Introduzione alla calcolabilità

Indice

Da *Time* del 29 marzo 1999: *"Nei moderni calcolatori confluiscono tante idee e tanti progressi tecnologici che sarebbe temerario assegnare a una sola persona il merito di averli inventati. Ma rimane il fatto che chiunque batta su una tastiera aprendo una tabella a doppia entrata, o un programma di videoscrittura, lavora su una incarnazione della macchina di Turing"*.

Obiettivo di questo corso è lo studio dei fondamenti teorici che hanno portato alla costruzione dei moderni calcolatori e che, nel contempo, hanno reso possibile la definizione di metodologie di analisi dei problemi rispetto alla loro possibilità (o impossibilità) ad essere risolti *automaticamente* ed in maniera *efficiente*. Ma andiamo con ordine.

Come mai nel Medio Evo non si sono osservati sostanziali progressi nel campo della Matematica?
Provate a risolvere il calcolo seguente:

$$\text{MMMCDXLVII} \times \text{MMDCCXXXIV}$$

Come a dire: *è tutta questione di linguaggio*. E tutto cominciò molto tempo fa.

1.1 La logica di Aristotele

La Logica Aristotelica è raccolta in una serie di scritti dal titolo complessivo di *Organon*, ossia, in greco "Strumento". Comunque, i termini "logica" e "organon" sono posteriori. Il termine che Aristotele (383 a.c.) utilizza per designare l'oggetto dei suoi studi è *analitica*, per indicare che si tratta di un *metodo di risoluzione del ragionamento ai suoi elementi costitutivi*.

Organon è il nome dato da Andronico di Rodi (I sec. a.C.), seguace di Aristotele tra i Peripatetici, all'edizione standard delle sue sei opere di logica. Secondo altre fonti, il termine Organon viene utilizzato per la prima volta da Alessandro di Afrodisia (I sec. d.C.) e fu poi adottato dal VI d.C. per denominare il complesso degli scritti aristotelici. Comunque sia, il termine "Organon" significa "strumento" e servirebbe a sottolineare la funzione propedeutica o introduttiva della logica come strumento di cui si avvalgono tutte le scienze.

Il termine "logica" è quasi certamente di origine stoica, e possiamo intenderlo come *scienza dei "logoi"*, ossia, *scienza dei discorsi*, riferendosi al suo oggetto di studio: il pensiero espresso nei discorsi.

Lo sviluppo della logica aristotelica va pensato in parallelo allo sviluppo della metafisica. In effetti, Aristotele non inserisce la logica nel quadro delle scienze come scienza a sé stante. Piuttosto, Aristotele la intendeva come studio della struttura della scienza in generale. Infatti, secondo Aristotele esiste un rapporto *necessario* fra le forme del pensiero, studiate dalla logica, e le forme della realtà, studiate dalla metafisica. L'esistenza di questo rapporto è una delle questioni più controverse che hanno occupato i matematici del periodo che ci accingiamo ad analizzare (XIX e XX secolo).

Alla base della logica aristotelica è il concetto di proposizione: una affermazione che può avere uno solo di due valori, *vero* oppure *falso*. Aristotele fondava la logica su tre principi:

- *principio di identità*: ogni proposizione è uguale a sé stessa;
- *principio di non contraddizione*: non possono essere contemporaneamente vere sia una proposizione che la sua negazione;
- *principio del terzo escluso*: fra una proposizione e la sua negazione, almeno una di esse deve essere vera.

Osserviamo che sul principio del terzo escluso è basata la tecnica di *dimostrazione per assurdo*: poiché una affermazione genera una contraddizione allora essa deve essere falsa e, quindi, in virtù del principio del terzo ecluso la sua negazione deve essere vera.
Il principio di non contraddizione è considerato da Aristotele il perno di ogni ragionamento, come è espresso nella sua *Metafisica*: "È impossibile che la stessa qualità appartenga e non appartenga alla stessa cosa ... Questo è il più certo di tutti i principi ... Per questa ragione tutti coloro che eseguono qualche dimostrazione si riferiscono ad esso come a una conoscenza fondamentale. Esso è infatti, per natura, la fonte di tutti gli altri assiomi"[1].

[1] Come riportato nel testo di Davies (Nota a fine dispensa), questo passo viene citato in G. Boole, *An investigation of the Laws of Thought on which are Founded the Mathematical Theories of Logic and Probabilities*, 1854.

Aristotele introdusse anche il ragionamento per *sillogismo*. Il sillogismo (dal greco syllogismòs, formato da syn, "insieme", e logismòs, "calcolo": quindi, *ragionamento concatenato*) è un tipo di ragionamento dimostrativo che, partendo dai tre tipi di termine giunge ad una *conclusione* collegando i suddetti termini attraverso brevi enunciati (*premesse*). Il sillogisma più noto di Aristotele è il seguente:

Socrate è un uomo	(premessa maggiore)
Tutti gli uomini sono mortali	(premessa minore)
Allora, Socrate è mortale	(conclusione).

I termini di un sillogismo sono detti *maggiore*, che funge da predicato nella conclusione (nell'esempio, *mortale*), *medio* (nell'esempio, *uomo* e *uomini*) e *minore*, che nella conclusione funge da soggetto (nell'esempio, *Socrate*); essi vengono classificati in base al rapporto contenente - contenuto, e sulla base di tale classificazione, dalle premesse si deriva la conclusione.

Le proposizioni che compongono un sillogismo categorico possono essere:

- *universali affermative* ("Tutti gli A sono B"),
- *universali negative* ("Nessun A è B"),
- *particolari affermative* ("Qualche A è B"),
- *particolari negative* ("Qualche A non è B").

Il sillogismo che abbiamo descritto risponde alla regola di inferenza detta del *modus ponens*, che nel linguaggio proposizionale viene espressa come

$$[\,(\,p \,\rightarrow\, q\,) \,\wedge\, p\,] \,\Rightarrow\, q,$$

che va interpretata nel modo seguente: "se so che ogni volta che è vera p anche q è vera, e mi accorgo che, nel mio caso particolare, p è vera, allora posso dedurre che, nel mio caso particolare, q è vera"[2]
Il *modus tollens* è una seconda regola di inferenza della logica proposizionale, sviluppata compiutamente per la prima volta dai logici medievali ma conosciuta già agli stoici. Letteralmente, "modus tollens" possiamo tradurlo come: "il modo che toglie la verità di una proposizione togliendo quella di un'altra". Nel linguaggio proposizionale essa viene espressa come

$$[\,(\,p \,\rightarrow\, q\,) \,\wedge\, \neg q\,] \,\Rightarrow\, \neg p,$$

che va interpretata nel modo seguente: "se so che ogni volta che è vera p anche q è vera, e mi accorgo che, nel mio caso particolare, q è falsa, allora posso dedurre che, nel mio caso particolare, p è falsa".
Osserviamo che, comunque, nella logica proposizionale l'unica regola di inferenza necessaria è il modus ponens.

Il sillogismo non è altro che una *regola di inferenza*. Nella logica matematica, una regola di inferenza è l'atto di trarre una conclusione basandosi sulla forma delle premesse. Nel caso una regola di inferenza sia corretta allora stabilisce quando un enunciato formalizzato (cioè, come si dirà molto più tardi, una formula di un linguaggio proposizionale o del primo ordine) è conseguenza logica di un altro *soltanto sulla base della struttura sintattica degli enunciati*. Possiamo, allora, considerare il sillogismo come il primo passo verso l'automatizzazione dei processi deduttivi, ossia, verso ciò che oggi chiamiamo *calcolabilità*.

1.2 Il sogno di Leibniz

Il matematico e filosofo Leibniz (1646-1716), affascinato dal sistema logico elaborato da Aristotele, e dalla suddivisione aristotelica dei concetti in "categorie", concepì fin da giovane quella che avrebbe definito la sua *idea meravigliosa*: creare un alfabeto speciale i cui elementi non stessero per suoni ma per concetti.

Il primo risultato pratico del suo lavoro risale al 1673, quando Leibniz presentò un modello di macchina calcolatrice capace di eseguire le quattro operazioni aritmetiche fondamentali. In effetti, era sua convinzione che "...è indegno

[2] Si osservi l'utilizzo dei due diversi simboli di implicazione $\rightarrow$ (implicazione materiale) e $\Rightarrow$ (implicazione logica).

di uomini eccellenti perdere ore come schiavi nelle fatiche di calcoli che potrebbero essere tranquillamente affidati a chicchessia se si usasse questa macchina".
Nel 1674 descrisse una macchina capace di risolvere equazioni algebriche, e un anno dopo, in un altro scritto paragonò il ragionamento logico ad un meccanismo. Il suo obiettivo era ormai chiaro: *ridurre ogni ragionamento a una sorta di calcolo e costruire, da ultimo, una macchina capace di "calcolare" in questo senso generale*.

L'esplosione della ricerca matematica nel Seicento era stata alimentata da due innovazioni decisive:

1. la manipolazione delle espressioni algebriche (l'algebra delle scuole medie) era stata sistematizzata fino a diventare, in pratica, quella tecnica molto potente che usiamo ancora oggi;
2. rappresentando i punti come coppie di numeri, Descartes e Fermat avevano mostrato che la geometria è riconducibile all'algebra.

Queste nuove possibilità venivano sfuttate dai matematici per risolvere problemi un tempo inaccessibili. Nei calcoli intervenivano processi di limite, nel senso che alla soluzione si perveniva attraverso approssimazioni successive che si avvicinavano sempre più ad essa; l'idea era di non accontentarsi di valori approssimati particolari, ma di *passare al limite* per ottenere il valore esatto.

Negli ultimi mesi del 1675, Leibniz fece importanti passi avanti (sia concettuali che computazionali) nell'uso dei passaggi al limite; questi risultati, nel loro insieme, costituiscono l'invenzione leibniziana del calcolo infinitesimale. Per l'esattezza, Leibniz:

1. riconobbe che il calcolo delle aree e la determinazione dei rapporti incrementali puntuali erano casi paradigmatici, nel senso che molti differenti tipi di problemi erano riconducibili all'uno o all'altro
2. comprese che, in modo molto simile alle operazioni di addizione e sottrazione (o moltiplicazione e divisione), le operazioni matematiche richieste per calcolare le soluzioni dei tipi di problemi menzionati sopra erano l'una l'inversa dell'altra - oggi queste operazioni sono dette rispettivamente integrazione e derivazione, e la proprietà in questione è nota come "teorema fondamentale del calcolo integrale"
3. inventò un simbolismo molto efficace (usato ancora oggi), $\int$ per l'integrazione e d per la derivazione, e, infine, scoprì le regole matematiche per eseguire concretamente integrazioni e derivazioni che si presentavano nella pratica.

Nel loro insieme, queste scoperte trasformarono i passaggi al limite, sino ad allora un metodo esoterico e accessibile a pochi iniziati, in una tecnica semplice e chiara, che potè essere insegnata a generazioni di studenti.
Ma la cosa più importante è che Leibniz si convinse che *era fondamentale scegliere simboli adatti e trovare regole che ne governassero la manipolazione*. Diversamente dalle lettere di un alfabeto fonetico, i simboli $\int$ e d non erano suoni privi di significato ma rappresentavano concetti, e fornivano quindi un modello di quell'idea meravigliosa, affacciatasi alla sua mente fin da ragazzo, di *un alfabeto che rappresentasse tutti i concetti fondamentali*.

Per Leibniz era chiaro che sia i segni speciali usati in aritmetica e algebra, sia i simboli usati in astronomia o in chimica, sia quelli introdotti da egli stesso per il calcolo differenziale e integrale, rappresentavano altrettanti paradigmi che mostravano tutta l'importanza di un simbolismo ben scelto. Così, ad esempio, l'algebra mostra come un sistema di simboli ben scelti sia utile e anzi indispensabile per il pensiero deduttivo.
Nella terminologia di Leibniz un sistema simbolico con queste proprietà era chiamato *caratteristica*; ciascuno degli esempi citati era una *caratteristica reale*, in cui ogni singolo simbolo rappresentava un'idea ben definita, in modo naturale ed adeguato (a differenza dei siboli dell'alfabeto che hanno solo valore fonetico). Ma quello di cui c'era bisogno, secondo Leibniz, era una *caratteristica universale*, cioè, un sistema di simboli che non solo fosse "reale" ma abbracciasse anche il pensiero umano in tutta la sua estensione.

Leibniz divideva il suo programma in tre parti:

1. creare un compendio che abbracciasse l'intera conoscenza umana - una sorta di enciclopedia dalla quale
2. scegliere le nozioni fondamentali e cruciali e individuare per ciascuna di esse i simboli adeguati

3. ridurre le regole deduttive a manipolazioni di questi simboli, ovvero a quello che Leibniz chiamava *calculus ratiocinator* e che oggi potremmo chiamare (in una sua forma più ristretta) logica simbolica - una vera e propria algebra della logica.

Una volta espresso un concetto in tale linguaggio, per analizzarne le proprietà sarebbe stato sufficiente dire: *calcolemus!*

1.3 Ma in pratica cosa significa *calcolemus*?

Consideriamo un semplice problema di geometria euclidea di base.

Problema. Si consideri un trapezio isoscele $ABCD$ avente la diagonale perpendicolare al lato obliquo, la cui base minore AB misuri 7 cm e la cui base maggiore CD misuri 12 cm. Calcolare il perimetro e l'area del trapezio.

Soluzione 1. Supponiamo che il problema sia stato proposto a studenti di terza media. Allora, tali studenti stanno imparando i primi rudimenti dell'algebra - forse conoscono il calcolo letterale limitato ai monomi, e forse sanno impostare e risolvere una equazione di primo grado. Certamente, hanno studiato, come parte della geometria euclidea, tutte le proprietà dei poligoni iscritti e circoscritti ad una circonferenza e certamente conoscono il teorema di Pitagora. Allora, dovranno osservare che, in riferimento alla Figura 1.1, poiché l'angolo DBC è retto allora il triangolo DBC è inscritto in una semicirconferenza e, analogamente, anche il triangolo DAC è inscritto in una semicirconferenza. D'altra parte, osserveranno gli studenti, le circonferenze che circoscrivono, rispettivamente, DBC e DAC insistono sullo stesso diametro, quindi esse sono coincidenti. In definitiva, il trapezio è inscritto in una semicircoferenza.
Allora, la base minore AB è una corda (parallela al diametro DC) e quindi il raggio OP ad essa perpendicolare la divide in due parti uguali, ossia $\overline{OM}$ è l'altezza del trapezio e $\overline{MB} = \frac{\overline{AB}}{2}$. Inoltre, poiché OC e OB sono due raggi della circonferenza di diametro DC, allora $\overline{OB} = \overline{OC} = \frac{\overline{DC}}{2}$ e i nostri studenti potranno (finalmente!) calcolare la lunghezza dell'altezza del trapezio applicando il teorema di Pitagora al triangolo OMB.
Una volta calcolata l'altezza, osserveranno che $\overline{BH} = \overline{OM}$ e che $\overline{HC} = \frac{\overline{DC}-\overline{AB}}{2}$ e, applicando il teorema di Pitagora al triangolo BCH, calcoleranno anche la lunghezza del lato obliquo BC e potranno, infine, rispondere ai quesiti posti dal problema.

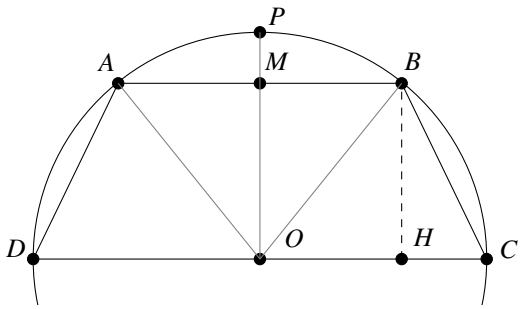

Figura 1.1: La soluzione a livello di scuola media: occorre sapere che, in questo caso, il trapezio è inscritto in una semicirconferenza.

Niente male davvero come richiesta a studenti di terza media! Anche se abbiamo assunto che i nostri studenti disponessero di tutti gli strumenti necessari alla soluzione del problema, possiamo certamente affermare che detta soluzione richiedeva una serie di deduzioni tutt'altro che *automatiche*. Vediamo, ora, come lo stesso problema sarebbe risolto da studenti del secondo o terzo anno delle superiori che conoscono, oltre al teorema di Pitagora, uno strumento più avanzato dell'algebra: i sistemi di equazioni.

Soluzione 2. In riferimento alla Figura 1.2, poiché l'angolo in $C\hat{B}D$ è retto, se indichiamo con H la proiezione del punto B sul segmento CD, possiamo applicare il teorema di Pitagora ai tre triangoli BCD, BCH e BHD ed ottenere il seguente sistema

$$\begin{cases} \overline{BH}^2 = \overline{BC}^2 - \overline{HC}^2 \\ \overline{BH}^2 = \overline{BD}^2 - \overline{DH}^2 \\ \overline{BD}^2 = \overline{DC}^2 - \overline{BC}^2 \end{cases} \tag{1.1}$$

da cui, sostituendo i valori noti,

$$\begin{cases} \overline{BH}^2 = \overline{BC}^2 - (2,5)^2 \\ \overline{BH}^2 = \overline{BD}^2 - (9,5)^2 \\ \overline{BD}^2 = (12)^2 - \overline{BC}^2 \end{cases}$$

che, come si osserva, è un sistema lineare di tre equazioni nelle tre variabili $\overline{BH}^2$, $\overline{BC}^2$ e $\overline{BD}^2$. Allora, per risolvere il problema, sarà sufficiente risolvere detto sistema con uno dei metodi ben noti agli studenti (ad esempio, per sostituzione), senza aver bisogno di conoscere altro della geometria euclidea che il teorema di Pitagora (oltre, naturalmente, la definizione dei poligoni).

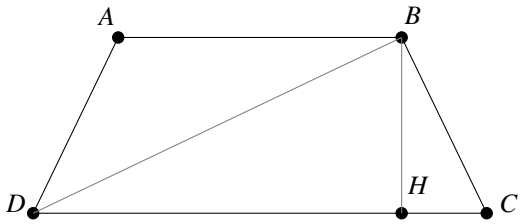

Figura 1.2: La soluzione a livello di scuola superiore: è sufficiente applicare il teorema di Pitagora a tre triangoli.

Certamente, il livello di difficoltà, è *sostanzialmente* diverso nei due casi. Ossia, è naturale aspettarsi che, procedendo con gli studi, gli studenti acquisiscano strumenti via via più potenti per la soluzione di problemi, ma quello che vogliamo sottolineare è che, mentre la soluzione basata sulla sola geometria euclidea richiede una catena di ragionamenti e deduzioni, una volta impostato il sistema di equazioni *si giunge alla soluzione del problema mediante una serie di calcoli che potrebbero essere eseguiti anche da una macchina*.
Ancora più notevolmente, osserviamo che l'impostazione del sistema di equazioni non è altro che la *riscrittura (ragionata) dei dati del problema*. Ossia che

la teoria dei sistemi di equazioni è un linguaggio che ci permette di rappresentare i dati dei nostri problemi in modo tale che, una volta descritto un problema in tale linguaggio, la sua soluzione diventa un procedimento pressoché automatico.

O, per dirla con le parole di Leibniz, impostiamo il sistema e, poi, *calcolemus!*

1.4 Problemi, istanze e algoritmi

Informalmente, ora, introduciamo il concetto di problema così come lo conosciamo: un insieme di dati ed una domanda. Tale concetto sarà formalizzato più avanti in questo corso.

Definizione 1.1: Un problema è definito da un insieme di istanze, per ciascuna delle quali è necessario trovare una soluzione che rispetti i vincoli del problema. Una istanza del problema è un insieme di dati. Una soluzione di una istanza del problema è un insieme di valori, correlati ai dati, che rispettano l'insieme dei vincoli del problema.

Segue dalla Definizione 1.1 che l'esempio che nel Paragrafo 1.3 abbiamo chiamato problema era, in realtà, un esempio di *istanza* di un problema. Il problema propriamente detto corrispondente ha la forma seguente:

Problema. Si consideri un trapezio isoscele *ABCD* avente la diagonale perpendicolare al lato obliquo, di cui sono note la lunghezza della base minore *AB* e la lunghezza della base maggiore *CD*. Calcolare il perimetro e l'area del trapezio.

Tipicamente, quando ci si trova di fronte ad un problema, quello che si vuole fare è *risolverlo*. Ma cosa significa risolvere un problema? Ovviamente, data una qualunque sua istanza, trovare una sua soluzione.
Si consideri, ora, il sistema di equazioni (1.1) che abbiamo impostato per risolvere il nostro problema sul trapezio: se assumiamo che

$$\overline{DC}^2, \qquad \overline{HC}^2 = \left(\frac{\overline{DC}-\overline{AB}}{2}\right)^2, \qquad \overline{DH}^2 = \left(\overline{DC}-\overline{HC}\right)^2$$

siano *parametri* del problema e risolviamo *quel* sistema di equazioni *in funzione* di tali parametri, allora avremo trovato una soluzione non solo per l'istanza data del problema, ma per tutte le istanze del problema. O, in altri termini, avremo trovato una *soluzione per il problema*.

Osserviamo esplicitamente che la soluzione di un problema esiste anche se non la troviamo (un qualsiasi trapezio ha un'area ed un perimetro anche se l'idea di calcolarli non ci sfiora nemmeno): in altri termini, la soluzione di un problema è implicitamente descritta dalla struttura stessa dell'insieme delle sue istanze. Trovare una soluzione di un problema significa dunque

estrarre informazione nascosta (implicita) dall'informazione in nostro possesso (esplicita).

Come abbiamo visto nel Paragrafo 1.3, per ogni problema, posso trovare *metodi di soluzione* differenti o, nel linguaggio informatico, *algoritmi differenti*.

Definizione 1.2: Un *algoritmo* è la descrizione di una sequenza di *passi elementari* che permettono ad un qualche esecutore di calcolare una soluzione di un problema a partire da una qualsiasi sua istanza.

Osserviamo che il concetto di algoritmo è strettamente connesso a quello di *esecutore* dello stesso. In questo senso, resta da chiarire cosa intendiamo con *passo elementare*. In generale, con il termine "passo elementare" ci riferiamo ad una operazione il cui significato sia immediatamente comprensibile a chi dovrà eseguire effettivamente l'algoritmo. Ad esempio, se l'esecutore è uno studente di terza media, il seguente *non* è un passo elementare di un algoritmo per il problema del trapezio:

calcola la soluzione del sistema (1.1).

Il resto di questa dispensa sarà dedicato proprio a chiarire il concetto di *passo elementare*.

1.5 Modelli di calcolo: la Macchina di Turing

Consideriamo il seguente classico problema

Ordinamento degli elementi di un insieme. Si consideri un insieme di n contenitori numerati da 0 a $n-1$, ossia, $c[0], c[1], \ldots, c[n-1]$, in ciascuno dei quali è contenuto un numero intero (per i più esperti, stiamo considerando un array c di interi). Vogliamo scambiare i contenuti dei contenitori in modo tale che, al termine del procedimento, i numeri contenuti nei contenitori siano ordinati in maniera non decrescente, ossia: il contenitore $c[n-1]$ contenga il numero più grande fra tutti quelli contenuti in $c[0], c[1], \ldots, c[n-1]$, il contenitore $c[n-2]$ contenga il secondo numero più grande, e, in generale, per $i = 0, \ldots, n-1$, il contenitore $c[n-i]$ contenga l'i-esimo numero più grande fra tutti quelli contenuti in $c[0], c[1], \ldots, c[n-1]$.

Per risolvere questo problema possiamo utilizzare algoritmi differenti. Nell'algoritmo *Bubble Sort* (ordinamento a bolle) che prendiamo qui in considerazione, l'ordinamento degli oggetti avviene in n fasi distinte: in ogni fase è

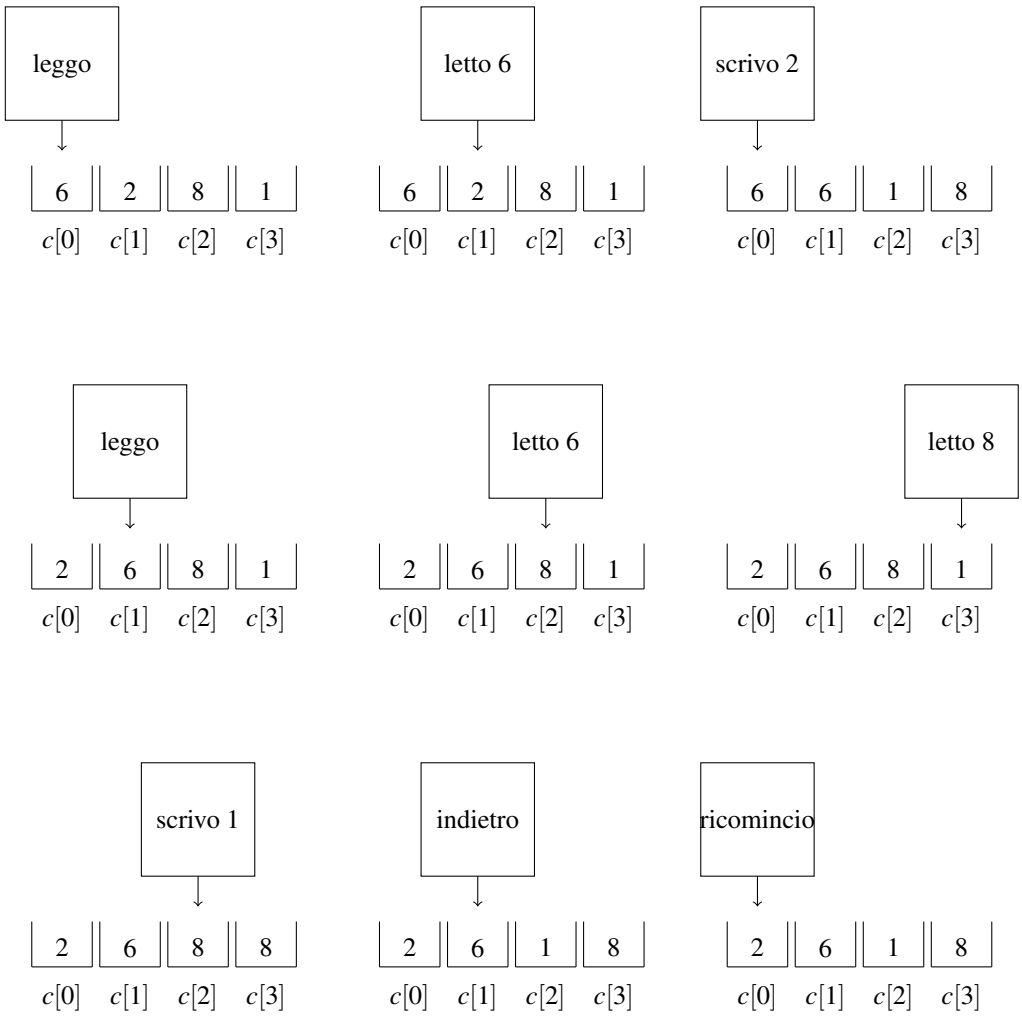

Figura 1.3: Esecuzione di una iterazione del loop esterno ($i = 0$) dell'algoritmo Bubble Sort con input 4 contenitori contenenti i numeri $6, 2, 8, 1$.

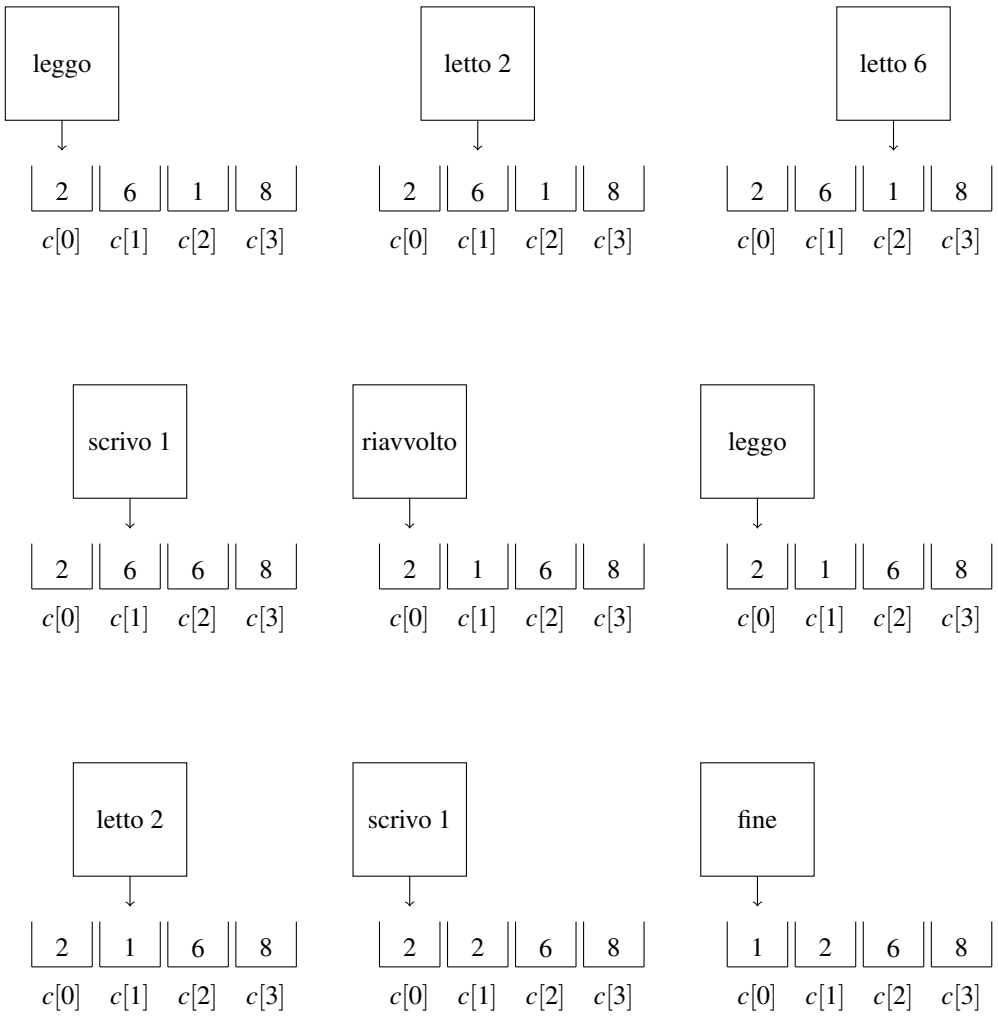

Figura 1.4: Continuazione dell'esecuzione dell'algoritmo Bubble Sort con input 4 contenitori contenenti i numeri $6, 2, 8, 1$.

Dati:	i contenitori $c[0], c[1], ..., c[n-1]$, contenenti ciascuno un numero intero.
Risultato:	una permutazione dei contenuti dei contenitori, in modo tale che risulti: il numero contenuto nel contenitore $c[i]$ è maggiore o uguale al numero contenuto nel contenitore $c[i-1]$, per ogi $i = 1, \ldots, n-1$.
1	impostiamo $i = 1$;
2	iniziamo la fase i:
3	impostiamo $j = 0$;
4	confrontiamo i numeri contenuti in $c[j]$ e in $c[j+1]$:
5	se il numero contenuto in $c[j+1]$ è più piccolo di quello contenuto in $c[j]$ allora li scambiamo di posto;
6	incrementiamo di 1 il valore di j (ossia, poni $j = j+1$) e, se $j \leq n-i-1$, torniamo al punto 4;
7	fine fase i: ora sappiamo che l'i-esimo numero più grande è contenuto in $c[n-i]$;
8	incrementiamo di 1 il valore di i (ossia, poniamo $i = i+1$) e, se $i < n-1$, iniziamo una nuova fase tornando al punto 2.

Tabella 1.1: Algoritmo `A:BubbleSort`.

garantito che almeno un numero sia inserito nel contenitore corretto. Più precisamente, durante la fase i è garantito che l'i-esimo numero più grande sia inserito nel contenitore $c[n-i]$.

In particolare, durante ciascuna fase dell'algoritmo *Bubble Sort*, per ogni coppia di contenitori consecutivi, vengono confrontati i loro contenuti e, se non sono nell'ordine corretto, essi vengono scambiati. In questo modo, durante la prima fase il numero più grande viene posizionato nel contenitore $c[n-1]$ e, in generale, durante la fase i l'i-esimo elemento più grande viene posizionato nel contenitore $c[n-i]$. Possiamo, quindi, descrivere la fase i nel modo seguente: per $j = 0, \ldots, n-i-1$, confronta i numeri contenuti in $c[j]$ e in $c[j+1]$ e, se il numero contenuto in $c[j+1]$ è minore del numero contenuto in $c[j]$, scambiali.

L'algoritmo *Bubble Sort* è descritto formalmente in Tabella 1.1.

L'algoritmo Bubble Sort è stato descritto utilizzando il linguaggio naturale e, quindi, assumendo che il suo esecutore fosse un essere umano. Ma è possibile astrarre dalle caratteristiche dell'esecutore per descrivere il processo di risoluzione di un problema?
Il matematico inglese Alan Turing (1912-1954), nell'intento di realizzare il sogno di Leibniz, considerò proprio questa questione, ossia, tentò di comprendere quali fossero le caratteristiche del processo di soluzione di un problema. Egli si accorse che tale processo è *sempre una sequenza di operazioni molto semplici* ciascuna delle quali

1) dipende da una porzione molto piccola dei dati del problema e

2) il cui esito dipende dal tipo di operazione che si sta eseguendo - o, in altri termini, dallo *stato* in cui si trova l'esecutore.

Sulla base delle precedenti osservazioni, Alan Turing definì il *modello di calcolo* che porta il suo nome: la macchina di Turing.

Introduciamo la macchina di Turing provando a pensare a come un qualunque esecutore eseguirebbe l'algoritmo Bubble Sort per ordinare i numeri contenuti in 6 contenitori. Supponiamo che i numeri siano scritti con il gesso su piccole lavagne contenute nei contenitori e che l'esecutore abbia una *memoria* (molto) limitata: esso è in grado di ricordare un solo numero alla volta. Inizialmente, l'esecutore andrebbe ad esaminare il numero contenuto nel primo contenitore e poi, *ricordando tale numero*, andrebbe ad esaminare il numero contenuto nel secondo contenitore: se tale numero fosse più piccolo di quello che *è nella sua memoria* allora:

- scriverebbe sulla lavagna del secondo contenitore il numero che si trova nella sua memoria, dimenticandolo e memorizzando, allo stesso tempo, il numero che vi era scritto in precedenza;
- tornerebbe al primo contenitore scrivendo sulla lavagna il numero che si trova nella sua memoria (cancellando il precedente contenuto);

- si riportebbe in prossimità del secondo contenitore, leggendone (e memorizzando) il contenuto per poterlo confrontare con il contenuto del terzo contenitore e ripetere il procedimento.

Se, invece, il numero contenuto nel primo contenitore non fosse più grande di quello contenuto nel primo contenitore, allora l'esecutore *memorizzerebbe* il contenuto del secondo contenitore (*dimenticando quello che aveva memorizzato in precedenza*) e passerebbe ad esaminare il terzo contenitore. Una volta esaminato l'ultimo contenitore, il numero più grande avrebbe raggiunto tale ultimo contenitore e l'esecutore dovrebbe ricominciare ad esaminare i contenitori dall'inizio. L'esecuzione dell'algoritmo Bubble Sort su un input di 4 contenitori il cui contenuto è, inizialmente, nell'ordine $6, 2, 8, 1$ è illustrato nelle Figure 1.3 e 1.4.

In riferimento alle osservazioni di un esecutore di algoritmi individuate da Turing, nell'esempio dell'algoritmo Bubble Sort le operazioni possibili sono cinque: lettura del numero contenuto in un contenitore e sua memorizzazione (operazione *leggi*), lettura del numero contenuto in un contenitore e confronto fra esso e il numero in memoria (operazione *confronta*), scambio fra il numero memorizzato e quello nel contenitore (operazione *scambia*), scrittura del numero memorizzato nel contenitore (operazione *scrivi*) e ritorno al primo contenitore per iniziare una nuova sequenza di operazioni (operazione *riavvolgi*). Per eseguire qualsiasi passo dell'algoritmo è necessario conoscere soltanto *al più due numeri della sequenza da ordinare* e quale delle cinque operazioni elencate poc'anzi deve essere eseguita in quel passo (*stato* dell'esecutore).

Le caratteristiche dell'esecutore che abbiamo appena descritto corrispondono (con qualche importante eccezione) al modello di calcolo *Macchina di Turing*: un dispositivo di calcolo dotato di una *memoria di lavoro ad accesso sequenziale* (nel nostro esempio, i contenitori), in grado di leggere e scrivere dati da/su tale memoria; ciò che viene scritto sulla memoria di lavoro dipende da ciò che vi era stato letto in precedenza (nel nostro esempio, ciò che l'esecutore ricordava). Abbiamo già osservato come il nostro esecutore avesse una memoria (molto) limitata: anche la memoria di controllo della macchina di Turing, ossia la quantità di informazioni che gli occorrono per decidere quali azioni intraprendere, è limitata: come chiariremo fra breve, essa ha dimensione costante. Nel seguito definiremo più formalmente la macchina di Turing.

Definizione 1.3: Sia Σ un *alfabeto* finito e Q un insieme finito di *stati* nel quale distinguiamo uno *stato iniziale* q_0 ed un insieme di *stati finali* Q_F. Una *Macchina di Turing* T sull'alfabeto Σ e sull'insieme di stati Q è un dispositivo di calcolo dotato di

- una unità di controllo che, ad ogni istante della computazione, può trovarsi in uno qualsiasi degli stati in Q,
- di un nastro di lettura/scrittura, di lunghezza infinita, suddiviso in celle indicizzate contenenti, ciascuna, un simbolo in Σ oppure il carattere $\Box$ (cella vuota), e alle quali si può accedere sequenzialmente,
- di una testina di lettura/scrittura che, ad ogni istante della computazione, è posizionata su una cella del nastro,
- un programma, ossia, un insieme P di quintuple del tipo

  $$\langle q_1, a_1, a_2, q_2 m \rangle$$

  in cui $q_1 \in Q - Q_F$, $q_2 \in Q$, $a_1, a_2 \in \Sigma$, e $m \in \{sinistra, destra, ferma\}$, e che ha il significato seguente: *se la testina legge nella cella sulla quale si trova il simbolo a_1 e l'unità di controllo si trova nello stato q_1, allora la testina scrive il simbolo a_2 nella cella sulla quale si trova, poi lo stato interno dell'unità di controllo diventa q_2, infine la testina si sposta di una cella nella direzione specificata da m.*

Data una parola x (ossia, una sequenza finita di elementi di Σ), informalmente, una *computazione* $T(x)$ della macchina T sull'input x è l'applicazione delle quintuple in P ad x partendo dal primo simbolo di x con l'unità di controllo che si trova nello stato iniziale q_0 di T.

L'esecuzione di una singola quintupla di una macchina di Turing è, appunto, un *passo elementare di calcolo*.

Chiariamo questi concetti con un semplice esempio. Esempi più complessi saranno presentati nei paragrafi che seguono.

Esempio. Ricordiamo che una parola è *palindroma* se rimane invariata leggendola da sinistra a destra o viceversa: ad esempio, le parole "onorarono" e "ingegni" sono palindrome.

Sia $\Sigma = \{a,b\}$. Vogliamo definire una macchina di Turing che *decida* se una parola data in input è palindroma. Tale decisione verrà codificata negli stati finali: vogliamo, cioè, che la nostra macchina di Turing termini la computazione nello stato finale q_{pal} se la parola in input è palindroma, termini nello stato $q_{non-pal}$ se la parola in input non è palindroma. Pertanto, definiamo $Q_F = \{q_{pal}, q_{non-pal}\}$.
La nostra macchina opera nel modo seguente: a partire dallo stato iniziale, legge il primo carattere della parola, lo cancella (sovrascrivendolo con un $\Box$) e, *ricordandolo* (ossia, entrando in q_a se ha letto a, o nello stato q_b se ha letto s_b), sposta la testina a destra fino a raggiungere l'ultimo carattere della parola (ossia, raggiunge il primo $\Box$ a destra e torna indietro di una posizione). Se l'ultimo carattere è uguale a quello che sta ricordando (ossia, se è nello stato q_a e legge a oppure se è nello stato q_b e legge b) allora torna sul primo carattere a sinistra che non ha ancora cancellato e ricomincia il procedimento, altrimenti conclude che la parola non è palindroma. Se riesce a cancellare tutte le coppie di caratteri della parola, allora può concludere che la parola è palindroma. In dettaglio, le quintuple che realizzano questo compito sono le seguenti:

$\langle q_0, a, \Box, q_a, destra\rangle$ $\langle q_0, b, \Box, q_b, destra\rangle$ $\langle q_0, \Box, \Box, q_{pal}, ferma\rangle$

$\langle q_a, a, a, q_a, destra\rangle$ $\langle q_a, b, b, q_a, destra\rangle$ $\langle q_a, \Box, \Box, q_a^1, sinistra\rangle$

$\langle q_b, a, a, q_b, destra\rangle$ $\langle q_b, b, b, q_b, destra\rangle$ $\langle q_b, \Box, \Box, q_b^1, sinistra\rangle$

$\langle q_a^1, a, \Box, q_{indietro}, sinistra\rangle$ $\langle q_a^1, b, b, q_{non-pal}, ferma\rangle$ $\langle q_a^1, \Box, \Box, q_{pal}, ferma\rangle$

$\langle q_b^1, a, a, q_{non-pal}, ferma\rangle$ $\langle q_b^1, b, \Box, q_{indietro}, sinistra\rangle$ $\langle q_b^1, \Box, \Box, q_{pal}, ferma\rangle$

$\langle q_{indietro}, a, a, q_{indietro}, sinistra\rangle$ $\langle q_{indietro}, b, b, q_{indietro}, sinistra\rangle$ $\langle q_{indietro}, \Box, \Box, q_0, destra\rangle$

1.6 Una macchina di Turing che somma due numeri interi

Vogliamo definire una macchina di Turing che, presi in input due numeri n e m espressi in notazione binaria, calcola il valore $n+m$.

Sia $\Sigma = \{0,1,+,-,*\}$. Assumiamo che, all'inizio della computazione, l'input sia memorizzato sul nastro, a partire dalla cella 0, nella forma: '$-$' seguito da $[n]_{bin}$ seguito dal carattere '$+$' seguito da $[m]_{bin}$ seguito da '$-$' (dove $[n]_{bin}$ e $[m]_{bin}$ indicano, rispettivamente, la rappresentazione binaria di n e di m). Assumiamo anche che la testina sia posizionata sul primo carattere di $[n]_{bin}$.

Inoltre, decidiamo di memorizzare il risultato nella porzione di nastro ad indirizzi negativi, con la cifra meno significativa nella cella di indirizzo -1 e di utilizzare la cella a destra dell'ultimo carattere '$-$' per memorizzare il valore del riporto.

1.6.1 Rappresentazione di n e m con lo stesso numero di caratteri

Per semplicità, iniziamo a descrivere una macchina di Turing T_F che calcola la somma di n e m quando tali valori sono rappresentati con lo stesso numero di caratteri: ad esempio, se $n = 5$ e $m = 8$, allora $[n]_{bin} = 0101$ e $[m]_{bin} = 1000$. In questo caso, l'algoritmo implementato dalla macchina T_F che andiamo a definire è il seguente:

1. nello stato iniziale q_0, sposta la testina a destra fino a quando non raggiungi l'ultimo carattere di $[n]_{bin}$ (ossia, la sua cifra meno significativa), ricorda il carattere letto (ossia, entra nello stato q_0^{letto} oppure q_1^{letto}) e cancellalo (ossia, sostituiscilo con il carattere '$*$'):

   $\langle q_0, 0, 0, q_0, destra\rangle$ $\langle q_0, 1, 1, q_0, destra\rangle$ $\langle q_0, +, +, q_1, sinistra\rangle$

   $\langle q_1, 0, *, q_0^{letto}, destra\rangle$ $\langle q_1, 1, *, q_1^{letto}, destra\rangle$ $\langle q_1, *, *, q_1, sinistra\rangle$

2. sposta la testina a destra fino a raggiungere l'ultimo carattere di $[m]_{bin}$, ricordalo (ossia, entra nello stato q_{00}^{letto} oppure q_{01}^{letto} oppure q_{11}^{letto}) e cancellalo,

$$\langle q_0^{letto}, x, x, q_0^{letto}, destra\rangle \quad \forall x \in \Sigma - \{-\} \qquad \langle q_0^{letto}, -, -, q_0^{ind}, sinistra\rangle$$

$$\langle q_1^{letto}, x, x, q_1^{letto}, destra\rangle \quad \forall x \in \Sigma - \{-\} \qquad \langle q_1^{letto}, -, -, q_1^{ind}, sinistra\rangle$$

$$\langle q_0^{ind}, 0, -, q_{00}^{letto}, destra\rangle \qquad \langle q_0^{ind}, 1, -, q_{01}^{letto}, destra\rangle$$

$$\langle q_1^{ind}, 0, -, q_{01}^{letto}, destra\rangle \qquad \langle q_1^{ind}, 1, -, q_{11}^{letto}, destra\rangle$$

3. sposta la testina a destra sul carattere di riporto, modificalo opportunamente e ricorda il carattere della somma (ossia, entra nello stato q_0^{scrivi} oppure q_1^{scrivi}):

$$\langle q_{00}^{letto}, -, -, q_{00}^{letto} destra\rangle \qquad \langle q_{01}^{letto}, -, -, q_{01}^{letto} destra\rangle \qquad \langle q_{11}^{letto}, -, -, q_{11}^{letto} destra\rangle$$

$$\langle q_{00}^{letto}, 0, 0, q_0^{scrivi} sinistra\rangle \qquad \langle q_{01}^{letto}, 0, 0, q_1^{scrivi} sinistra\rangle \qquad \langle q_{11}^{letto}, 0, 1, q_0^{scrivi} sinistra\rangle$$

$$\langle q_{00}^{letto}, 1, 0, q_1^{scrivi} sinistra\rangle \qquad \langle q_{01}^{letto}, 1, 1, q_0^{scrivi} sinistra\rangle \qquad \langle q_{11}^{letto}, 1, 1, q_1^{scrivi} sinistra\rangle$$

$$\langle q_{00}^{letto}, \square, 0, q_0^{scrivi} sinistra\rangle \qquad \langle q_{01}^{letto}, \square, 0, q_1^{scrivi} sinistra\rangle \qquad \langle q_{11}^{letto}, \square, 1, q_0^{scrivi} sinistra\rangle$$

ove queste ultime tre quintuple gestiscono il caso in cui vengono sommati i caratteri meno significativi di $[n]_{bin}$ e $[m]_{bin}$ e, quindi, il riporto non è ancora stato inizializzato;

4. sposta la testina sul primo $\square$ a sinistra, scrivi il carattere calcolato e vai nello stato q_2:

$$\langle q_0^{scrivi}, x, x, q_0^{scrivi}, sinistra\rangle \quad \forall x \in \Sigma \qquad \langle q_0^{scrivi}, \square, 0, q_2, destra\rangle$$

$$\langle q_1^{scrivi}, x, x, q_1^{scrivi}, sinistra\rangle \quad \forall x \in \Sigma \qquad \langle q_1^{scrivi}, \square, 1, q_2, destra\rangle$$

5. sposta la testina a destra fino a posizionarla sul primo carattere '$-$' e poi torna nello stato q_0 spostando ancora a destra di una posizione la testina

$$\langle q_2, x, x, q_2, destra\rangle \quad \forall x \in \{0,1\} \qquad \langle q_2, -, -, q_0, destra\rangle$$

6. a questo punto, se il carattere letto dalla testina è '$*$', allora la scansione di $[n]_{bin}$ (e di $[m]_{bin}$) è terminata: per completare il calcolo della somma è necessario aggiungere il valore del riporto, se diverso da 0, e poi entrare nello stato finale q_{finale}:

$$\langle q_0, *, *, q_3, destra\rangle$$

$$\langle q_3, *, *, q_3, destra\rangle \qquad \langle q_3, +, +, q_3, destra\rangle \qquad \langle q_3, -, -, q_3, destra\rangle$$

$$\langle q_3, \square, \square, q_{finale}, ferma\rangle \qquad \langle q_3, 0, 0, q_{finale}, ferma\rangle \qquad \langle q_3, 1, 1, q_4, sinistra\rangle$$

$$\langle q_4, -, -, q_4, sinistra\rangle \qquad \langle q_4, +, +, q_4, sinistra\rangle \qquad \langle q_4, *, *, q_4, sinistra\rangle$$

$$\langle q_4, 0, 0, q_4, sinistra\rangle \qquad \langle q_4, 1, 1, q_4, sinistra\rangle \qquad \langle q_4, \square, 1, q_{finale}, ferma\rangle$$

Osserviamo che il funzionamento della macchina T_F è definito solo quando l'input è scritto sul nastro rispettando i vincoli precedentemente descritti: $[n]_{bin}$ e $[m]_{bin}$ sono codificati in binario utilizzando lo stesso numero di caratteri, la stringa è scritta sul nastro nella forma $-[n]_{bin} + [m]_{bin} -$. Se l'input non soddisfa tali vincoli il comportamento della macchina non è definito. Osserviamo che è questo il concetto di *precondizione*: garantiamo il funzionamento di una porzione di codice solo se la precondizione è verificata.

1.6.2 Rappresentazione di n e m con numero arbitrario di caratteri

Assumiamo ora che i valori n e m sono rappresentati in binario utilizzando il minor numero di caratteri possibile: così, se $n = 5$ e $m = 8$, allora rappresentiamo $[n]_{bin} = 101$ e $[m]_{bin} = 1000$. La macchina T_V che calcola $n+m$ in queste ipotesi differisce da T_F soltanto per le questioni elencate di seguito.

- Se il numero di caratteri $|[n]_{bin}|$ di $[n]_{bin}$ è maggiore del numero di caratteri $|[m]_{bin}|$ di $[m]_{bin}$ allora, dopo aver letto l'ultimo carattere di $[n]_{bin}$ ed essere entrata nello stato q_0^{letto} o q_1^{letto}, al punto 2. sopra, che descrive T_F, T_V potrebbe non trovare alcun carattere di $[m]_{bin}$; allora, è necessario sommare ciascuno dei rimanenti caratteri di $[n]_{bin}$ con il riporto (ricalcolando ogni volta il valore del riporto) ed aggiungere i caratteri così calcolati al risultato. In definitiva, *alle istruzioni del punto 2. devono essere aggiunte le seguenti*

  2'. se nello stato q_0^{ind} oppure q_1^{ind} leggi il carattere '+' (e, quindi, tutti i caratteri di $[m]_{bin}$ sono già stati considerati), entra nello stato $q_0^{mfinito}$ o, rispettivamente, $q_1^{mfinito}$ e sposta la testina a destra fino a posizionarla sul carattere del riporto; poi calcola il nuovo carattere di riporto ed il carattere da aggiungere all'output

  $\langle q_0^{ind}, +, +, q_0^{mfinito}, destra\rangle$

  $\langle q_1^{ind}, +, +, q_1^{mfinito}, destra\rangle$

  $\langle q_0^{mfinito}, -, -, q_0^{mfinito}, destra\rangle$

  $\langle q_1^{mfinito}, -, -, q_1^{mfinito}, destra\rangle$

  $\langle q_0^{mfinito}, 0, 0, q_0^{scrivi}, sinistra\rangle$

  $\langle q_0^{mfinito}, 1, 0, q_1^{scrivi}, sinistra\rangle$

  $\langle q_1^{mfinito}, 0, 0, q_1^{scrivi}, sinistra\rangle$

  $\langle q_1^{mfinito}, 1, 1, q_0^{scrivi}, sinistra\rangle$

- Se $|[n]_{bin}| < |[m]_{bin}|$ allora, al punto 7. sopra, che descrive T_F, se la testina legge il carattere '$+$' mentre la macchina è nello stato q_0 non si può concludere che la computazione è terminata, ma è necessario verificare che sia stata terminata la scansione di tutti i caratteri di $[m]_{bin}$. Allora, *il punto 6. deve essere sostituito dal seguente*

  6'. se nello stato q_0 il carattere letto dalla testina è '$*$' allora la scansione di $[n]_{bin}$ è terminata: la macchina entra nello stato $q^{nfinito}$ e verifica se è terminata anche la scansione di $[m]_{bin}$ oppure se deve aggiungere altri caratteri all'output (analogamente alle operazioni descritte al punto 2'.):

  $\langle q_0, *, *, q^{nfinito}, destra\rangle$

  $\langle q^{nfinito}, *, *, q^{nfinito}, destra\rangle$

  $\langle q^{nfinito}, +, +, q^{nfinito}, destra\rangle$

  $\langle q^{nfinito}, -, -, q^{nmfiniti}, destra\rangle$

  $\langle q^{nmfiniti}, -, -, q^{nmfiniti}, destra\rangle$

  $\langle q^{nmfiniti}, 0, 0, q_{finale}, destra\rangle$

  $\langle q^{nmfiniti}, 1, 0, q_1^{scrivi}, destra\rangle$

  $\langle q^{nfinito}, 0, -, q_{00}^{letto}, destra\rangle$

  $\langle q^{nfinito}, 1, -, q_{01}^{letto}, destra\rangle$

  Nelle ultime due istruzioni la macchina entra nello stato q_{00}^{letto} o q_{01}^{letto} in quanto ha scoperto che esistono ancora caratteri di $[m]_{bin}$ ma che non esistono più caratteri di $[n]_{bin}$: il singolo carattere di $[m]_{bin}$ (che dovrà essere sommato al riporto, analogamente al punto 2'.) avrà sempre lo stesso valore della sua somma con 0.

Osserviamo che, come nel caso di T_F, anche T_V opera correttamente solo se sono rispettate le sue precondizioni. Osserviamo anche esplicitamente che una delle precondizioni per il corretto funzionamento di entrambe le macchine è che $|[n]_{bin}| > 0$ e $|[m]_{bin}| > 0$.

1.7 Macchine a più nastri

Le macchine di Turing possono essere definite anche con un numero arbitrario (ma costante) di nastri su ciascuno dei quali opera una diversa testina di lettura/scrittura. Nastri differenti possono essere utilizzati, ad esempio, per contenere porzioni dell'input cui sono associati diversi significati (ad esempio, le macchine di Turing che definiamo in questa sezione utilizzano un nastro per contenere n ed uno per contenere m), oppure un nastro può essere dedicato all'output, o, ancora, uno o più nastri possono essere utilizzati come nastri di lavoro. In ogni caso, come vedremo in questa sezione, l'utilizzo di più nastri, in generale, semplifica il progetto della macchina. Consideriamo macchine di Turing a più nastri di due tipi differenti.

- In una macchina di Turing a *testine solidali*, in ogni istruzione, le celle dei nastri scandite dalle testine di lettura/scrittura hanno il medesimo indirizzo: così, assumendo che all'inizio della computazione le testine siano posizionate sulle celle di indirizzo 0 dei rispettivi nastri, all'istante 1 esse saranno *tutte* posizionate sulle celle di indirizzo $+1$ oppure -1 oppure 0 (dipendentemente dal movimento specificato dalla quintupla eseguita). In generale, se dopo un certo numero di passi le testine sono posizionate sulle celle di indirizzo h, al passo successivo (ricordiamo che un passo corrisponde all'esecuzione di una quintupla) esse saranno *tutte* posizionate sulle celle di indirizzo $h+1$ oppure $h-1$ oppure h.

  Una quintupla di una macchina di Turing a k nastri a testine solidali ha la forma

  $$\langle q_i, \bar{s}_1, \bar{s}_2, q_j, mov \rangle,$$

  dove $\bar{s}_1 = (s_{1_1}, s_{1_2}, \ldots, s_{1_k})$, $\bar{s}_2 = (s_{2_1}, s_{2_2}, \ldots, s_{2_k})$ e $mov \in \{sinistra, ferma, destra\}$. Il significato di una quintupla è il seguente: *se* la macchina è nello stato q_i, la testina 1 legge il simbolo s_{1_1} sul nastro 1, la testina 2 legge il simbolo s_{1_2} sul nastro 2, ..., e la testina k legge il simbolo s_{1_k} sul nastro k, *allora* la testina 1 scrive il simbolo s_{2_1} sul nastro 1, la testina 2 scrive il simbolo s_{2_2} sul nastro 2, ..., la testina k scrive il simbolo s_{2_k} sul nastro k, la macchina entra nello stato q_j e *tutte* le testine eseguono il movimento *mov*.

- In una macchina di Turing a *testine indipendenti*, in seguito all'esecuzione di una quintupla le testina si muovono indipendentemente le une dalle altre. Dunque, dopo l'esecuzione di un certo numero di passi, la posizione di una testina non ha alcuna relazione con la posizione delle altre testine. Una quintupla di una macchina di Turing a k nastri a testine indipendenti ha quindi la forma

  $$\langle q_i, \bar{s}_1, \bar{s}_2, q_j, \overline{mov} \rangle,$$

  dove $\bar{s}_1 = (s_{1_1}, s_{1_2}, \ldots, s_{1_k})$, $\bar{s}_2 = (s_{2_1}, s_{2_2}, \ldots, s_{2_k})$, $\overline{mov} = (mov_1, mov_2, \ldots, mov_k)$ e $mov_h \in \{sinistra, ferma, destra\}$, per $h = 1, \ldots, k$. Il significato di una quintupla è il seguente: *se* la macchina è nello stato q_i, la testina 1 legge il simbolo s_{1_1} sul nastro 1, la testina 2 legge il simbolo s_{1_2} sul nastro 2, ..., e la testina k legge il simbolo s_{1_k} sul nastro k, *allora* la testina 1 scrive il simbolo s_{2_1} sul nastro 1, la testina 2 scrive il simbolo s_{2_2} sul nastro 2, ..., la testina k scrive il simbolo s_{2_k} sul nastro k, la macchina entra nello stato q_j e la testina 1 esegue il movimento mov_1, la testina 2 esegue il movimento mov_2, ... , e la testina k esegue il movimento mov_k.

Risolviamo il problema della somma di due numeri definendo macchine a 3 nastri: il nastro N_1 per l'input n, il nastro N_2 per l'input m, ed il nastro N_3 per l'output. Osserviamo che, avendo dedicato ciascun nastro alla memorizzazione di una sola entità, non abbiamo bisogno di caratteri separatori nell'alfabeto. Sia allora $\Sigma = \{0, 1\}$. Assumiamo che, all'inizio della computazione, l'input sia memorizzato, sui nastri N_1 ed N_2, a partire dalle celle 0 e che le testine siano posizionate sulle celle che contengono i caratteri meno significativi di $[n]_{bin}$ e $[m]_{bin}$, ossia, sulle celle di indirizzo $\max\{|[n]_{bin}|, |[m]_{bin}|\}$.

Al solito, le macchine che andiamo a definire garantiscono il proprio operato a condizione che le precondizioni siano verificate. In questo caso, le precondizioni sono: all'inizio della computazione, entrambe le testine che scandiscono i nastri input leggono un carattere diverso da $\Box$ e che ogni cella del nastro N_3 contiene $\Box$.

1.7.1 Somma di due numeri con una macchina a testine solidali

L'algoritmo implementato dalla macchina T_S è, in questo caso, immediato:

1. nello stato iniziale q_0, leggi i due caratteri di input sui nastri N_1 ed N_2, calcola il primo carattere del risultato e scrivilo sul nastro N_3, calcola il valore del riporto e memorizzalo nel nuovo stato e, infine, sposta le testine a sinistra:

   $\langle q_0, (0,0,\square), (0,0,0), q_{r_0}, sinistra\rangle,$

   $\langle q_0, (0,1,\square), (0,1,1), q_{r_0}, sinistra\rangle,$

   $\langle q_0, (1,0,\square), (1,0,1), q_{r_0}, sinistra\rangle,$

   $\langle q_0, (1,1,\square), (1,1,0), q_{r_1}, sinistra\rangle;$

2. nello stato q_{r_0}, la macchina sa che la somma effettuata al passo precedente non ha generato riporti; pertanto,

   se esistono ancora caratteri da sommare sia su N_1 che su N_2, allora

   $\langle q_{r_0}, (0,0,\square), (0,0,0), q_{r_0}, sinistra\rangle,$

   $\langle q_{r_0}, (0,1,\square), (0,1,1), q_{r_0}, sinistra\rangle,$

   $\langle q_{r_0}, (1,0,\square), (1,0,1), q_{r_0}, sinistra\rangle,$

   $\langle q_{r_0}, (1,1,\square), (1,1,0), q_{r_1}, sinistra\rangle,$

   se rimangono ancora caratteri da sommare solo su N_1, allora

   $\langle q_{r_0}, (0,\square,\square), (0,\square,0), q_{r_0}, sinistra\rangle,$

   $\langle q_{r_0}, (1,\square,\square), (1,\square,1), q_{r_0}, sinistra\rangle,$

   se rimangono ancora caratteri da sommare solo su N_2, allora

   $\langle q_{r_0}, (\square,0,\square), (\square,0,0), q_{r_0}, sinistra\rangle,$

   $\langle q_{r_0}, (\square,1,\square), (\square,1,1), q_{r_0}, sinistra\rangle,$

   se non vi sono più caratteri da sommare, poiché il valore del riporto è 0, allora non viene scritto alcun nuovo carattere sul nastro N_3 e la computazione termina

   $\langle q_{r_0}, (\square,\square,\square), (\square,\square,\square), q_{finale}, sinistra\rangle;$

3. nello stato q_{r_1}, la macchina sa che la somma effettuata al passo precedente ha generato un riporto; pertanto,

   se esistono ancora caratteri da sommare sia su N_1 che su N_2, allora

   $\langle q_{r_1}, (0,0,\square), (0,0,1), q_{r_0}, sinistra\rangle,$

   $\langle q_{r_1}, (0,1,\square), (0,1,0), q_{r_1}, sinistra\rangle,$

$\langle q_{r_1}, (1,0,\square), (1,0,0), q_{r_1}, sinistra\rangle,$

$\langle q_{r_1}, (1,1,\square), (1,1,1), q_{r_1}, sinistra\rangle,$

se rimangono ancora caratteri da sommare solo su N_1, allora

$\langle q_{r_1}, (0,\square,\square), (0,\square,1), q_{r_0}, sinistra\rangle,$

$\langle q_{r_1}, (1,\square,\square), (1,\square,0), q_{r_1}, sinistra\rangle,$

se rimangono ancora caratteri da sommare solo su N_2, allora

$\langle q_{r_1}, (\square,0,\square), (\square,0,1), q_{r_0}, sinistra\rangle,$

$\langle q_{r_1}, (\square,1,\square), (\square,1,0), q_{r_1}, sinistra\rangle,$

se non vi sono più caratteri da sommare, poiché il valore del riporto è 1, allora viene scritto il carattere '1' sul nastro N_3 e la computazione termina

$\langle q_{r_1}, (\square,\square,\square), (\square,\square,1), q_{finale}, sinistra\rangle.$

1.7.2 Somma di due numeri con una macchina a testine indipendenti

L'algoritmo implementato dalla macchina T_I è identico a quello del caso precedente. Lo ripetiamo al solo scopo di illustrare le differenze nella struttura delle quintuple:

1. nello stato iniziale q_0, leggi i due caratteri di input sui nastri N_1 ed N_2, calcola il primo carattere del risultato e scrivilo sul nastro N_3, calcola il valore del riporto e memorizzalo nel nuovo stato e, infine, sposta le testine a sinistra:

   $\langle q_0, (0,0,\square), (0,0,0), q_{r_0}, (sinistra, sinistra, sinistra)\rangle,$

   $\langle q_0, (0,1,\square), (0,1,1), q_{r_0}, (sinistra, sinistra, sinistra)\rangle,$

   $\langle q_0, (1,0,\square), (1,0,1), q_{r_0}, (sinistra, sinistra, sinistra)\rangle,$

   $\langle q_0, (1,1,\square), (1,1,0), q_{r_1}, (sinistra, sinistra, sinistra)\rangle;$

2. nello stato q_{r_0}, la macchina sa che la somma effettuata al passo precedente non ha generato riporti; pertanto,

   se esistono ancora caratteri da sommare sia su N_1 che su N_2, allora

   $\langle q_{r_0}, (0,0,\square), (0,0,0), q_{r_0}, (sinistra, sinistra, sinistra)\rangle,$

   $\langle q_{r_0}, (0,1,\square), (0,1,1), q_{r_0}, (sinistra, sinistra, sinistra)\rangle,$

   $\langle q_{r_0}, (1,0,\square), (1,0,1), q_{r_0}, (sinistra, sinistra, sinistra)\rangle,$

   $\langle q_{r_0}, (1,1,\square), (1,1,0), q_{r_1}, (sinistra, sinistra, sinistra)\rangle,$

   se rimangono ancora caratteri da sommare solo su N_1, allora

   $\langle q_{r_0}, (0,\square,\square), (0,\square,0), q_{r_0}, (sinistra, sinistra, sinistra)\rangle,$

$$\langle q_{r_0}, (1, \square, \square), (1, \square, 1), q_{r_0}, (sinistra, sinistra, sinistra)\rangle,$$

se rimangono ancora caratteri da sommare solo su N_2, allora

$$\langle q_{r_0}, (\square, 0, \square), (\square, 0, 0), q_{r_0}, (sinistra, sinistra, sinistra)\rangle,$$

$$\langle q_{r_0}, (\square, 1, \square), (\square, 1, 1), q_{r_0}, (sinistra, sinistra, sinistra)\rangle,$$

se non vi sono più caratteri da sommare, poiché il valore del riporto è 0, allora non viene scritto alcun nuovo carattere sul nastro N_3 e la computazione termina

$$\langle q_{r_0}, (\square, \square, \square), (\square, \square, \square), q_{finale}, (sinistra, sinistra, sinistra)\rangle;$$

3. nello stato q_{r_1}, la macchina sa che la somma effettuata al passo precedente ha generato un riporto; pertanto,

   se esistono ancora caratteri da sommare sia su N_1 che su N_2, allora

   $$\langle q_{r_1}, (0, 0, \square), (0, 0, 1), q_{r_0}, (sinistra, sinistra, sinistra)\rangle,$$

   $$\langle q_{r_1}, (0, 1, \square), (0, 1, 0), q_{r_1}, (sinistra, sinistra, sinistra)\rangle,$$

   $$\langle q_{r_1}, (1, 0, \square), (1, 0, 0), q_{r_1}, (sinistra, sinistra, sinistra)\rangle,$$

   $$\langle q_{r_1}, (1, 1, \square), (1, 1, 1), q_{r_1}, (sinistra, sinistra, sinistra)\rangle,$$

   se rimangono ancora caratteri da sommare solo su N_1, allora

   $$\langle q_{r_1}, (0, \square, \square), (0, \square, 1), q_{r_0}, (sinistra, sinistra, sinistra)\rangle,$$

   $$\langle q_{r_1}, (1, \square, \square), (1, \square, 0), q_{r_1}, (sinistra, sinistra, sinistra)\rangle,$$

   se rimangono ancora caratteri da sommare solo su N_2, allora

   $$\langle q_{r_1}, (\square, 0, \square), (\square, 0, 1), q_{r_0}, (sinistra, sinistra, sinistra)\rangle,$$

   $$\langle q_{r_1}, (\square, 1, \square), (\square, 1, 0), q_{r_1}, (sinistra, sinistra, sinistra)\rangle,$$

   se non vi sono più caratteri da sommare, poiché il valore del riporto è 1, allora viene scritto il carattere ‘1’ sul nastro N_3 e la computazione termina

   $$\langle q_{r_1}, (\square, \square, \square), (\square, \square, 1), q_{finale}, (sinistra, sinistra, sinistra)\rangle.$$

Nota

Il Paragrafo 1.2 è liberamente tratto da M. Davis :“Il calcolatore universale”, che invito tutti a leggere.