- Sia M una TM deterministica che si ferma su tutti gli input.
- Il tempo di esecuzione (o complessità di tempo) di M è la funzione $f : \mathbb{N} \mapsto \mathbb{N}$ tale che $f(n)$ è il numero massimo di passi che M utilizza su un input di lunghezza n.
- Se $f(n)$ è il tempo di esecuzione di M, diciamo che M è una TM di tempo $f(n)$.
- Useremo n per rappresentare la lunghezza dell'input.
- Ci interesseremo dell'analisi del caso pessimo.

Let f and g be functions $f, g\colon \mathcal{N} \longrightarrow \mathcal{R}^+$. Say that $\boldsymbol{f(n) = O(g(n))}$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$,

$$f(n) \leq c\, g(n).$$

When $f(n) = O(g(n))$, we say that $g(n)$ is an ***upper bound*** for $f(n)$, or more precisely, that $g(n)$ is an ***asymptotic upper bound*** for $f(n)$, to emphasize that we are suppressing constant factors.

Let $t\colon \mathcal{N} \longrightarrow \mathcal{R}^+$ be a function. Define the ***time complexity class***, $\textbf{TIME}(\boldsymbol{t(n)})$, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

It shows that $A \in \mathrm{TIME}(n \log n)$.

M_2 = "On input string w:
1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat as long as some 0s and some 1s remain on the tape:
3. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, *reject*.
4. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
5. If no 0s and no 1s remain on the tape, *accept*. Otherwise, *reject*."

We can decide the language A in $O(n)$ time (also called ***linear time***) if the Turing machine has a second tape. The following two-tape TM M_3 decides A in linear time. Machine M_3 operates differently from the previous machines for A. It simply copies the 0s to its second tape and then matches them against the 1s.

M_3 = "On input string w:
1. Scan across tape 1 and *reject* if a 0 is found to the right of a 1.
2. Scan across the 0s on tape 1 until the first 1. At the same time, copy the 0s onto tape 2.
3. Scan across the 1s on tape 1 until the end of the input. For each 1 read on tape 1, cross off a 0 on tape 2. If all 0s are crossed off before all the 1s are read, *reject*.
4. If all the 0s have now been crossed off, *accept*. If any 0s remain, *reject*."

THEOREM 7.8

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multitape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine.

Let N be a nondeterministic Turing machine that is a decider. The ***running time*** of N is the function $f\colon \mathcal{N} \longrightarrow \mathcal{N}$, where $f(n)$ is the maximum number of steps that N uses on any branch of its computation on any input of length n, as shown in the following figure.

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,

$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k).$$

The class P plays a central role in our theory and is important because
1. P is invariant for all models of computation that are polynomially equivalent to the deterministic single-tape Turing machine, and
2. P roughly corresponds to the class of problems that are realistically solvable on a computer.

Per dimostrare che un problema/algoritmo è in P:

- Descrivi l'algoritmo per fasi numerate
- Dai un limite superiore polinomiale al numero di fasi che l'algoritmo esegue per un input di lunghezza n
- Assicurati che ogni fase possa essere completata in tempo polinomiale su un modello di calcolo deterministico ragionevole
- L'input deve essere codificato in modo ragionevole

Raggiungibilità in un grafo

$PATH = \{\langle G, s, t\rangle \mid G$ grafo che contiene un cammino da s a $t\}$

Numeri relativamente primi

$RELPRIME = \{\langle x, y\rangle \mid 1$ è il massimo comun divisore di x e $y\}$

(a

4. A **triangle** in an undirected graph is a 3-clique. Show that $\mathit{TRIANGLE} \in \mathrm{P}$, where $\mathit{TRIANGLE} = \{\langle G \rangle \mid G \text{ contains a triangle}\}$.

   Answer: Let $G = (V, E)$ be a graph with a set V of vertices and a set E of edges. We enumerate all triples (u, v, w) with vertices $u, v, w \in V$ and $u < v < w$, and then check whether or not all three edges (u, v), (v, w) and (u, w) exist in E. Enumeration of all triples requires $O(|V|^3)$ time. Checking whether or not all three edges belong to E takes $O(|E|)$ time. Thus, the overall time is $O(|V|^3\,|E|)$, which is polynomial in the length of the input $\langle G \rangle$. Therefore, $\mathit{TRIANGLE} \in \mathrm{P}$.

Let $ALL_{\mathrm{DFA}} = \{\langle \mathrm{A} \rangle \mid \mathrm{A}$ is a DFA that recognizes $\Sigma^*\}$

7. **Exercise 7.10** Show that ALL_{DFA} is in P.

   Solution: ALL_{DFA} is decided by the following deterministic Turing machine.

   $T =$ "On input $\langle M \rangle$ where M is a DFA

   1. Mark the start state of M.
   2. Repeat until no new states are marked.
   3. Mark any state that has a transition coming into it from a marked state.
   4. If every marked state is an accept state, *accept*.

   Steps 1 and 4 are done once. Each time through the loop in Steps 2 and 3 except the last time, an unmarked state is marked, so the loop is executed no more than m times, where m is the number of states in M. Thus, the total number of steps executed is no more than $2m + 2$ and this is polynomial in the size of $\langle M \rangle$.

   Step 1 involves finding the start state in the list of states and marking it. Step 3 involves processing each transition once and for each transition checking if it goes from a marked state to an unmarked state. Step 4 involves checking if each marked state is in the list of accept states. Each of these steps can be implemented in time polynomial in the size of $\langle M \rangle$, so the algorithm runs in polynomial time.

4. Mostra che A_{TM} non è riducibile mediante funzione a E_{TM}. In altre parole, dimostra che non esiste una funzione calcolabile che riduce A_{TM} ad E_{TM}.

5.5 Suppose for a contradiction that $A_{\mathsf{TM}} \leq_{\mathrm{m}} E_{\mathsf{TM}}$ via reduction f. It follows from the definition of mapping reducibility that $\overline{A_{\mathsf{TM}}} \leq_{\mathrm{m}} \overline{E_{\mathsf{TM}}}$ via the same reduction function f. However, $\overline{E_{\mathsf{TM}}}$ is Turing-recognizable (see the solution to Exercise 4.5) and $\overline{A_{\mathsf{TM}}}$ is not Turing-recognizable, contradicting Theorem 5.28.

5. Mostra che se A è Turing-riconoscibile e $A \leq_m \overline{A}$, allora A è decidibile.

We know that $A \leq_m \overline{A}$ for some language A. By definition, this means that there exits a computable function f such that for all w, $w \in A$ if and only if $f(w) \in \overline{A}$. We can rewrite the latter as "$w \in \overline{A}$ if and only if $f(w) \in A$". Therefore, $\overline{A} \leq_m A$. Since A is Turing-recognizable, by Theorem 5.22 (page 193 in the textbook), $\overline{A}$ is also Turing-recognizable. By Theorem 4.16 (page 167 in the textbook), A is decidable since is Turing-recognizable and co-Turing-recognizable. (Recall that A is co-Turing-recognizable if $\overline{A}$ is Turing-recognizable).

3. (8 punti) *Una* Turing machine con alfabeto binario *è una macchina di Turing deterministica a singolo nastro dove l'alfabeto di input è* $\Sigma = \{0, 1\}$ *e l'alfabeto del nastro è* $\Gamma = \{0, 1, \sqcup\}$. *Questo significa che la macchina può scrivere sul nastro solo i simboli* $0, 1$ *e blank: non può usare altri simboli né marcare i simboli sul nastro.*

Dimostra che che le Turing machine con alfabeto binario machine riconoscono tutti e soli i linguaggi Turing-riconoscibili sull'alfabeto $\{0, 1\}$.

Per risolvere l'esercizio dobbiamo dimostrare che *(a)* ogni linguaggio riconosciuto da una Turing machine con alfabeto binario è Turing-riconoscibile e *(b)* ogni linguaggio Turing-riconoscibile sull'alfabeto $\{0, 1\}$ è riconosciuto da una Turing machine con alfabeto binario.

(a) Questo caso è semplice: una Turing machine con alfabeto binario è un caso speciale di Turing machine deterministica a nastro singolo. Quindi ogni linguaggio riconosciuto da una Turing machine con alfabeto binario è anche Turing-riconoscibile.

(b) Per dimostrare questo caso, consideriamo un linguaggio L Turing-riconoscibile, e sia M una Turing machine deterministica a nastro singolo che lo riconosce. Questa TM potrebbe avere un alfabeto del nastro Γ che contiene altri simboli oltre a $0, 1$ e blank. Per esempio potrebbe contenere simboli marcati o separatori.
Per costruire una TM con alfabeto binario B che simula il comportamento di M dobbiamo come prima cosa stabilire una *codifica binaria* dei simboli nell'alfabeto del nastro Γ di M. Questa codifica è una funzione C che assegna ad ogni simbolo $a \in \Gamma$ una sequenza di k cifre binarie, dove k è un valore scelto in modo tale che ad ogni simbolo corrisponda una codifica diversa. Per esempio, se Γ contiene 4 simboli, allora $k = 2$, perché con 2 bit si rappresentano 4 valori diversi. Se Γ contiene 8 simboli, allora $k = 3$, e così via.

La TM con alfabeto binario B che simula M è definita in questo modo:

$B =$ "su input w:

1. Sostituisce $w = w_1 w_2 \dots w_n$ con la codifica binaria $C(w_1)C(w_2)\dots C(w_n)$, e riporta la testina sul primo simbolo di $C(w_1)$.
2. Scorre il nastro verso destra per leggere k cifre binarie: in questo modo la macchina stabilisce qual è il simbolo a presente sul nastro di M. Va a sinistra di k celle.
3. Aggiorna il nastro in accordo con la funzione di transizione di M:
   - Se $\delta(r, a) = (s, b, R)$, scrive la codifica binaria di b sul nastro.
   - Se $\delta(r, a) = (s, b, L)$, scrive la codifica binaria di b sul nastro e sposta la testina a sinistra di $2k$ celle.
4. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di M, allora *accetta*; se la simulazione raggiunge lo stato di rifiuto di M allora *rifiuta*; altrimenti prosegue con la simulazione dal punto 2."

3. **(8 punti)** Una *Turing machine con alfabeto ternario* è una macchina di Turing deterministica a singolo nastro dove l'alfabeto di input è $\Sigma = \{0, 1, 2\}$ e l'alfabeto del nastro è $\Gamma = \{0, 1, 2, \sqcup\}$. Questo significa che la macchina può scrivere sul nastro solo i simboli 0, 1 e blank: non può usare altri simboli né marcare i simboli sul nastro.

Dimostra che ogni linguaggio Turing-riconoscibile sull'alfabeto $\{0, 1, 2\}$ può essere riconosciuto da una Turing machine con alfabeto ternario.

Per dimostrare che ogni linguaggio Turing-riconoscibile sull'alfabeto {0, 1, 2} può essere riconosciuto da una Turing machine con alfabeto ternario, possiamo procedere come segue:

Sia L un linguaggio Turing-riconoscibile sull'alfabeto {0, 1, 2}. Esiste una Turing machine M che riconosce L.

Costruiamo una Turing machine M' con alfabeto ternario Σ = {0, 1, _} (dove "_" rappresenta il blank) che riconosce L nel seguente modo:

1. M' simula M sul nastro, con la seguente codifica:

   - 0 è codificato come 0

   - 1 è codificato come 01

   - 2 è codificato come 011

2. Quando M legge un simbolo a dal suo nastro, M' legge la sua codifica dal proprio nastro. In particolare:

   - Se a = 0, M' legge 0

   - Se a = 1, M' legge 01

   - Se a = 2, M' legge 011

3. Quando M scrive un simbolo a sul suo nastro, M' scrive la sua codifica sul proprio nastro, sovrascrivendo tutti i simboli necessari.

4. M' simula le transizioni di stato e i movimenti della testina di M fedelmente.

5. Se M entra in uno stato di accettazione, anche M' entra in uno stato di accettazione. Se M entra in uno stato di rifiuto o non si ferma, lo stesso fa M'.

In questo modo, M' riconosce esattamente lo stesso linguaggio L di M, utilizzando solo l'alfabeto ternario {0, 1, _}.

Infatti, per ogni stringa w in {0, 1, 2}*, M accetta w se e solo se M' accetta la codifica di w in {0, 1, _}*.

Poiché L era un linguaggio Turing-riconoscibile arbitrario sull'alfabeto {0, 1, 2}, abbiamo dimostrato che ogni tale linguaggio può essere riconosciuto da una Turing machine con alfabeto ternario {0, 1, _}.

In conclusione, abbiamo mostrato che le Turing machine con alfabeto ternario sono sufficienti per riconoscere tutti i linguaggi Turing-riconoscibili sull'alfabeto {0, 1, 2}, e quindi hanno la stessa potenza computazionale.

2. (9 punti) Chiamiamo k-PDA un automa a pila dotato di k pile. In particolare, uno 0-PDA è un NFA e un 1-PDA è un PDA convenzionale. Sappiamo già che gli 1-PDA sono più potenti degli 0-PDA (nel senso che riconoscono una classe più ampia di linguaggi. Mostra che i 2-PDA sono più potenti degli 1-PDA.

1. Definire un linguaggio L che possa essere decisa da un 2-PDA.

2. Dimostrare che L non può essere deciso da alcun 1-PDA dimostrando che L non è libero dal contesto.

3. Dimostrare che L può essere deciso da una macchina di Turing.

Prendiamo come esempio la lingua $L = \{a^n b^n c^n \mid n >= 0\}$.

Passo 1: Dimostrare che L può essere deciso da un 2-PDA.

Possiamo costruire un 2-PDA che decide L come segue:

- Leggi le 'a' e spingile sulla prima pila.

- Leggi le 'b' e abbinale alle 'a' della prima pila. Spingi 'c' sulla seconda pila.

- Leggi le 'c' e abbinale alle 'c' della seconda pila.

- Accetta se tutti i simboli sono abbinati ed entrambe le pile sono vuote.

Passo 2: Dimostrare che L non può essere deciso da nessun 1-PDA.

Possiamo usare il lemma di pompaggio per i linguaggi context-free per mostrare che L non è context-free.

Si supponga che L sia privo di contesto. Sia p la lunghezza di pompaggio data dal lemma di pompaggio.

Si consideri la stringa $s = a^p b^p c^p$ in L. Con il lemma di pompaggio, s può essere diviso in cinque parti, s = uvxyz, soddisfacendo le condizioni del lemma.

Tuttavia, il pompaggio di v e y risulterà in una stringa con un numero disuguale di 'a', 'b' e 'c', che non è in L. Questo contraddice il lemma di pompaggio.

Pertanto, L non è libero dal contesto e quindi non può essere deciso da alcun PDA 1.

Passo 3: Dimostrare che L può essere deciso da una macchina di Turing.

Possiamo costruire una macchina di Turing M che decide L come segue:

- Scansiona l'input e cancella una 'a', una 'b' e una 'c' alla volta fino a quando non è più possibile cancellare i simboli.

- Se l'input è esaurito e tutti i simboli sono stati cancellati, accettare. In caso contrario, rifiutare.

M decide L perché accetta tutte le stringhe in L e rifiuta tutte le stringhe non in L.

In conclusione, abbiamo dimostrato che il linguaggio $L = \{a^n b^n c^n \mid n >= 0\}$ può essere deciso da un 2-PDA e da una macchina di Turing, ma non può essere deciso da nessun 1-PDA. Ciò dimostra che i 2-PDA sono più potenti degli 1-PDA in termini di classe di linguaggi che possono decidere, e che le macchine di Turing sono almeno altrettanto potenti dei 2-PDA.

2. Dimostra che il seguente linguaggio è indecidibile:

$$L_2 = \{\langle M, w \rangle \mid M \text{ accetta la stringa } ww^R\}.$$

Per dimostrare che il linguaggio L2 = { ⟨M, w⟩ | M accetta la stringa ww^R} è indecidibile, possiamo procedere come segue:

Possiamo ridurre il problema della fermata (Halting problem) a L2. Il problema della fermata è noto per essere indecidibile.

Costruiamo una riduzione f che, dato un input ⟨M, w⟩ per il problema della fermata, genera una nuova macchina di Turing M' che funziona nel seguente modo:

- M' simula M sull'input w.

- Se M si ferma sull'input w, M' accetta la stringa ww^R (w concatenato con il suo reverse).

- Se M non si ferma su w, M' entra in un loop infinito su qualsiasi input.

Quindi, f restituisce ⟨M', ww^R⟩ .

Se ⟨M, w⟩ è un'istanza positiva del problema dell'arresto (cioè M si ferma su w), allora ⟨M', ww^R⟩ è in L2 perché M' accetta ww^R.

Se ⟨M, w⟩ è un'istanza negativa del problema della fermata (cioè M non si ferma su w), allora ⟨M', ww^R⟩ non è in L2 perché M' non accetta alcun input.

Quindi, f è una riduzione dal problema della fermata a L2. Poiché il problema della fermata è indecidibile e riducibile a L2, anche L2 deve essere indecidibile.

In conclusione, abbiamo dimostrato che L2 è indecidibile riducendo il problema indecidibile della fermata ad esso. Ciò mostra che non può esistere un algoritmo generale per determinare se una macchina di Turing accetta una stringa nella forma ww^R.