input $h_1 \in \{1, \ldots, |w|\}$. Richiede $O(\log |w|)$ bit.
- Posizione testina lavoro $h_2 \in \{1, \ldots, S(|w|)\}$. Poiché $S(|w|) \in O(\log |w|)$, h_2 richiede $O(\log(\log |w|))$ bit.
- Contenuto nastro di lavoro $C \in \Gamma^{S(|w|)}$ (simboli dell'alfabeto del nastro). Poiché $S(|w|) \in O(\log |w|)$, il contenuto richiede $O(\log |w|)$ simboli.

Una configurazione può essere rappresentata da una tupla (q, h_1, h_2, C). La lunghezza di questa tupla (come stringa binaria) è $O(\log |w|)$. Il numero totale di configurazioni possibili è esponenziale in $O(\log |w|)$, il che è polinomiale in $|w|$ (es. $2^{c \log n} = n^c$).

Costruzione del Grafo $G_w = (V, E)$:

- **Vertici V:** L'insieme dei vertici di G_w è l'insieme di tutte le possibili configurazioni di M su w.
- **Archi E:** Esiste un arco diretto da una configurazione C_i a una configurazione C_j se e solo se M può passare da C_i a C_j in un singolo passo di computazione (secondo la funzione di transizione di M). Poiché M è non deterministica, da una configurazione possono partire più archi.
- **Nodi speciali:**
  - s_{start}: Rappresenta la configurazione iniziale di M su w (stato iniziale q_0, testine all'inizio, nastro di lavoro vuoto).
  - t_{final}: Un nodo speciale aggiunto al grafo, non corrispondente a una configurazione di M. Tutti gli archi che provengono da configurazioni accettanti di M (quelle in cui M entra in uno stato di accettazione) puntano a t_{final}.

Funzionamento del Trasduttore f (la riduzione LogSpace): Il trasduttore f deve generare $(G_w, s_{start}, t_{final})$ sull'output tape usando solo spazio logaritmico sul work tape. Non può costruire l'intero grafo in memoria, dato che G_w può avere un numero polinomiale di nodi e archi.

1. **Generazione di s_{start} e t_{final}:** Questi sono facili da generare, rappresentano configurazioni specifiche o un simbolo speciale. Richiedono spazio costante o logaritmico.
2. **Generazione dei vertici di G_w:** Il trasduttore itera attraverso tutte le possibili combinazioni dei parametri di una configurazione (q, h_1, h_2, C). Per ogni combinazione valida (cioè, che rispetta i limiti di dimensione e stati di M), il trasduttore la scrive sul nastro di output come un nodo del grafo. Questo processo richiede solo spazio logaritmico per memorizzare la configurazione corrente in fase di iterazione.
3. **Generazione degli archi di G_w:** Il trasduttore itera attraverso tutte le possibili coppie di configurazioni (C_i, C_j). Per ogni coppia, verifica se M può passare da C_i a C_j in un passo. Se sì, scrive l'arco (C_i, C_j) sul nastro di output. Questo richiede solo spazio logaritmico per memorizzare C_i e C_j e simulare il singolo passo di M. Gli archi dalle configurazioni accettanti a t_{final} sono generati analogamente.

La verifica della validità di una configurazione o della transizione tra due configurazioni può essere fatta deterministicamente in spazio logaritmico, poiché coinvolge solo la lettura di pochi bit (lo stato corrente, i simboli sotto le testine, le transizioni di M) e semplici calcoli. Il trasduttore f scrive l'output progressivamente, senza memorizzare l'intero grafo in memoria di lavoro.

Equivalenza:

- Se $w \in L'$, allora esiste una sequenza di configurazioni $C_0, C_1, \ldots, C_k$ tale che $C_0 = s_{start}$, C_k è una configurazione accettante, e ogni C_{i+1} è un successore legale di C_i. Questo cammino corrisponde a un cammino da s_{start} a t_{final} nel grafo G_w. Quindi $(G_w, s_{start}, t_{final}) \in$ REACHABILITY.

- Se $(G_w, s_{start}, t_{final}) \in$ REACHABILITY, allora esiste un cammino da s_{start} a t_{final} in G_w. Questo cammino rappresenta una sequenza di configurazioni $C_0, C_1, \ldots, C_k$ di M che parte da s_{start} e arriva a una configurazione accettante (che poi si collega a t_{final}). Quindi M accetta w, il che significa $w \in L'$.

Dato che f è calcolabile da un trasduttore LogSpace, abbiamo dimostrato che $L' \leq_L$ REACHABILITY per ogni $L' \in$ NL. Quindi, **REACHABILITY** è NL-Complete. □

Questo teorema ha implicazioni importanti: se qualcuno trovasse un algoritmo deterministico in LogSpace per **REACHABILITY**, allora L $=$ NL. Dato che finora non è stato trovato tale algoritmo, si tende a credere che L $\neq$ NL, ovvero che il non determinismo aggiunga un potere significativo quando lo spazio è limitato a $\log n$.