Lezione di Informatica Teorica: Space Complexity Avanzata

Appunti da Trascrizione Automatica

30 giugno 2025

Indice

1 Introduzione e Richiami

Oggi proseguiamo il discorso iniziato ieri sulla Complessità Spaziale, esplorando ulteriori classi di complessità e risultati importanti. Ieri abbiamo introdotto le classi **DSPACE** e **NSPACE**, e ci siamo focalizzati in particolare su **L** (**DSPACE**$(\log n)$) e **NL** (**NSPACE**$(\log n)$). Abbiamo visto che il problema **REACHABILITY** (raggiungibilità in un grafo orientato) è **NL**-completo.

Sebbene non sia dimostrato che **REACHABILITY** appartenga a **L** (il che implicherebbe **L** = **NL**), vedremo oggi un risultato interessante sulla sua complessità in spazio deterministico che sarà fondamentale per un teorema più generale. Non si ritiene che **REACHABILITY** sia in **L**.

2 Complessità di REACHABILITY in DSPACE

Nonostante non si sappia se **REACHABILITY** sia risolvibile in spazio logaritmico deterministico (**L**), siamo in grado di dimostrare che appartiene a **DSPACE**$(\log^2 n)$, ovvero spazio polinomiale nel logaritmo della dimensione dell'input n. Questo è un risultato significativo in quanto mostra che è comunque risolvibile in spazio polilogaritmico.

L'algoritmo che utilizzeremo per dimostrare questo risultato è di tipo ricorsivo e sfrutta una strategia di "ricerca binaria" sulla lunghezza dei cammini.

2.1 Principio dell'Algoritmo

L'osservazione chiave è la seguente: se esiste un cammino da un nodo sorgente S a un nodo destinazione T di lunghezza al più K, allora deve esistere un nodo intermedio U tale che esista un cammino da S a U e un cammino da U a T, entrambi di lunghezza al più $K/2$. Questa idea, combinata con il riuso dello spazio, ci permette di progettare un algoritmo efficiente in termini di memoria.

Intuitivamente, se vogliamo determinare se esiste un cammino da S a T, possiamo iterare su tutti i possibili nodi intermedi U e verificare ricorsivamente se esistono cammini da S a U e da U a T entrambi di lunghezza al più $K/2$.

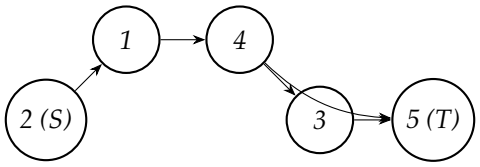
2.2 Algoritmo `exists-path`

Definiamo un algoritmo ricorsivo `exists-path`(G, S, T, K) che restituisce `vero` se nel grafo orientato $G = (V, A)$ esiste un cammino da S a T di lunghezza al più K.

```
function exists-path(G, S, T, K):
    # Caso Base 1: Se K=0, S e T devono coincidere
    if K == 0:
        if S == T:
            return TRUE
        else:
            return FALSE

    # Caso Base 2: Se K=1, S e T devono essere collegati direttamente da un arco
    if K == 1:
        if (S, T) in G.A: # G.A è l'insieme degli archi di G
            return TRUE
```

```
        else:
            return FALSE

    # Passo Ricorsivo: K > 1
    # Iteriamo su tutti i possibili nodi intermedi U
    for each U in G.V:
        # Verifichiamo ricorsivamente l'esistenza di due cammini più corti
        # ceil(K/2) assicura che K si riduca correttamente anche per K dispari.
        if exists-path(G, S, U, ceil(K/2)) and \
           exists-path(G, U, T, ceil(K/2)):
            return TRUE # Trovato un cammino, possiamo terminare

    # Se nessuna U intermedia porta a un cammino, non esiste un cammino
    return FALSE
```

Esempio 1. *Consideriamo il seguente grafo orientato:*



Vogliamo sapere se esiste un cammino da $S = 2$ a $T = 5$. Inizialmente chiamiamo `exists-path`$(G, 2, 5, |V|)$ *dove $|V|$ è il numero di vertici (5 in questo caso), per cercare un cammino di lunghezza arbitraria ma al massimo il numero di vertici. Il cammino diretto è* $2 \to 1 \to 4 \to 3 \to 5$.

2.3 Analisi della Complessità Spaziale di `exists-path`

Per analizzare la complessità spaziale, dobbiamo considerare lo spazio richiesto da una singola chiamata della funzione e l'altezza massima dello stack di chiamate ricorsive.

1. **Spazio per una singola chiamata**: Una singola chiamata a `exists-path` deve memorizzare i parametri S, T, K e la variabile U nel ciclo `for`. * S, T, U: sono identificativi di nodi. Per un grafo con N nodi, un identificativo richiede $O(\log N)$ bit di spazio (ad esempio, un indice numerico). * K: rappresenta la lunghezza massima del cammino. Inizialmente K può essere al massimo N (il numero di nodi). Quindi K richiede $O(\log N)$ bit. * In totale, una singola chiamata richiede $O(\log N)$ spazio sul *work tape*. Questo spazio viene riutilizzato per ogni iterazione del ciclo `for` (cambiando U) e per ogni chiamata ricorsiva (passando nuovi S, T, K).

2. **Altezza dello stack di chiamate ricorsive**: L'algoritmo ricorsivo dimezza il parametro K ad ogni passo (da K a $\lceil K/2 \rceil$). Questo significa che il numero di livelli di ricorsione è logaritmico rispetto al valore iniziale di K. Poiché K può essere al massimo il numero di nodi N, la profondità massima di ricorsione è $O(\log N)$. Durante l'esecuzione, il ciclo `for each U` implica che le chiamate ricorsive per ogni U non sono eseguite in parallelo. L'algoritmo esplora un ramo dell'albero di ricorsione (corrispondente a una scelta di U) e solo dopo aver completato quel ramo (o aver trovato un `TRUE`) passa al successivo U. Questo significa che, in memoria, lo stack contiene al più una sequenza di chiamate ricorsive in un singolo "percorso" attraverso l'albero di computazione (un ramo di ricerca in profondità).

Combinando questi due fattori, lo spazio totale richiesto è il prodotto dello spazio per una singola chiamata per l'altezza massima dello stack:

$$\text{Spazio totale} = O(\text{Spazio per chiamata}) \times O(\text{Profondità stack})$$

$$\text{Spazio totale} = O(\log N) \times O(\log N) = O(\log^2 N)$$

Poiché in informatica teorica N (numero di nodi) è solitamente uguale o proporzionale a n (dimensione dell'input del problema), possiamo affermare che **REACHABILITY** può essere risolto in $\mathbf{DSPACE}(\log^2 n)$.

3 Il Teorema di Savitch

Il Teorema di Savitch è uno dei risultati più importanti nella teoria della complessità spaziale, poiché stabilisce una relazione sorprendente tra le classi di complessità spaziali deterministiche e non deterministiche.

Teorema 1 (Teorema di Savitch)**.** *Sia $S(n)$ una funzione di spazio, tale che $S(n) \geq \Omega(\log n)$. Allora,*

$$\mathbf{NSPACE}(S(n)) \subseteq \mathbf{DSPACE}(S(n)^2)$$

Questo teorema implica che, per funzioni di spazio che crescono almeno logaritmicamente, qualsiasi problema risolvibile in spazio non deterministico $S(n)$ può essere risolto in spazio deterministico $S(n)^2$. Questo è un contrasto netto con le classi di complessità temporale, dove il passaggio dal non deterministico al deterministico (ad esempio da **NP** a **P**) è congetturato essere esponenziale (ovvero $\mathbf{P} \neq \mathbf{NP}$). Per lo spazio, il "costo" della determinizzazione è solamente quadratico.

3.1 Sketch della Dimostrazione del Teorema di Savitch

L'idea della dimostrazione si basa su due concetti che abbiamo già introdotto: 1. La riduzione di qualsiasi linguaggio in $\mathbf{NSPACE}(S(n))$ al problema **REACHABILITY** su un grafo di configurazioni. 2. L'algoritmo `exists-path` per **REACHABILITY** in $\mathbf{DSPACE}(\log^2 n)$.

1. **Linguaggio e Macchina Non Deterministica:** Sia L un linguaggio appartenente a $\mathbf{NSPACE}(S(n))$. Per definizione, esiste una macchina di Turing non deterministica M che decide L e utilizza $O(S(n))$ spazio sul nastro di lavoro.

2. **Costruzione del Computation Graph:** Per simulare M deterministicamente, possiamo costruire il suo *computation graph* (o grafo delle configurazioni). I nodi di questo grafo rappresentano tutte le possibili *configurazioni* di M, e un arco (C_1, C_2) esiste se M può passare da C_1 a C_2 in un singolo passo.

3. **Dimensioni di una Configurazione:** Una configurazione di una macchina di Turing tipicamente è definita da:

   - Il contenuto del nastro di lavoro: $O(S(n))$ simboli. Se $|\Gamma|$ è la cardinalità dell'alfabeto del nastro di lavoro, ci sono $|\Gamma|^{O(S(n))}$ possibili contenuti del nastro di lavoro.
   - La posizione della testina sul nastro di input: $O(\log n)$ bit (dove n è la lunghezza dell'input).

- La posizione della testina sul nastro di lavoro: $O(\log S(n))$ bit (poiché il nastro di lavoro ha lunghezza $O(S(n))$).
- Lo stato corrente della macchina: una costante c di stati ($O(1)$ bit).

Il numero totale di possibili configurazioni N_{conf} è il prodotto delle possibilità per ciascun componente. Poiché $S(n) \geq \Omega(\log n)$, il termine dominante è il contenuto del nastro di lavoro. Dunque, il numero totale di configurazioni (nodi del computation graph) N_{conf} è $O(|\Gamma|^{S(n)} \cdot n \cdot S(n) \cdot c) = O(|\Gamma|^{S(n)} \cdot n \cdot S(n))$. Se consideriamo solo il termine dominante, $N_{conf} = O(|\Gamma|^{S(n)})$.

4. **Applicazione di** `exists-path`**:** Per decidere se l'input x è in L, dobbiamo determinare se esiste un cammino nel computation graph dalla configurazione iniziale C_{start} (con input x) a una delle configurazioni accettanti C_{accept} (possiamo far confluire tutte le configurazioni accettanti in un unico stato finale virtuale). Questo è esattamente il problema **REACHABILITY**.

   Utilizziamo l'algoritmo `exists-path` visto in precedenza. Lo spazio richiesto da `exists-path` per un grafo con N' nodi è $O(\log^2 N')$. Nel nostro caso, N' è il numero di configurazioni N_{conf}. Quindi, lo spazio richiesto è $O(\log^2 N_{conf})$. Sostituendo $N_{conf} = O(|\Gamma|^{S(n)} \cdot n \cdot S(n))$, abbiamo:

$$\begin{aligned} \text{Spazio} &= O(\log^2(|\Gamma|^{S(n)} \cdot n \cdot S(n))) \\ &= O((\log(|\Gamma|^{S(n)}) + \log n + \log S(n))^2) \\ &= O((S(n) \log |\Gamma| + \log n + \log S(n))^2) \end{aligned}$$

   Poiché $S(n) \geq \Omega(\log n)$, il termine dominante all'interno della parentesi è $S(n) \log |\Gamma|$. Il fattore $\log |\Gamma|$ è una costante (dipende dall'alfabeto della TM, non dalla dimensione dell'input). Quindi, lo spazio totale richiesto è $O((S(n))^2) = O(S(n)^2)$.

Questo dimostra che qualsiasi linguaggio decidibile in spazio non deterministico $S(n)$ può essere deciso in spazio deterministico $S(n)^2$.

4 Classi di Complessità Polinomiale Spaziale

Basandosi sul Teorema di Savitch, possiamo definire le classi di complessità polinomiale in spazio.

Definizione 1 (Classe **PSPACE**). *La classe **PSPACE** (Polynomial Space) è l'insieme di tutti i linguaggi che possono essere decisi da una macchina di Turing deterministica che utilizza una quantità di spazio polinomiale rispetto alla dimensione dell'input.*

$$\mathbf{PSPACE} = \bigcup_{c \geq 1} \mathbf{DSPACE}(n^c)$$

Definizione 2 (Classe **NPSPACE**). *La classe **NPSPACE** (Non-deterministic Polynomial Space) è l'insieme di tutti i linguaggi che possono essere decisi da una macchina di Turing non deterministica che utilizza una quantità di spazio polinomiale rispetto alla dimensione dell'input.*

$$\mathbf{NPSPACE} = \bigcup_{c \geq 1} \mathbf{NSPACE}(n^c)$$

4.1 Relazione tra PSPACE e NPSPACE

Dalle definizioni è immediato che **PSPACE** $\subseteq$ **NPSPACE**. Tuttavia, applicando il Teorema di Savitch a funzioni di spazio polinomiali ($S(n) = n^c$): Se un linguaggio $L \in$ **NSPACE**(n^c), allora per il Teorema di Savitch $L \in$ **DSPACE**$((n^c)^2) =$ **DSPACE**(n^{2c}). Poiché $2c$ è ancora una costante, n^{2c} è ancora un polinomio. Quindi, **NPSPACE** $\subseteq$ **PSPACE**.

Questo porta alla conclusione sorprendente:

$$\textbf{PSPACE} = \textbf{NPSPACE}$$

Questo risultato è molto potente. Implica che la non-determinismo non aggiunge potere computazionale significativo quando si considera lo spazio polinomiale (e oltre). La ragione intuitiva è che, a differenza del tempo (che una volta speso è "andato"), lo spazio può essere riutilizzato. Una macchina deterministica può provare tutte le scelte possibili di una macchina non deterministica riutilizzando lo stesso spazio, pur impiegando un tempo esponenziale.

5 Relazioni tra le Classi di Complessità

Possiamo ora visualizzare le relazioni di inclusione tra alcune delle classi di complessità che abbiamo studiato:

- **L**: Linguaggi decidibili in spazio logaritmico deterministico.
- **NL**: Linguaggi decidibili in spazio logaritmico non deterministico. Si congettura che **L** $\neq$ **NL**, anche se è noto che **L** $\subseteq$ **NL**. Inoltre, per il Teorema di Savitch, **NL** $\subseteq$ **DSPACE**$((\log n)^2)$.
- **P**: Linguaggi decidibili in tempo polinomiale deterministico.
- **NP**: Linguaggi decidibili in tempo polinomiale non deterministico.
- **co-NP**: Linguaggi i cui complementi sono in **NP**. Si congettura che **P** $\neq$ **NP**.
- **PSPACE**: Linguaggi decidibili in spazio polinomiale deterministico.
- **NPSPACE**: Linguaggi decidibili in spazio polinomiale non deterministico. Sappiamo che **PSPACE** $=$ **NPSPACE**.
- **EXP**: Linguaggi decidibili in tempo esponenziale deterministico.
- **NEXP**: Linguaggi decidibili in tempo esponenziale non deterministico.

Le relazioni di inclusione note sono le seguenti:

$$\textbf{L} \subseteq \textbf{NL} \subseteq \textbf{P} \subseteq \textbf{NP} \subseteq \textbf{PSPACE} = \textbf{NPSPACE} \subseteq \textbf{EXP} \subseteq \textbf{NEXP}$$

È noto che tutte le inclusioni sono strette, tranne possibilmente **P** $\subseteq$ **NP** e **L** $\subseteq$ **NL**. In particolare, sappiamo che **PSPACE** è strettamente più grande di **P** e **NL** è strettamente più grande di **L** (quest'ultima è una congettura ampiamente accettata, implicata dal fatto che si ritiene **REACHABILITY** non essere in **L**).

Questo conclude la discussione sulle classi di complessità spaziale.

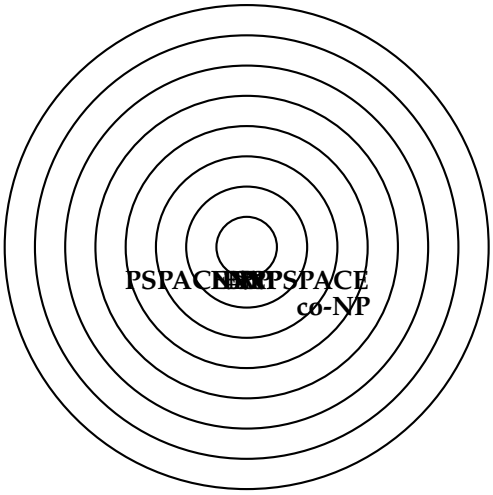


Figura 1: Diagramma di inclusione delle principali classi di complessità.