<u>Appello 23-08</u>

1) Esercizio 1

$L = \{1^m 0^n \mid 5m \leq 3n\}$

Parola → $w = xyz \rightarrow w = 1^p 0^q$, $p = 5m, q = 3n$

$w = xy^i z = 1^p 0^q 0^{k-q}$

La parola ∈ L ma hai un num. diverso di 0 rispetto al numero di 1 (es. $5m > 3n$) e il linguaggio non è regolare.

(Alternativamente – in modo completo)

Supponiamo per assurdo che L sia regolare.

Sia p la costante di pumping garantita dal lemma.

Consideriamo la stringa s = 1^(5p)0^(3p) ∈ L, poiché 5(5p) ≤ 3(3p).

|s| = 8p ≥ p, quindi possiamo applicare il pumping lemma.

Sia s = xyz, con |xy| ≤ p, |y| ≥ 1 e xy^i z ∈ L per ogni i≥0.

Poiché |xy| ≤ p, y può contenere solo 1.

Sia y = 1^k, con k≥1. Consideriamo i=0.

Allora xy^0 z = x1^0 z = xz = 1^(5p-k)0^(3p)

Ma (5p-k) non è divisibile per 5 se k≥1, quindi xz ∉ L.

Ciò è assurdo per il pumping lemma, quindi L non può essere regolare.

2) Esercizio 2

G’ (Forma Normale di Chomsky)

Esempio classico grammatica CF:

S -> aA

A -> aBa

B -> bCb

C -> ϵ

Caratteristiche:

G’ con:

- Stesso stato iniziale
- Produzione di regole
  - A -> a
  - Ogni regola che produce
    una lettera “a” permette la traslitterazione T
  - A -> T(a)
- Stessa regola finale

Se G è CF allora traslitterazione $L(G)$ allora T:

- (→) Ogni regola rispetta la funzione di transizione e genera per ogni $a_i \in L$ ogni regola del tipo $T(a_i)$
- (←) Ogni produzione è in forma $T(a_i)$ derivante da una funzione di transizione δ che mappa ogni a_i simbolo secondo la forma normale di Chomsky → $w = T(a_i) \ldots T(a_n) \in L$.

Sia L un linguaggio context-free su Σ.

Esiste quindi una grammatica context-free G tale che L = L(G).

Costruiamo una nuova grammatica G' su Γ come segue:

- G' ha gli stessi non-terminali di G
- Per ogni produzione A → α in G, G' ha la produzione A → T(α),
- dove T è applicata a ogni simbolo di α
- Gli stessi simboli iniziali

G' genera esattamente le stringhe T(w) per ogni w ∈ L(G).

Infatti, se S =>* w in G, allora S =>* T(w) in G',

e viceversa se S =>* w' in G', w' = T(w) per qualche w tale che S =>* w in G.

Quindi L(G') = T(L(G)) = T(L).

Poiché G' è context-free, anche T(L) lo è.

3) Esercizio (3)

k-PDA: k Pile

La pila fa due operazioni:

- Pop (toglie il simbolo)
- Push (mette il simbolo)

0-PDA = NFA

- Qua non abbiamo pile
- Semplicemente, è l'NFA normale
    - Stato iniziale / Insieme di stati finali

1-PDA = PDA

- Pop/Push

Domanda tipica: Mostrami che 2-PDA è più potente di 1-PDA

(2 Pile > 1 Pila)

2 Pile → TM Cosa Fa?

- Legge
- Scrive
- Va a sx
- Va a dx

Due possibilità per risolvere:

1)

- 1 Pila per leggere
- 1 Pila per scrivere
- Entrambe vanno a dx e a sx
- Risolto = fanno le stesse operazioni della TM

2)

- 1 Pila per gestire stesso nastro a sx
- 1 Pila per gestire stesso nastro a dx
- Entrambe leggono e scrivono
- Risolto = fanno le stesse operazioni della TM

Per mostrare che uno è più potente dell'altro:

1) Dimostriamo che uno legge un linguaggio che l'latro non può
2) Ogni nastro fa una certa cosa:

https://cseweb.ucsd.edu/classes/fa99/cse105/hw3sol.pdf

4) Esercizio 4

1)

Dato ⟨n,W⟩ e un certificato C, possiamo verificare in tempo polinomiale che:

1. For i=1 ad N verificando che esistano tutti gli elettori
2. Esiste l'elemento pivot → W[n]
    a. Perché sicuramente esistevano "i" da 0 ad "n"
3. Disuguaglianze verificate

2)

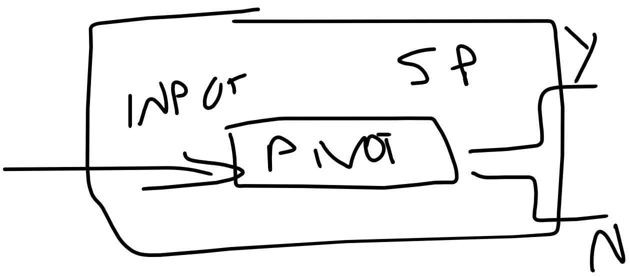
Set Partition = Due sottoinsiemi S1 ed S2 tali che:

- La somma degli elementi in S1 è uguale alla somma degli elementi in S2
- Entrambi danno lo stesso risultato
- = non importa quanta roba c'è dentro, ottengo sempre lo stesso valore

$$SetPartition \to SP$$

$$A \leq_m B$$

$$SP \leq_m PIVOT$$



Dato il problema:

$$4\ 3\ 3\ 2\ 1$$

$$S_1 = 4 + 2 = 6$$

$$S_2 = 3 + 3 = 6$$

$$SetPartition\ dice\ S_1 = S_2!$$

$$PIVOT\ invalida\ questa\ cosa!$$

Se aggiungo 1 ad S_1 ed S_2 fallisce la condizione di SP.

(Per esempio):

Dato un'istanza ⟨S⟩ di SET-PARTITION, con S = {x_1, ..., x_m}, costruiamo:

- n = m+1

- W[i] = x_i per i = 1, ..., m

- W[n] = (1/2) Σ_{x∈S} x

Allora:

- Se ⟨S⟩ ∈ SET-PARTITION, esiste S_1 ⊆ S tale che

  Σ_{x∈S_1} x = (1/2) Σ_{x∈S} x = W[n].

  Quindi C = {i | x_i ∈ S_1} dimostra che m+1 è un pivot.

- Se ⟨n,W⟩ ∈ PIVOT, esiste C ⊆ {1, ..., m} che soddisfa le disuguaglianze.

  Ponendo S_1 = {x_i | i ∈ C}, le disuguaglianze implicano che

  Σ_{x∈S_1} x = (1/2) Σ_{x∈S} x, quindi ⟨S⟩ ∈ SET-PARTITION

$$SP \leq_m PIVOT$$

Data un'istanza del problema, usando un certo numero di elettori:

- Presi due sottoinsiemi (due pezzi a caso) di elettori, entrambi daranno esattamente quello che mi dice il problema:

$$\sum_{j \in \frac{C}{2}} W[j] + \sum_{j \in \frac{C}{2}} W[j] < \frac{1}{2} \sum_{j=1}^{n} W[j]$$

Aggiungendo l'elemento pivot $W[n]$, tutto somma allo stesso numero x, rispettando le condizioni del problema (i due sottoinsiemi hanno stessa somma)

(Adesso abbiamo fatto: usando SET PARTITIONING, risolvi PIVOT)

- Ora il contrario (avendo PIVOT, siamo in SP)

Avendo PIVOT, abbiamo tutti gli elettori.

Questo vuol dire che, aggiungendo W[n], allora entrambi i sottoinsiemi valutano ad x.

$S_1 + S_2 = x$ grazie a $W[n]$

Problema risolto! (NP-Hard)

Appello 05/02/2024

1. Esercizio 3

$\exists E$ (enumeratore) ⇔ Ordinamento standard

Decidibile → Termina usando un oggetto finito

(→)

Usando un enumeratore, seguiamo l'ordinamento.

Il nostro E avrà a disposizione un alfabeto $\Sigma = \{a \dots z\}$ e stampa tutti i simboli nell'ordine previsto (esempio del ciclo for → w: $\{1..n\}$, stampiamo tutti i caratteri seguendo l'ordine previsto dalla funzione di transizione. Se l'enumeratore termina, allora significa che è stato correttamente seguito l'ordine previsto dall'alfabeto.

(←)

Avendo l'ordinamento standard, esiste un enumeratore. Questo succede perché la funzione di transizione dell'enumeratore prende in input tutte le stringhe ordinate e, se non fossero ordinate, non riesce a stamparle. Correttamente lui stampa solo le stringhe come gli arrivano nel modo ordinato.

Risolto!

2. Esercizio 4

   a.

$MUL_{TM} = \{\langle M \rangle \mid M$ è una TM che, dati in input due numeri binari x ed y, ottiene $x * y\}$

   b.

$$A_{TM} \leq_m MUL_{TM}$$

F funzione di riduzione, prendendo in input M la TM e w input:

$F = \langle M, w \rangle$ su input w:

- Simula M sull'input w
- Verifica l'esistenza di x ed y
- Se la TM riesce ad ottenere il prodotto binario, lo scrive sul nastro ed esegue M'
  - M' simula la propria esecuzione, avendo input x
  - Se riesce a scrivere x sul nastro, si ferma accettando, altrimenti rifiuta
- Restituisci $\langle M \rangle$

⇒ $\langle M, w \rangle \in MUL_{TM}$, allora la macchina A_{TM} si ferma accettando dato che ha svolto il prodotto tra i numeri binari previsti

⇒ $\langle M, w \rangle \notin MUL_{TM}$, allora la macchina A_{TM} rifiuta dato che i simboli x ed y non hanno permesso di ottenere in output il prodotto in formato binario dei due numeri previsti